DESIGN, FABRICATION, AND IMPLEMENTATION OF AN EMBEDDED FLIGHT COMPUTER IN SUPPORT OF THE IONOSPHERIC-THERMOSPHERIC SCANNING PHOTOMETER FOR ION-NEUTRAL STUDIES CUBESAT MISSION

by

Matthew Lee Handley

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

of

Master of Science

in

Electrical Engineering

MONTANA STATE UNIVERSITY
Bozeman, Montana

July, 2017

ACKNOWLEDGEMENTS

I would like thank all the members of the Space Science and Engineering Laboratory at Montana State University for their dedicated work on this mission, as well as their technical advice and assistance in designing, building, and testing of this project.

Funding Acknowledgment

This work was kindly supported by NSF Grant Number AGS-1445450 to Montana State University.

TABLE OF CONTENTS

TABLE OF CONTENTS – CONTINUED

LIST OF TABLES

Table Page

LIST OF FIGURES

LIST OF FIGURES – CONTINUED

Figure Page

ABSTRACT

As society increasingly relies on space-based assets for everything from GPS-based directions and global communications to human-driven research on the ISS, our understanding of space weather becomes vital. Timely predictions of a solar storm's impact on the ionosphere are imperative to safing these assets before damaging storms hit, while minimizing downtime during lighter storms. The topside transition region (TTR) is a global boundary where the concentration of O^+ significantly decreases due to charge exchange with H^+ and He^+ from the thermosphere, as well as protons and neutral atomic oxygen from the plasmasphere. When high-energy electrons in the ionosphere intercept O^+ ions, they combine and release photons at 135.6-nm.

The Ionospheric-Thermospheric Scanning Photometer for Ion-Neutral Studies (IT-SPINS) mission will provide 135.6-nm nightglow measurements from a 3U CubeSat equipped with a high-sensitivity UV photometer. The CubeSat will spin about orbit normal, sweeping its photometer field of view through the ionosphere. Ground-based post processing will yield 2D altitude/in-track images of O^+ density, providing weighting parameters for models of the TTR. This low-earth orbit (LEO) small satellite mission is a collaboration between the John Hopkins University Applied Physics Laboratory, SRI International, and Montana State University (MSU).

This research describes the design, fabrication, and implementation of the space flight computer (SFC) hardware and software responsible for handling all commands, telemetry, and scientific data required by this National Science Foundation (NSF) funded mission. The SFC design balances requirements derived from the mission objectives while leveraging heritage hardware and software from MSU's many successful CubeSat missions (HRBE, FIREBIRD, FIREBIRD-II) and payloads (EPISEM) [1–3]. This low-power (100 mW) embedded computer features dual 16-bit PIC microcontrollers running at 16 MHz with only 96 kB of RAM and runs the μC/OS-II real-time operating system (RTOS). The SFC also includes a TCXO-driven mission elapsed time clock with ± 2 ppm temperatures stability, a 1 GB NAND flash for data storage, and interfaces to all other subsystems in the satellite.

The SFC has passed all standalone testing. It is currently being integrated and tested with the entire IT-SPINS spacecraft and is planned to fly in late 2018.

1

INTRODUCTION

Space Weather

Space weather is a broad term used to describe the temporal and spatial variation of charged particles and plasmas throughout the solar system and around the Earth's magentosphere, ionosphere, and thermosphere. Many of these charged particle are released from the Sun and travel outward through the solar system, forming what is known as the solar wind. This stream of charged particles interact with the planets' magnetic fields, which can trap and release these particles. Figure 1.1 shows the relative concentration of charged particles trapped along Earth's magnetic field lines. The field lines form the particles into two toroidal regions, known as the inner and outer Van Allen radiation belts.

Humans have long been aware of one of the more visible effects of the radiation belts, the aurora, also known as the northern and southern lights. Auroras occur when excess charged particles are trapped in the radiation belts and precipitate down along the field lines, entering the Earth's upper atmosphere, generally near the poles. The particles interact with the atmosphere, and disperse their excess energy as visible light. However, in today's technological age, the high energy particles can have much more damaging effects on our infrastructure.

Electronic semiconductors, which are used in virtually every electronic device from cellphones to satellites and airplane avionics, are susceptible to two main types of disturbances from ionizing radiation Total Ionizing Dose (TID) and Single Event Effects (SEE). TID refers to the total dose of radiation a semiconductor device is exposed to. Over time, radiation breaks down the semiconductor material,

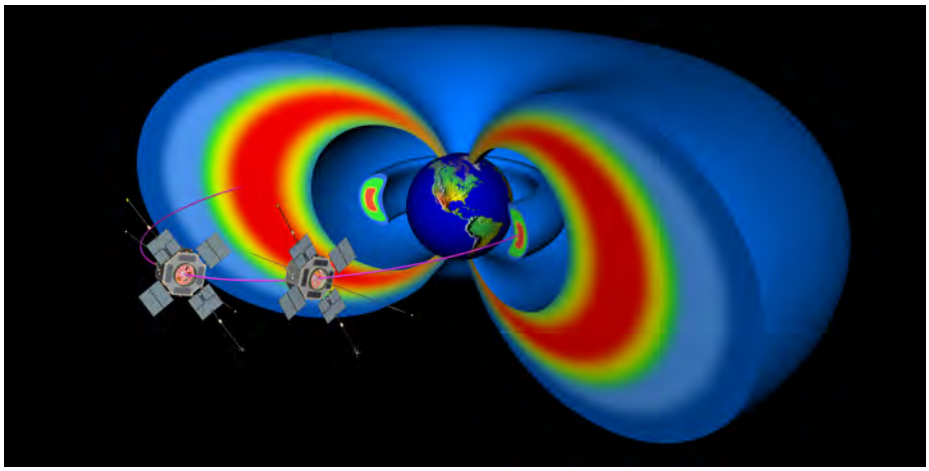

Figure 1.1: Artist's rendering showing NASA's Van Allen Probes orbit through Earth's radiation belts. Image credit: JHU/APL, NASA.

eventually causing irreversible damage which prevents the device from operating as designed. SEE result from high energy particle temporarily changing the state of a semiconductor gate. In a digital system, this would change a binary 1 to a 0, or vice versus. SEE can generally be mitigated by power-cycling the device and do not cause permanent damage. However, if not detected and mitigated this temporary change could cause a satellite computer to execute commands incorrectly or airplane avionics to give inaccurate readings.

Figure 1.2 highlights many of the other effects of the radiation belts and solar storms on key infrastructure. Variation in the Earth's magnetic fields due to solar storms can induce currents in power transmission lines as well as submarine cables and pipelines. In March 1986, a geomagnetic storm caused widespread blackouts and damage in the Hydro Quebec power system [4]. These storms can also disrupt GPS and communication signals and damage space-based assets. Radiation exposure can also have adverse health effects on humans aboard aircraft, the International Space Station (ISS), and other crewed space exploration missions [5].

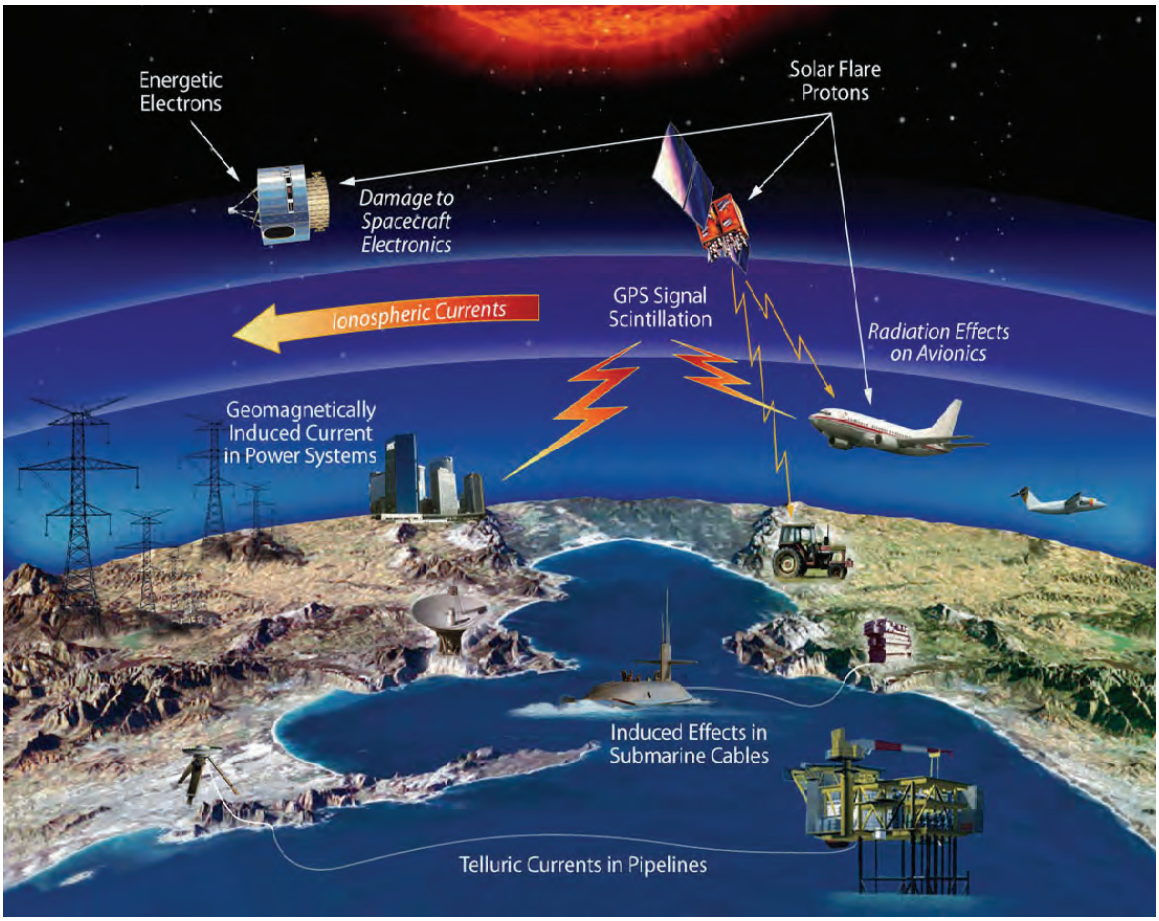

Figure 1.2: Examples of effects of space weather on critical infrastructure [6].

IT-SPINS Mission

The topside transition region (TTR) is a global altitude boundary where the concentration of O^+ decreases to the point of being equal to the concentration of H^+. The height of the TTR is a key metric in characterizing the upper ionosphere [7]. Current studies of the TTR rely on data from ground-based incoherent scatter radar [8] or satellite in situ observations [9]. However, both of these methods are limited to localized observations.

Equatorial plasma bubbles (EPB) are large-scale (20-100 km) plasma instabil-

ities which form in the lower ionosphere and rise upwards through the ionosphere like bubbles. EPBs have been observed from space as regions devoid of 135.6-nm nightglow and are disruptive to RF signals (GPS and communications) which pass through them [10]. Polar patches are similar large-scale ionospheric instabilities which also cause scintillation of satellite communications but form over the polar regions of the Earth [11]. Polar patches and EPBs are also generally studied with ground-based incoherent scatter radar or from satellite observations [10].

The Ionospheric-Thermospheric Scanning Photometer for Ion-Neutral Studies (IT-SPINS) mission is a National Science Foundation (NSF) project that will study the TTR, EPB, and polar patches from a 3U CubeSat platform. The specific mission objectives are outlined in Table 1.1, taken from the the project requirements verification matrix (RVM). By globally studying the TTR, EPBs, and polar patches IT-SPINS will provide valuable input to ionospheric models, furthering our understanding and enabling better real-time predictions of the impact of space weather storms.

Table 1.2 lists the IT-SPINS mission requirements from the project RVM, which are derived from the mission objectives and NSF requirements. From these top-level objectives and requirements, intermediate-level requirements were derived and the spacecraft design shown in Figure 1.3 was developed. The primary payload, the CubeSat Tiny Ionospheric Photometer (CTIP), was developed by SRI International and previously flown on the USAF Space Environment NanoSatellite Experiment (SENSE) mission in a nadir viewing orientation [12]. CTIP is a high sensitivity UV photometer, tuned to measure 135.6-nm emission ($\epsilon_{135.6}$) along a 3.8° field of view (FOV). The IT-SPINS spacecraft will rotate about orbit normal at 2 RPM, sweeping the CTIP FOV through the orbit plane as shown in Figure 1.4 while it makes half-second integration measurements of $\epsilon_{135.6}$ resulting in 60 measurements per rotation.

Table 1.1: IT-SPINS Mission Objectives.

Requirement Number	Baseline Requirement	Source
MO-1	The IT-SPINS mission will study the global variability and underlying physics of the top side transition region of the ionosphere where O^+ transitions to H^+ and He^+.	Proposal
MO-2	The IT-SPINS mission will study the global variability of O^+ altitude profiles throughout the F-region and topside ionosphere.	Proposal
MO-3	The IT-SPINS mission will image the mesoscale structuring and evolution of RF-disruptive structures such as equatorial plasma bubbles and polar cap patches.	Proposal

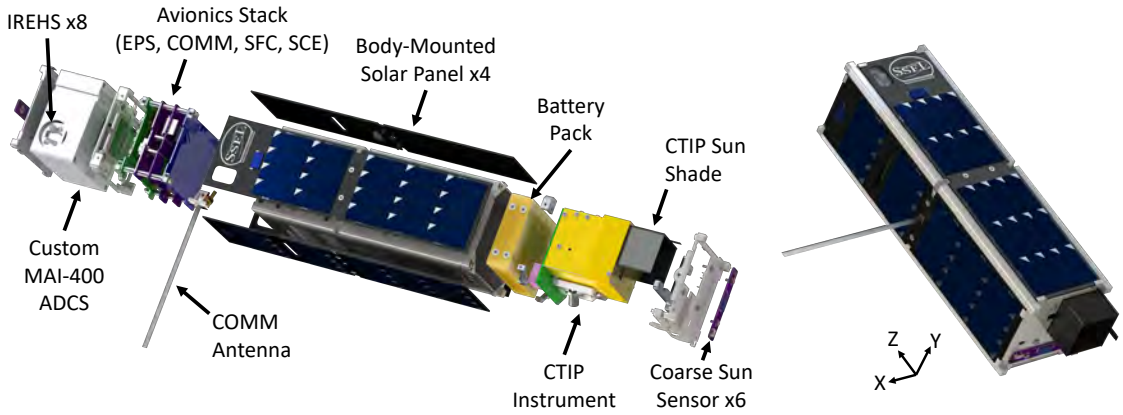

Figure 1.3: 3D CAD renders of exploded view of IT-SPINS CubeSat major subsystems (left) and fully integrated spacecraft with coordinate definitions (right).

Table 1.2: IT-SPINS Mission Requirements.

Requirement Number	Baseline Requirement	Source
M-1	IT-SPINS shall collect line-of-sight observations of 135.6nm nightglow produced from the recombination of O^+ and electrons in the nocturnal ionosphere.	MO-1 MO-2 MO-3
M-2	IT-SPINS shall utilize tomographic inversion to reconstruct two-dimensional images (altitude vs. in-track look direction) from the 135.6nm emissions along multiple intersecting ray paths along the spacecraft orbit with a vertical resolution of at least 50km and a horizontal resolution of at least 200km.	MO-1 MO-2 MO-3
M-3	IT-SPINS shall have an on-orbit operational lifetime of at least six months following commissioning of the spacecraft and science payload.	MO-1 MO-2 MO-3
M-4	IT-SPINS shall have an orbital inclination of greater than 45 degrees with an appropriate nodal precession rate to provide routine eclipses.	MO-1 MO-2 MO-3
M-5	IT-SPINS shall conform to the current CubeSat Design Specification.	NSF
M-6	IT-SPINS shall conform to the NASA CLSI mission-specific launch canister Interface Control Document (ICD).	NSF

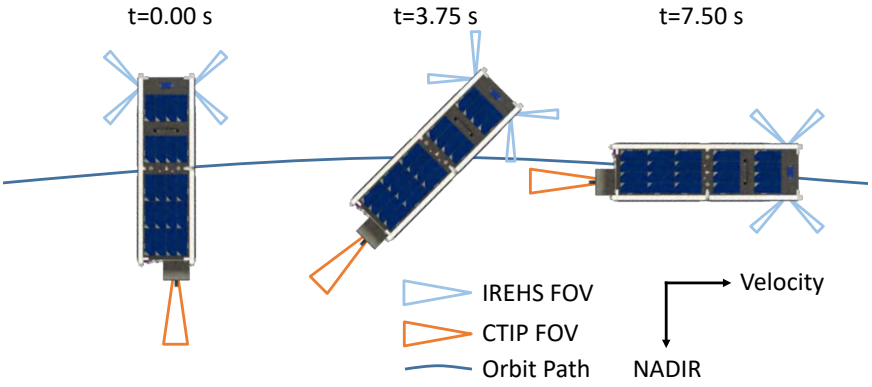

Figure 1.4: Diagram of IT-SPINS rotating about orbit normal with CTIP and IREHS field of views shown.

IT-SPINS will provide a unique dataset compared to previous projects. Incoherent scatter radar studies of the ionosphere are limited to localized observations. Previous satellite-based studies of the TTR, EPBs, and polar patches used nadir-viewing UV instruments [12], some with cross-track scans [10], which only take integrating vertical measurements. The individual CTIP line-of-sight measurements will be post processed on the ground with a tomographic inversion algorithm to produce global two-dimensional altitude versus along-track distance images of $\epsilon_{135.6}$ for the nightside ionosphere.

In order to achieve the desired resolution from requirement M-2, simulations determined the spacecraft's Attitude Determination and Control Systems (ADCS) would need to achieve a spin rate of 12°/s ±1.2°/s, about an axis within ±1.5°of orbit normal. The ADCS must also determine the angular orientation of the spacecraft to within 0.3°.

Early trade studies on how to achieve the required spin and attitude knowledge quickly ruled out ADCS solutions based on start field trackers, as the spin-rate would smear star fields over the required integration time. Instead, Maryland Aerospace Inc.

(MAI) proposed an infrared Earth horizon sensor (IREHS) based variation of their commercial off the shelf (COTS) MAI-400 three-axis ADCS. Four pairs of IREHS grouped into two instrument clusters are used to derive the precise attitude of IT-SPINS relative to the Earth limbs.

The spacecraft avionics and integration are the responsibility of Montana State University's (MSU) Space Science and Engineering Laboratory (SSEL). The avionics are based on that of the FIREBIRD CubeSats and include a revised Phoenix Electrical Power System (EPS), AstroDev Li-2 Radio, Space Flight Computer (SFC) Command and Data Handling (CDH) system, and a Solar Cell Experiment (SCE).

Figure 1.5 shows a simulated reconstruction of $\epsilon_{135.6}$ based on six rotations worth of data from the IT-SPINS spacecraft. The top panels shows the simulated $\epsilon_{135.6}$ of a reference ionosphere based on the Thermosphere-Ionosphere-Mesosphere-Electrodynamics General Circulation Model (TIME-GCM) [13]. The discrete line of sight measurements from IT-SPINS are shown in the middle panel. The reconstructed ionosphere based on the simulated measurements is shown in the bottom panel. The John Hopkins University Applied Physics Laboratory (JHUAPL) is responsible for the tomographic inversion processing that will produce the 2D reconstruction images of the ionosphere.

Problem Statement

The IT-SPINS mission needs a CDH subsystem to coordinate data transfer and storage between all other subsystems. Minimization of mission risk and minimization of manpower would dictate using a duplicate of a previously flown CDH, such as that of FIREBIRD. However, there were a number issues with both the hardware and software of that CDH, which are described in Chapter 2 below. Additionally, the IT-SPINS mission has a very different set of requirements from FIREBIRD. Therefore

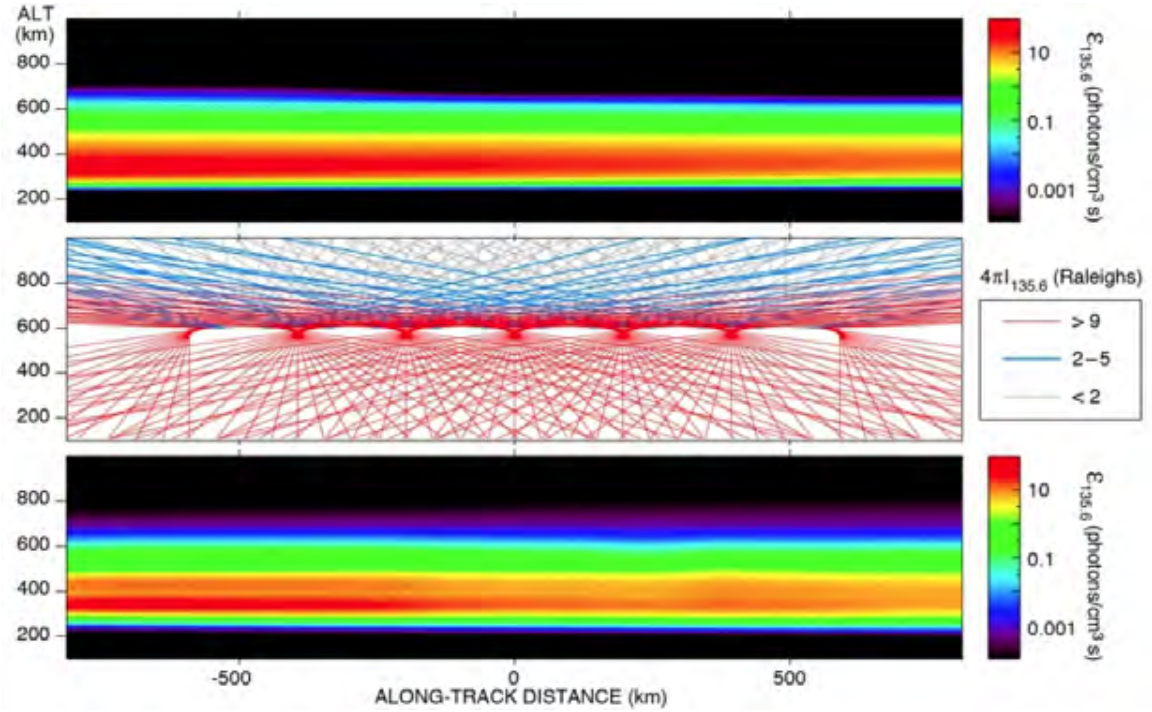

Figure 1.5: Simulated IT-SPINS reconstruction of $\epsilon_{135.6}$ over six rotations.

a new CDH subsystem has been designed to fit the needs of the IT-SPINS mission, while leveraging heritage from successful SSEL projects.

BACKGROUND

The SSEL at MSU has designed, built, and delivered over half a dozen CubeSats for launch to date, and collaborated on many more small satellite projects. The successes and failures of these projects, both on-orbit and during development, have yielded many robust designs, rules of thumb, and other lessons learned. In this chapter, three past SSEL projects are presented which have heritage designs and lessons learned that will heavily influence the design of the IT-SPINS hardware and software.

FIREBIRD Mission

The Focused Investigation of Relativistic Electron Burst Intensity, Range, and Dynamics (FIREBIRD) missions are a collaborative effort between University of New Hampshire, Montana State University, the Aerospace Corporation, and Los Alamos National Laboratory to study relativistic electron microbursts. These microbursts are very brief but intense increases in electron precipitation from the earth's radiation belts into the upper atmosphere. FIREBIRD's mission objectives are to quantify the spatial scale and energy dependence of individual microbursts, as well as to quantify the total electron loss from the radiation belts due to microbursts globally. This NSF-funded project has launched two pairs of 1.5U CubeSats.

The first pair of identical CubeSats, known collectively as FIREBIRD-I, were launched on December 6, 2013 into a 121° inclination LEO orbit of 467 by 883 km [2]. Unfortunately, due to an over-complicated and under-characterized electrical power system (EPS) both FIREBIRD-I CubeSats had a very short operational life. Additionally, a software bug prevented one of the pair from properly booting and beaconing until a command was able to clear the bug several months into the mission.

Due to the low inclination orbit and limited operational life, neither FIREBIRD-I CubeSat observed a microburst.

A second pair, known collectively as FIREBIRD-II, were launched on January 31, 2015 into a 99° inclination LEO orbit of 440 by 670 km [2]. This pair featured a re-designed and simplified EPS, resolving the lifetime issue. The FIREBIRD-II CubeSats have exceeded 2 years in space and are still fully operational. Both have observed hundreds of microbursts, resulting in scientific publication [14], with many more in the works.

Hardware

Each FIREBIRD CubeSat features identical hardware and software, with some revisions between the two flights. Figure 2.1 shows an exploded view of a FIREBIRD CubeSat. The FIRE instrument, developed at University of New Hampshire, features two solid-state silicon detectors, a low-noise pulse processor application specific integrated circuit (ASIC), and a field programmable gate array (FPGA) to discretize and package the electron measurements upto 50 times per second.

The Bus in Support of Radiation Detector (BIRD) package provides power, data storage, and communications for the spacecraft. Developed by SSEL, BIRD includes the Command and Data Handling (CDH), Electrical Power System (EPS), radio Communications (COMM), and Multi-Function Interface Board (MFIB) subsystems.

The CDH is responsible for handling all commands and telemetry throughout the satellite, requiring it to interface to each major subsystem (either directly or through communications with another subsystem). The CDH on FIREBIRD is a Commercial Off-The-Shelf (COTS) product purchased from Pumpkin Inc. It includes a PIC24F microcontroller, 64MB NOR flash memory, 2GB SD Card, Real Time Clock Calendar (RTCC) chip with battery backup, and Electrical Ground Support

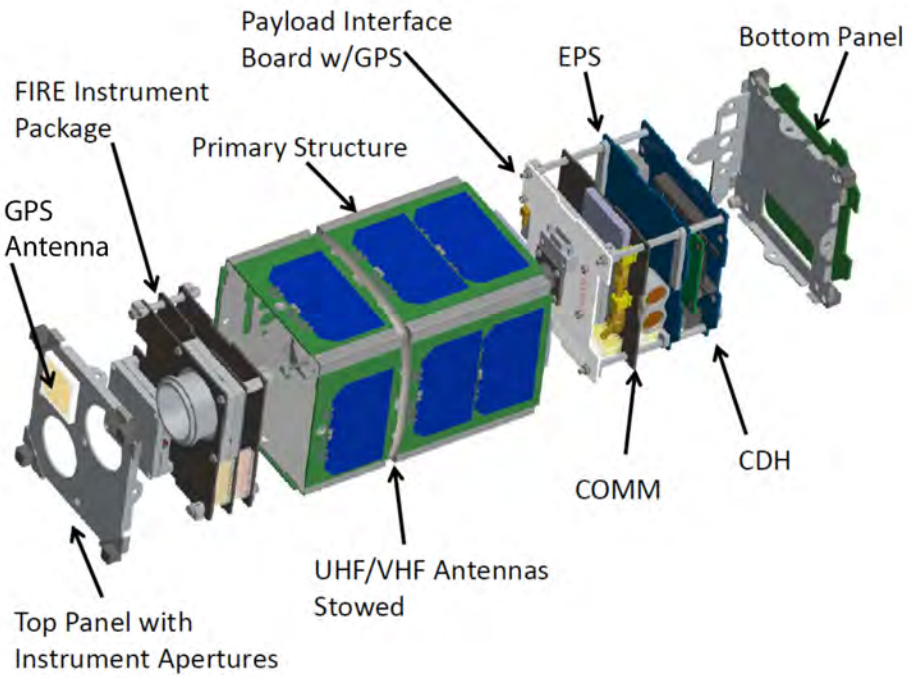

Figure 2.1: Exploded view of FIREBIRD CubeSat major subsystems. [2]

Equipment connections.

The EPS is responsible for managing power gathered by the body-mounted solar panels to charge the batteries and run the system. As mentioned above, FIREBIRD-I featured an over-complicated EPS which was purchased from a vendor hoping to develop it as a COTS product. It included Maximum Power Point Tracking (MPPT) for the solar panels, an excess power shunt, a configurable Watch Dog Timer (WDT) circuit, and a lithium-ion battery pack all managed by an FPGA. As a COTS-targeted design, it was designed to cover the requirements of many different types of missions, each with different power requirements. In order to meet the requirements of higher power missions, it lacked the resolution and efficiency required by FIREBIRD, which has an orbit-average power consumption of only 0.3 Watts. The FPGA contained many configuration registers for every aspect of the system; however, many of them

were under-documented or the documentation was simply wrong. As a result, the batteries were overcharged and the solar panels were under-utilized.

For the FIREBIRD-II flight, the EPS was completely redesigned from the ground up by SSEL. This in-house design is known as Phoenix. It relied on direct energy transfer, connecting the batteries directly (through a protection diode) to the matched solar panels, eliminating the complexity of a MPPT system. Phoenix also included a simplified WDT circuit based on a simple 4000-series logic counter driven by an RC oscillator. This WDT circuit would count up to 12 hours before briefly removing power from the entire spacecraft, forcing a hard reset. This WDT reset mitigates the effects of Single Event Upsets (SEU) due to radiation and other latchups due to potential software bugs. The Phoenix EPS includes a PIC24 microcontroller for telemetry aggregation.

COMM is responsible for receiving and transmitting radio communications between the satellite and a ground station. For FIREBIRD, the COMM subsystem was a COTS He-100 from AstroDev Inc. and two tape measure antennas mounted to the body of the CubeSat.

The MFIB is responsible for interfacing to the FIRE payload and storing it's data. It includes another PIC24 microcontroller, a 2GB NAND flash memory, and a GPS module for timing. FIRE data is stored here, and the CDH requests the data to be downlinked one packet at a time. FIREBIRD relies on a permanent magnet on the MFIB for passive attitude control and has no direct method of attitude determination. The magnet acts as a compass needle, roughly aligning the satellite to the earth's magnetic field.

The FIREBIRD IMM Solar Cell Experiment (FISCE) is an additional payload added on the FIREBIRD-II mission. This experiment records IV characteristics of a single string of solar cells assembled with a proprietary THINS technique developed

by Vanguard Space Technologies, Inc. FISCE includes another PIC24 microcontroller for control and telemetry gathering. The entire FISCE circuitry is included on the back side of a solar panel.

Software

As shown in Figure 2.2, the FIREBIRD CDH uses the μC/OS-II Real-Time Operating System (RTOS) as its kernel. This kernel as well as the other firmware and application layers are written primarily in the C language, with low-level OS functions implemented in assembly language. The μC/OS-II allows the flight software to have multiple tasks, more commonly known as threads, running independently of each other. It includes semaphores, event flags, mutual-exclusion semaphores that eliminate unbounded priority inversions, message mailboxes, queues, task time and timer management, and fixed sized memory block management.

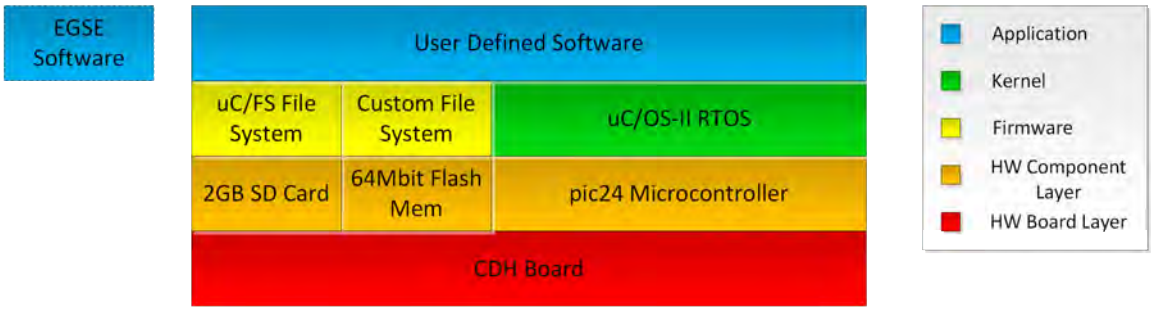

Figure 2.2: Software stack for the FIREBIRD CDH fight software.

Within the "User Defined Software" of the flight software is a further subdivision known as modules. A Module is a collection of functions that provides a specific functionality to the flight software and provides the necessary interface for outgoing data on hardware peripheral. In the FIREBIRD Flight software, a module can never be an OS task, but instead it provides functions called by OS tasks. Modules can be disabled or enabled at compile time as needed. Operations between modules are kept

separate, and the utility provided by the module is highly independent of the flight software and the OS in which the operations need to occur.

The FIREBIRD flight software contains the following eleven modules: Command Manager (CMDMGR), Command Sequence (CMDSEQ), COMM, EPS, FISCE, GPS, MFIB, SpaceCraft DataBase (SCDB), System, Telemetry Manager (TLMMGR), and Telemetry Monitor (TLMMON). The CMDMGR module acts as a router passing commands received from other modules (possibly from external hardware such as the EGSE or COMM) to the appropriate destination module. The CMDSEQ module retrieves and executes sequences of commands stored on the CDH's NOR flash. The COMM, EPS, FISCE, GPS, and MFIB modules all handle interfacing to their respective subsystems throughout the spacecraft. The SCDB module acts as a large current value table, maintaining telemetry values from all modules in the flight software. The TLMMGR module handles output of telemetry from the SCDB to external hardware at a configurable cadence or on demand. Finally, the TLMMON module can be configured to monitor up to 7 telemetry points in the SCDB simultaneously and trigger execution of a command sequence if the the telemetry point falls out of limits.

Additionally, the "User Defined Software" includes hardware peripheral drivers, which interface to the various built-in hardware peripherals (UART, I2C, SPI, etc.) on the PIC and directly connected to the PIC.

In an early unused version of the flight software, each module had it's own OS task. This allowed each module to run on it's own and rely on the OS to pass information from module to module. However, each task requires it's own software stack and the limited RAM on the PIC microcontroller did not accommodate enough tasks once the SD Card filesystem was added to the flight software. As a result only five OS tasks were implemented: `astrodev_rx`, `novatel_rx`, `power_rx`,

`sys_ctrl`, and `wdt`. Each of the `rx` tasks monitors one of the four UART drivers receive buffer and processes incoming bytes. The `sys_ctrl` task is responsible for funneling all incoming packets to the CMDMGR, monitoring the rx tasks' buffers, and uses an OS timer to call worker functions within the various modules which run continuously in the background.

The CDH PIC microcontroller is the only one to run μC/OS-II. All other PIC microcontrollers on the spacecraft (EPS, MFIB, FISCE) have much simpler roles and therefore use simple Interrupt Service Routine (ISR) based code. However, many of the hardware drivers developed for the CDH microcontroller are duplicated for the other subsystem microcontrollers.

Lessons Learned

Throughout the design, build, and flight of all four FIREBIRD CubeSats there have been a number of key lessons learned, both positive and negative. On the postive side, we have now flown twelve discrete PIC24F microcontrollers and seen no signs of permanent faults from any of them. Eight of these have operated on FIREBIRD-II for over two years in a polar orbit. The 2GB NAND flash ICs used on the MFIB for FIRE data storage have also performed without significant fault and are much simpler to interface to and manage than the SD Card on the CDH. Additionally, the Phoenix EPS WDT circuit has saved the system from several temporary latchups and does not have a significant impact on operations.

One of most frustrating challenges facing the FIREBIRD science team is determining what time a measurement was actually taken, both relative to two satellites in a pair, and relative to official UTC time on the ground. The intent was to use the on-board GPS receiver to synchronize the CDH's RTCC to GPS time. This time could then be used to timestamp the FIRE data as it is saved to the MFIB's

NAND flash. However, due to poor implementation and lack of timing calibration on the ground, this method produces timing errors on the order of several seconds. It is not known exactly when or if turning the GPS on for an orbit results in synchronizing the RTCC. The RTCC's oscillator drift was not characterized on the ground across a range of temperatures. The MFIB timestamps FIRE data based on a software RTCC running on the MFIB that is only synchronized once at the start of a data run to the CDH's RTCC, resulting in an additional unknown drift.

Another challenge faced by the FIREBIRD science team is an issue with data downlink, which reduces the overall amount of data downlinked per pass. During a pass, operators request data from a specific location in NAND memory. Ideally, the satellite responds with a continuous stream of packets based on the request. However, it is quite common for the satellite to respond with one packet at a time, with several seconds of dead time between packets. This bug is due to a handshaking issue between CDH and COMM, where COMM does not acknowledge the packet given to it to downlink. This is a software bug in the AstroDev He-100 that has not been resolved; however, there are several workarounds that could be written into future flight software.

While fairly robust, because the COTS CDH is designed as a one-size-fits-all part, it lacks the volume and power efficiency of a mission specific solution. The USB and barrel jack EGSE interface takes up a significant external surface area, but only provides six electrical connections into the satellite. The CDH is a two-board construction with a bottom motherboard and replaceable processor module. The motherboard supports different processor modules with different operating voltages, which requires it to level shift all input/output (IO) lines. It also re-regulates the 5V power provided to it down to 3.3V, resulting in power lost to the regulator efficiency. Additionally, the CDH lacks interfaces and functionality that had to be implemented

on the MFIB board.

The SD Card on the CDH has proven to be less than useful. The initial FIREBIRD design did not include a NAND flash or processor on the MFIB. However, early software development showed that the CDH processor could not handle writing data to SD Card fast enough to support FIRE's data rates. As a result, the SD Card was left only for storing CDH telemetry from the SCDB and GPS data. However, implementing a proper SD Card filesystem with low enough overhead for a microcontroller with 96kB of RAM proved challenging. While the FIREBIRD-II CubeSats are capable of storing some data to their SD Cards, it is frequently corrupted and inefficient to downlink compared to downlinking from the NAND flash. SD Cards slots are also not mechanically designed to withstand the strenuous launch environment. Other CubeSats, such as RAX-2 have also experienced issues with SD Cards [15].

As mentioned above, the FIREBIRD flight software only implemented a few tasks within the OS. This low task count reduces the modularity of each module, since the OS's inter-task communication (mailboxes, semaphores, etc.) could not be used. Therefore, more direct inter-module function calls are required. The main reason for the low task count was to accommodate the memory usage of the SD Card file system.

During initial development of the flight software, some tools were written to auto-generate code for each module, based on the commands and telemetry required by the module. This worked well initially; however, as the modules were further customized to meet the evolving mission requirements, these tools fell out of use and the modules morphed away from the standard templates generated by the tools. A similar story is true for the hardware drivers written, which were initially written to conform to a format and interface standard. As the drive software evolved, each

devolved from the standard. The result was a code base not conforming any standard, with each module and driver different from each other.

The lack of an attitude determination system has also caused frustration with the science team. While the mission requirements do not require attitude knowledge, knowing the rough attitude of each satellite would add significant value to the science measurements.

In summary, while FIREBIRD has been extremely successful, there are several inefficiencies and bugs in the design. Many of the hardware inefficiencies result from using mission-agnostic COTS hardware. Some of the software issues discussed could be resolved with minor updates; however, given the state of the code, a full refactor and standardization would drastically increase the modularity and maintainability of the code for future missions.

EPISEM

The Energetic Particle Integrating Space Environment Monitor instrument (EPISEM) is a single board scientific payload developed by SSEL to fly on the NASA Ames Edison Demonstration of Smallsat Networks (EDSN) 1.5U CubeSat mission. The payload consists of a Geiger-Müller tube to detect ionizing radiation and the required digital and analog support circuitry, as shown in Figure 2.3. The EDSN mission was designed to test a star-type networking concept where multiple small spacecraft would share their stored data over a low data rate cross link. A ground station could then communicate with only one of the spacecraft to command a high data-rate S-Band downlink. This platform of eight distributed 1.5U CubeSat were ideal for the EPISEM payload to study the small-scale spatio-temporal variability of of the Van Allen Radiation Belts.

SSEL delivered 14 flight-ready EPISEM payloads to NASA Ames in the spring

of 2013 for integration and testing with EDSN's eight flight units and two flight spares. The eight EDSN flight units were lost due to failure of the Super Strypi launch vehicle on November 3, 2015. The two flight spares were later deployed off the ISS as the Network and Operation Demonstration Satellite (NODES) mission on May 16, 2016 [3].

Figure 2.3: Image of the EPISEM circuit board displaying the location of the primary circuits and components. [16]

The EPISEM project provided a unique opportunity for SSEL to branch into large-scale production and test of a single design. Projects like FIREBIRD require only a few copies of each board to be built and tested. EPISEM required the development of an extensive Manufacturing Planning Sheet (MPS) document to ensure each of the 33 steps to produce a single board were followed exactly on each copy.

In total, 30 EPISEM boards were built and tested. This high number was due to a mistake in the Computer Aided Design (CAD) phase of the project. The dimensions of the mounting holes were improperly transcribed into CAD and the 14 boards were manufactured and had begun testing before the error was caught. The solution was to build 15 more flight ready boards. This error could have been caught earlier if a full 3D model of EPISEM had been output directly from the Electrical CAD (ECAD) program to a Mechanical CAD (MCAD) program for a virtual fit check. Instead the MCAD design was drawn by hand based on the correct mounting hole locations, so the virtual fit check yielded no errors.

SSEL ICE

The SSEL Integrated CDH and EPS (ICE) was a senior capstone project at MSU's Electrical and Computer Engineering (ECE) department sponsored by SSEL. The goal was to develop a prototype level single-board integrated CDH, EPS, and COMM based on that of FIREBIRD. In this one year project, the team successfully designed, built, and tested the board shown in Figure 2.4.

ICE includes the same PIC24 microcontroller and NAND flash used on FIREBIRD, as well as the 12-hour WDT and an analog mux used for telemetry collection on the Phoenix EPS. ICE also includes two DC-DC voltage regulators used on Phoenix, a Remove Before Flight (RBF) switch, and accommodation for inhibit foot switches. The PIC select DEMUX is a new circuit designed for ICE which allows multiple PIC24 microcontrollers to be programmed through a single Microchip ICSP bus entering the satellite through the EGSE connector. Provisions were also made for the AstroDev Li-1 transceiver. The stacking bus header allowed for solar panel and battery input as well as a range of digital IO to connect to a mission specific payload.

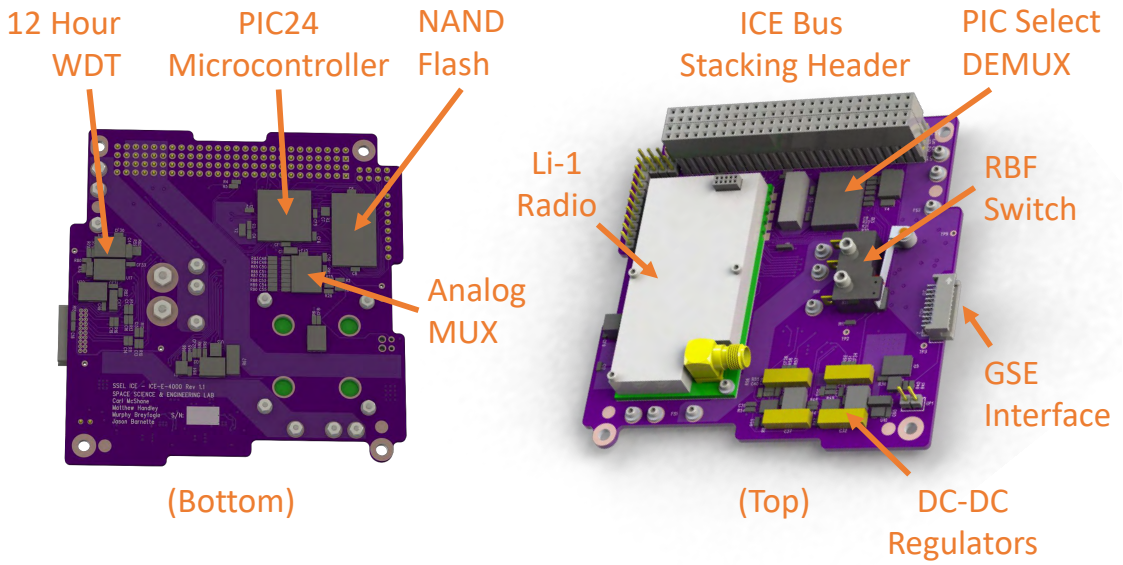

Figure 2.4: Annotated CAD renders of SSEL ICE showing all major sub-circuits on the bottom (left) and top (right) of the board.

ICE proved that the FIREBIRD bus could have been greatly simplified into a single ICE board with a mission specific battery, solar panel, and payload interface board. It also verified the PIC select programming logic, which could have simplified the FIREBIRD EGSE interfaces. However, software testing on ICE showed that a single PIC processor could not easily handle running the full μC/OS-II while buffering telemetry and simulated science data into NAND flash.

DESIGN

As mentioned above in the Problem Statement, the IT-SPINS mission needs a CDH subsystem to handle commands and telemetry and control all other subsystems. Previously flown CDH subsystems such as that of FIREBIRD, while successful for that mission, still have a number of issues and do not meet the requirements of the IT-SPINS mission. Therefore, the decision was made to design a new in-house CDH subsystem for the IT-SPINS mission, known as the Space Flight Computer (SFC). The SFC was designed to take on the roles of both the CDH and MFIB from FIREBIRD's avionics stack, and re-use as much of those designs as reasonable, to reduce mission risk.

SFC Subsystem Requirements

The complete SFC design requirements are listed in Appendix A. These requirements were derived from the top level requirements listed above in Tables 1.1 and 1.2, as well as intermediate level requirements developed from the spacecraft design and packaging.

Requirement SFC-1 is derived from the design decision to use CTIP as the primary scientific payload. The 5V CTIP RS-422 hardware interface was already defined from the SENSE mission which previously flew CTIP. The conversion to UART was based on the availability of hardware peripherals on the heritage PIC24 microcontroller which will be used on the SFC.

Requirement SFC-2 is derived from the design decision to use a custom variation of the MAI-400 ADCS. While the MAI-400 does offer a range of interfaces, the 3.3V UART interface was choosen as the most compatible with the PIC24 microcontroller.

The decision to use an AstroDev radio was based on heritage from FIREBIRD

and lack of a comparable competitor within the same price bracket, despite the handshaking issues experienced on FIREBIRD. However, changes in IARU frequency coordination rules, meant the VHF uplink/UHF downlink AstroDev He-100 used on FIREBIRD could not be used on IT-SPINS. Therefore, the decision was made to use a UHF uplink/UHF downlink AstorDev Li-1, which has a similar 3.3V UART interface to the He-100, which is captured in requirement SFC-3.

Requirement SFC-4 is derived from the design decision to use an updated Pheonix EPS, based on that of FIREBIRD. While the FIREBIRD Pheonix EPS featured a 3.3V UART interface, the PIC24 microcontroller used on the SFC only has 4 dedicated UART hardware peripherals. These 4 UART interfaces are mapped to CTIP (SFC-1), the MAI-400 (SFC-2), the Li-1 (SFC-3), and the GSE (SFC-7). Therefore, the EPS interface was changed to an 3.3V I2C register interface for IT-SPINS. This 3.3V I2C bus will be shared with the SCE to meet requirement SFC-6, since the FISCE payload on FIREBIRD-II also used an I2C interface.

Requirements SFC-5 and SFC-7 are derived from existing designs for the MFIB (3.3V SPI) and GSE (3.3V UART) interfaces on FIREBIRD. However, the 3.3V UART GSE interface will be passed directly out of the satellite's EGSE interface on IT-SPINS, rather than through a UART-USB serial port converter, which was built into the FIREBIRD CDH.

Requirement SFC-8 is derived from the angular attitude knowledge requirement of 0.3°and spin rate requirement of 12°/s ±1.2°/s. Additionally, the problems with the FIREBIRD RTCC timing scheme prompted the design of the new temperature compensated oscillator (TCXO) based mission elapsed time (MET) clock.

Requirement SFC-9 is derived from the need to store CTIP and ADCS data long term, until it can be downlinked over a ground station. Requirements SFC-10 through SFC-16 are based on software features and command and telemetry processing that

worked well on the FIREBIRD missions, and will support the science data, telemetry, commands, and concept of operations for IT-SPINS.

Hardware

Figure 3.1 shows a system level block diagram for the SFC design. The system bus (left) provides an interface the EPS, COMM, and SCE subsystems, while CTIP and the MAI-400 will interface directly to the SFC via their own dedicated busses (lower right).

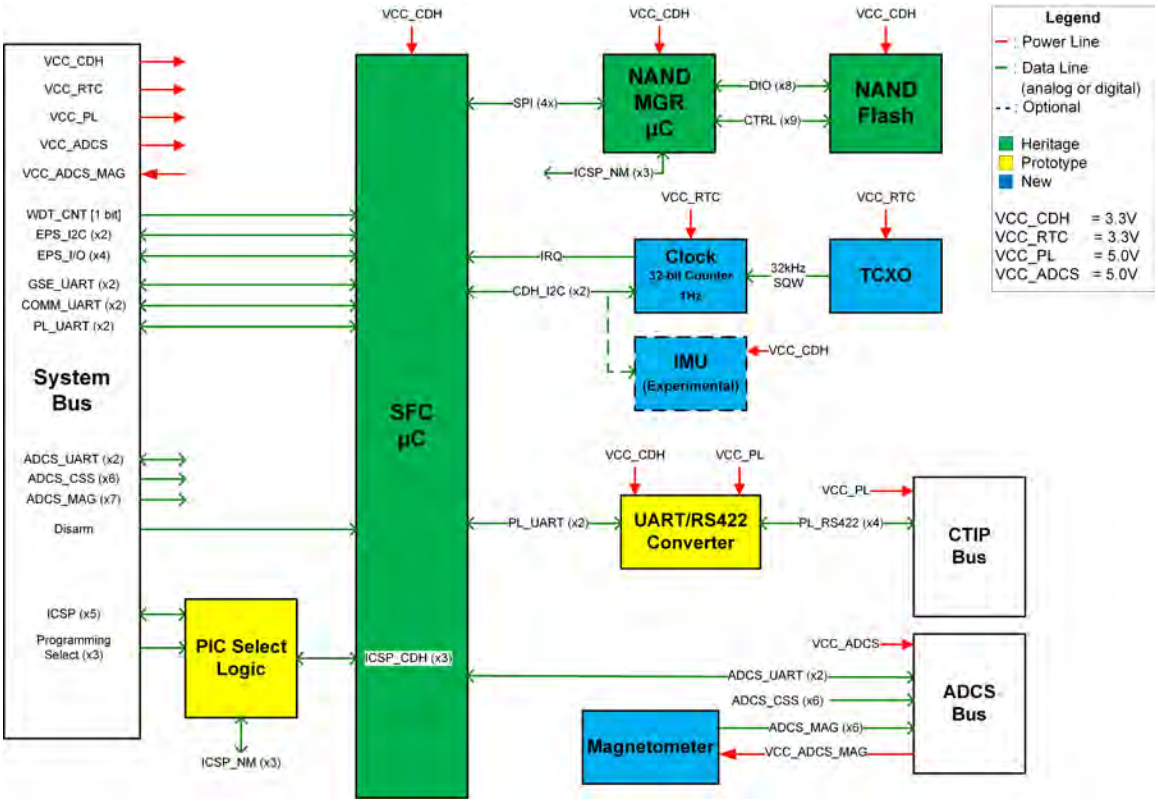

Figure 3.1: Block diagram of SFC hardware.

CTIP Interface

The SFC shall provide CTIP with regulated $5V_{DC} \pm 0.2V_{DC}$, $2A$ maximum. This voltage rail will be provided to the SFC from the EPS through the IT-SPINS System Bus. CTIP controls all Payload internal power distribution and switching. CTIP shall not be damaged from the power service on/off states. CTIP power, along with data and programming signals, is received through a 21-pin micro-D socket receptacle Airborn P/N MS-262-021-435-220S. This is similar to a standard 21-pin Micro-D; however, it is considered "low-profile" and not physically compatible. The CTIP payload communicates via a four-wire full duplex RS-422 protocol at 57.6kbps. The SFC's PL1 connector provides the required interface to CTIP through a standard-profile Micro-D connector from ITT Cannon, P/N: MDM-21SCBR-A174-F222. The pinout of SFC's PL1 and CTIP's J3 connectors was intended to be identical; however, due to a PCB footprint issue it is not. For future revisions of this board, it is suggested that the same Airborn part be used on the SFC side of the CTIP interface with the correct footprint. Lower-cost Harwin M80 series connectors, used on the IT-SPINS Phoenix EPS, would also work well for this interface.

The UART/RS422 Converter block could have been based on one used on the FIREBIRD MFIB to interface to the FIRE payload. The FIREBRID MFIB used Texas Instruments part numbers DS90LV011AH and DS90LT012AH for differential signaling to UART conversion. However, these parts had a poor ESD tolerance (2kV HBM [17]) and had to be replaced on several occasions due to failure. Additionally, the maximum supply voltage of the TI parts was only 4.0V. CTIP's RS422 interface required 5.0V signal levels. However, the SFC's microcontroller can only driver UART lines at its supply voltage of 3.3V. For these reasons, an alternative part(s) had to be found.

After some searching, the Linear Technology LTC2865 RS485/RS422 transceiver

was found to be a good replacement. It has higher ESD tolerance (15kV HBM), combines transmitter and receiver into a single package, and also performs the level shifting from 3.3V UART to 5.0V RS422 [18]. Figure 3.2 shows a block diagram of the LTC2865. $\overline{\text{RE}}$, DE, and $\overline{\text{SLO}}$ are control lines to enable the receiver, transmitter, and "slow" mode, respectively. These control lines are driven by the SFC's main microcontroller. DE (driver enable) may be pulled low when not transmitting data, which saves several milliwatts of power. The $\overline{\text{SLO}}$ pin puts the LT2865 into the slew rate limited 250kbps max data rate mode, still plenty for CTIP's 57.6kbps data rate. This slow mode also saves power on the LT2865, while reducing transmission of high frequency noise.

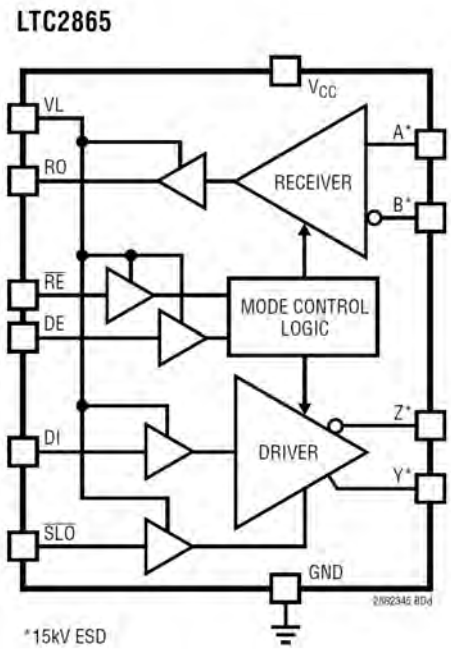

Figure 3.2: LTC2865 transceiver functional block diagram [18].

Because of the low data rate of the RS422 interface, little effort was made to match lengths on the differential pairs; however, they were kept to a minimum. Table 3.1 shows the final trace lengths l and associated propagation time delay T_D , based

on Equation 3.1, for each of the CTIP RS422 traces. Since the propagation delay of the longest trace is several orders of magnitude less than the Unit Interval (UI), calculated to be 17.36 μs using equation 3.2, this length mismatch can be ignored.

Table 3.1: CTIP differential pair lengths and propagation delays.

Net Name	Length (mils)	T_D (ps)
CMD_A	612.07	99.21
CMD_B	514.11	87.71
TLM_A	419.91	68.06
TLM_B	705.48	114.30

$$T_D = \frac{l\sqrt{D_k}}{c} \tag{3.1}$$

$$UI = \frac{1}{DR} \tag{3.2}$$

In Equations 3.1 and 3.2, l is the length of the trace, D_k is the dielectric constant of the PCB substrate equal to 3.66 for this design, c is the speed of light in a vacuum, and DR is the data rate.

ADCS Interface

The SFC shall provide the MAI-400 with regulated $5V_{DC}$, $1.7A$ maximum. This voltage rail will be provided to the SFC from the EPS through the IT-SPINS System Bus. The MAI-400 internally regulates a $3.3V_{DC}$ bus for its external magnetometer, which will be mounted to the SFC. This magnetometer is external to the MAI-400 system, so it can be placed away from the magnetically noisy reaction wheel motors and torque coils.

The MAI-400 uses two 13x2 pin connectors, J1 and J2, with 2mm pitch for communications and power interfaces. SSEL will build an ADCS connector adapter board that translates the MAI-400's two 13x2 headers into a single Micro-D connector interface. The SFC's ADCS connector provides this interface using a Micro-D connector from ITT Cannon, P/N: MDM-21PCBR-A174-F222. This ADCS connector is nearly identical to that of the CTIP interface; however, it is the opposite gender (male/pin) to avoid connecting the CTIP harness to the ADCS interface or vice versus. The pinout of the SFC's ADCS connector is based what was simplest to route on the SFC, as no other constraints were placed on the pinout. This connector pinout and footprint match correctly, unlike that of the CTIP interface.

The MAI-400 also requires input from six external Coarse Sun Sensors (CSS), one on each face of the CubeSat. The CSS were placed on the solar panel substrates and have signal conditioning circuits on the EPS and SCE, which pass the scaled $0V_{DC}$ to $3.3V_{DC}$ analog signal to the SFC through the IT-SPINS System Bus.

System Bus Interface

The system bus provides an interface between the SFC, EPS, COMM, and SCE. The pinout and form-factor of this bus is based on that of the FIREBIRD-II mission which was based on the Pumpkin CubeSat Kit (CSK) header. However, the current revision of the system bus on the SFC is not directly compatible with the CSK standard.

When the SFC was being designed, the final Phoenix EPS design for IT-SPINS was not complete. It was known that the EPS would be made as similar to the FIREBIRD-II Phoenix EPS as possible. One major change that had to be made was the switch from a UART to I2C for EPS commands and telemetry. The EPS will also act as a passthrough for programing and debug interfaces EGSE signals

from the EPS's GSE connector to the SFC via the system bus. The EPS provides 4 regulated power busses to the SFC via the system bus VCC_CDH, VCC_RTC, VCC_PL, and VCC_ADCS. All power busses from the EPS will be power-cycled by the EPS's WDT every 12 hours, with the exception of VCC_RTC and VCC_ADCS. This provides latchup recovery for the SFC without interrupting the clock or ADCS. The SFC may manually power cycle the VCC_RTC or VCC_ADCS via the EPS I2C interface.

Microcontrollers and Flash Memory

The main microcontroller on the SFC is a Microchip PIC24FJ256GB210. This part was selected since it has been used on many previous SSEL designs, including the FIREBIRD CDH, MFIB, Phoenix EPS, FISCE, SSEL ICE, and EPISEM. Because of its heritage within the SSEL, there is a large code-base of software drivers and modules for this part. SSEL has also ported the μC/OS-II to work on this part. This 16-bit, 100-pin microcontroller is run at 16 MHz and includes 96kB of RAM, 256kB of program flash memory, 4 UART hardware peripherals, 3 SPI hardware peripherals, and 3 I2C hardware peripherals.

From what we learned on the FIREBIRD missions, it was decided to use a single NAND flash for storage of both science data and system telemetry. Based on experience from SSEL ICE, it was decided to use another PIC24 microcontroller as a dedicated buffer and memory manager for the NAND flash. Originally, this NAND flash was intended to also store command sequence data, eliminating the need for the NOR flash from the FIREBIRD CDH. However, early software development on the SFC proved the NOR flash necessary, so it was added with a daughterboard on the bottom of the SFC.

The PIC Programming Select Logic was developed on the SSEL ICE project as

a way to program multiple PIC24F microcontrollers on a single ICSP bus, since they do not have good support for JTAG or other similar interfaces. This allows a single programming bus to enter the satellite through the EGSE (Electrical Ground Support Equipment) connector and program multiple PIC24 microcontrollers throughout the satellite. The key to this circuit is the use of a DEMUX to route the reset signal within the ICSP bus to only one of many microcontrollers, based on the binary code present on the DEMUXX's select lines. All microcontrollers on the bus will receive the same programming data and clock lines (GSE_PGED and GSE_PGEC), but will ignore these signals unless they have just been reset.

TCXO Driven Mission Elapsed Time Clock

The FIREBIRD CDH featured a RTCC IC, which was occasionally synced to GPS time. However, due to numerous issues with this architecture, there was a desire to switch to a simpler integer-counter based clock. There was also concern about the temperature stability of the FIREBIRD clock, so a TCXO (Temperature Compensation Oscillator) was desired for the IT-SPINS mission. After a trade study of the available TCXO's and counter IC, the Maxim DS32KHZSN TCXO and Maxim DS1372 32-bit binary counter were selected. This circuit is referred to as the MET (Mission Elapsed Time) Clock.

The DS32KHZ is a 32.768 kHz oscillator which measures its own temperature every 64 seconds and adjusts its output frequency accordingly. It has a frequency stability of ±2ppm from 0°C to +40°C and ±7.5ppm from -40°C to +85°C. It can be supplied either $4.5V_{DC}$ to $5.5V_{DC}$ on its V_{CC} pin or $2.7V_{DC}$ to $5.5V_{DC}$ on its V_{BAT} pin. [19]

The DS1372 is a 32-bit binary counter clock with an integrated 64-bit ID. The 32-bit counter is designed to be driven by a 32.768 kHz oscillator and increment its

32-bit counter once per second (1 Hz). This results in a maximum value of 2^{32} seconds or approximately 136 years. The DS1372 also features a 64-bit ID which is unique to each device manufactured by Maxim. This ID will serve as the serial number of an SFC board once assembled and tested. It can be supplied $2.4V_{DC}$ to $5.5V_{DC}$ on its V_{CC} pin. [20] The DS1372 communicates as a slave to the SFC main microcontroller via the SFC I2C bus at $3.3V_{DC}$.

Since the DS1372 must communicate at $3.3V_{DC}$, both it and DS32KHZ are supplied with regulated $3.3V_{DC}$ from the EPS via the RTC_3V3 bus. RTC_3V3 will not be turned off by the EPS during watchdog resets, so that the clock may continue counting. In the event that the clock becomes latched-up, the SFC may manually power cycle the MET circuit via a command to the EPS.

Experimental IMU

While the IT-SPINS ADCS will have its own IMU (Inertial Measurement Unit) for determining the attitude of the spacecraft, there is a strong desire to develop a familiarity and flight heritage with an IMU, so that it could be used on future missions. After researching the available low-cost integrated 3-axis accelerometer, 3-axis gyroscope, and 3-axis magnetometers available on the market, the InvenSense MPU-9250 was selected. The MPU-9250 is multi-chip module featuring two dies integrated into a single QFN package. The primary die houses the 3-Axis gyroscope, the 3-Axis accelerometer, and a temperature sensor. The other die houses the AK8963 3-Axis magnetometer from Asahi Kasei Microdevices Corporation. Since this subcircuit is not critical to mission success, its full implementation and testing is not required unless time is available.

Software

Science Data as Telemetry

In the FIREBIRD flight software, telemetry and science data were handled differently. FIRE's data bandwidth was too high for the CDH to handle writing to SD Card, so science data storage was moved to the MFIB's NAND flash, while the SD Card was left to store telemetry data. Because of the priority placed on FIRE data, optimizing downlink from the SD Card was not performed. Only FIRE data downlink was optimized. However, the TLMMGR system for routing SCDB telemetry to different hardware outputs (GSE, COMM, or SD Card) at fixed time intervals and on-demand worked well.

For IT-SPINS the science data storage requirements are much lighter. Only an integer photon count, MET timestamp, and ADCS attitude knowledge data needs to be stored twice per second. Therefore, the design decision was made to treat science data as telemetry, taking advantage of the SCDB and TLMMGR while using the MFIB-like NAND Manager as an alternative SD Card output. Data downlink will be based on that of the MFIB, combing the best of both systems.

Code Base Refactor

As discussed above in the Background chapter on FIREBIRD, the flight software code base diverged from the standards originally set out for it. Additionally, the low task count prevented full utilization of the μC/OS-II RTOS's semaphores, event flags, mutual-exclusion semaphores, queues, and time management. In order to resolve these issues, two changes were proposed: (1) switch to Microchip MPLAB Code Configurator (MCC) auto-generated hardware peripheral drivers and (2) redefine a standard that each module will conform to, including an OS task for each module.

Microchip's MCC tool gives the user a graphical interface to configure the functionality of each hardware peripheral built into the PIC, such as the UART, SPI, I2C, and IO peripherals. MCC auto-generates C code with a standard interface for each peripheral configured. Microchip offers an alternative Libraries for Applications (MLA) standard library of drivers. However, an internal trade study showed that MCC was a better fit for this project, as it maintains a standard source code format, while allowing the necessary customization of each driver.

During initial implementation of the NAND Manager module, it was discovered that calls to `OSMutexPend()` and `OSMutexPend()`, which ensure exclusive access to a resource, were silently failing. Because the `sys_ctrl` task handles multiple modules simultaneously, if module A acquires mutex on a resource, when module B requests access to the resource, it would get a failure code indicating the `sys_ctrl` already has exclusive access. This caused problems, since the failure was not handled properly and module B would continue assuming it had acquired the exclusive access to the resource. This triggered a full refactor and reorganization of the flight software into an individual task for each module.

The new standard for each module is outlined below. Each module will be organized into the following 3 files `<ModName>.h`, `<ModName>_task.c`, and `<ModName>_cmds.h`, where `<ModName>` is the name of the module. Modules which interface to external hardware (NANDMGR, MET, EPS, COMM, etc.) will have two additional files `<module>_driver.c` and `<module>_driver.h`. All module files (both `.c` and `.h`) be contained within one directory of the same name. Templates of each of these files can be found in Appendix B.

`<ModName>.h` shall contain only function prototypes and data that should be accessed outside the module, while others will be defined within the `.c` files. Previously the header files had contained many declarations which unnecessarily made

them globally accessible to other modules. These internal declarations will be moved to the top of the appropriate `.c` file in where they will be used. If a declared variable or function needs to be accessible within multiple `.c` files in the same module, it be redefined with the `extern` attribute.

`<ModName>_task.c` shall contain the `<ModName>Task()` function which is the entry point for task, once initialized. This file shall also contain functions that are intended to be called from other modules. To reduce inter-module communication, increasing modularity, `<ModName>_task.c` should generally only contain `<ModName>QPost()` and `<ModName>SetFlag()`. These functions post an OS message pointer to the module's queue and set an OS flag within the module respectively. Functions which process OS Flags or events may also be defined in this file, but should not be declared globally in the module's header file.

`<ModName>_cmds.c` shall contain the `<ModName>ProcessLocalCmd()` function as well as per-command functions. When a message is posted to the module's queue using `<ModName>QPost()` a flag is posted as well. The task loop, pending on flags, will then call `<ModName>ProcessLocalCmd()` until the message queue is empty. `<ModName>ProcessLocalCmd()` will validate the command message and call the corresponding per-command function. The message will always be freed in `<ModName>ProcessLocalCmd()` to prevent memory leaks.

`<ModName>_driver.c` shall abstract calls to the hardware drivers (SPI, UART, etc.) and provide utility functions. For example, read and write register functions for an I2C device or SendPacket and GetData functions for UART devices.

`<ModName>_driver.h` shall declare only functions and data that should be accessible to the module. This file should only be include by its own module.

HARDWARE FABRICATION

This chapter covers the steps required to take the block diagram in Figure 3.1 and build a fully functional Printed Circuit Board (PCB). This includes the CAD steps of schematic capture and layout, as well as fabrication and assembly.

CAD

The first step in CAD schematic capture is drawing each part that will be used in the CAD program and saving it within a library. In the case of the SFC, many parts have heritage from previous designs, so their drawings were simply imported from an existing library. The part drawing includes all of the pins on the device and is often organized based on pin function, rather than pin location on the actual device.

Once all the parts are drawn they are placed on schematic pages and wires or "nets" are drawn to indicate connections between pins on each part. Figure 4.1 shows an example of a the completed sub-circuit schematic for the experimental IMU. The connections made are based on the device's datasheet recommendations. Off-page symbols are used to show that a net connects to something elsewhere in the schematic. For example `SYS_3V3` is the 3.3V supply for most of the devices, so it is cleaner to use off-page symbols than to draw a continuous line to each device that needs it. De-coupling capacitors `C27` through `C28` are connected to each power pin of the device, to filter out noise on the power line before entering the device, as well as prevent noise from the device propagating out onto the power line. Notes can also be made on the schematic such as `Slace Addr:  1101001`, which indicates that the IMU will have the given slave address in binary on the I2C bus.

Following schematic entry, the design moves onto PCB layout, where the actual PCB is drawn. As with the schematic, each part must have a corresponding drawing

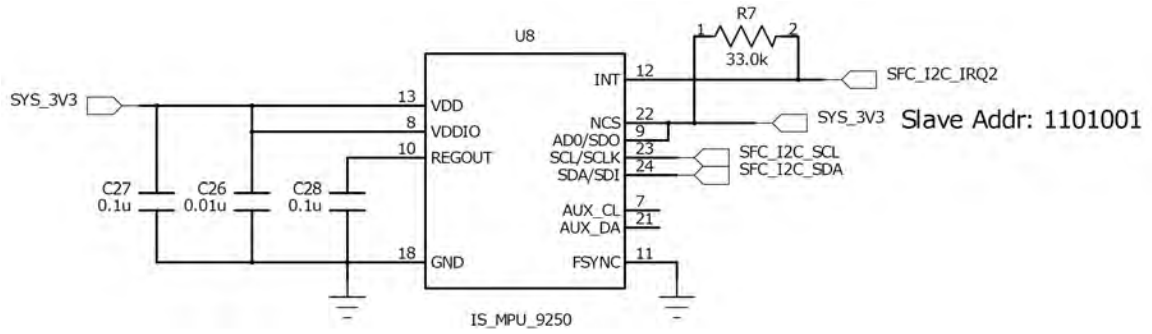

Figure 4.1: Example of schematic entry view, showing experimental IMU circuit.

with all pins and dimensions accurately represented. In layout parts are first organized by sub-circuit and then sub circuits are organized based on the interconnectivity between them. In the case of the SFC, a 4 layer PCB design was chosen with one internal ground plane layer and another internal power plane. Since almost all devices require a connection to both power an ground, they can simply drop down to the internal layer to for these connections, rather than having to route power and ground traces across the while board. All parts were placed on the top surface, with the bottom surface reserved for additional routing.

Figure 4.2 shows an example of a the completed sub-circuit layout for the experimental IMU. Layout software tools are linked to the schematic tool, so they can indicate which connections need to be drawn based on the schematic's connections. The colors used in the layout indicate the layer a copper trace is drawn on. Additionally, power and ground connected pads are shown in yellow and green respectively.

After making all required connections in layout, the Design Rule Check (DRC) tool is used to verify that all connections drawn in the schematic are present in the layout and that no copper traces will be too close together to manufacture. Once all checks pass and the final design has been reviewed, Computer-Aided Manufacturing

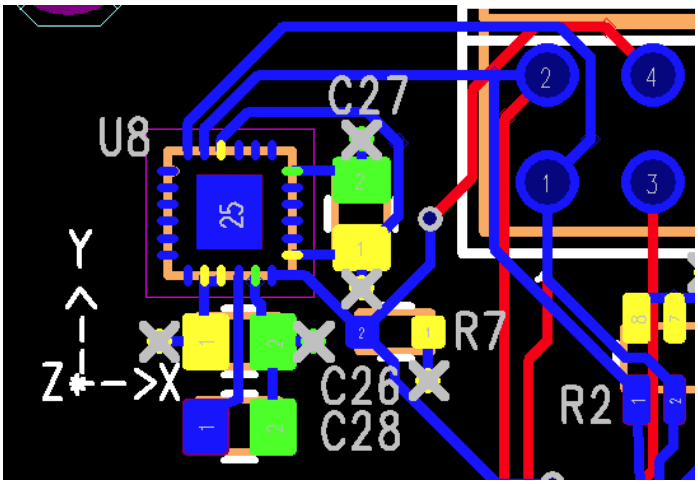

Figure 4.2: Example of layout and routing view, showing experimental IMU circuit.

(CAM) files are output to describe the layout to the board house that will fabricate the board.

Fabrication

The SFC PCBs were ordered from OSHPark.com at a cost of $149.10 for 3 copies shipped, based on their 4-layer cost of $10 per square inch. Several minor issues were found with the completed boards. First, one of the three boards received had a slight misalignment of the topside solder mask. This misalignment was worst in the upper left corner of the board around the IMU, as shown in Figure 4.3.

Another issue visible in Figure 4.3 are small "bubbles" in the solder mask. These bubbles appear near any copper on both the top and bottom side of the PCB. OSHParksupport reasoned that the panel may have dried and cured too quickly, leaving water trapped near the traces and causing some water bubbles to pop up. This is not normal for OSHPark PCBs. This cosmetic issue does not affect the performance of the boards, and required no correct actions.

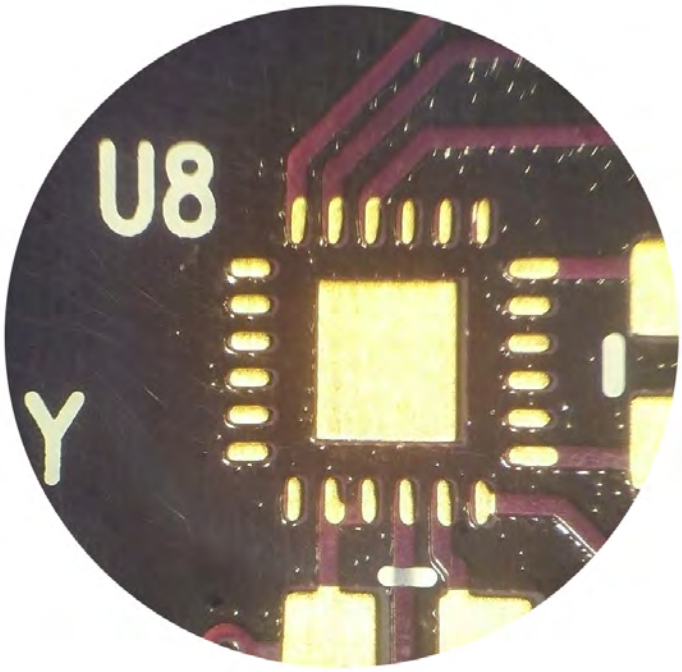

Figure 4.3: Microscope picture of solder mask misalignment and bubbles.

Assembly

Parts were ordered from DigiKey.com; however, some components were already in available in SSEL's stock. Two of the three PCBs were assembled in stages by Randy Larimer, PE. All components were hand-soldered, with the exception of `U8` (the IMU) and `U5` (RS422 transceiver) which were reflowed, since both have thermal ground pads under the package. Initially all but the expensive `PL1-2` connectors were loaded on one PCB. After some basic testing and verification, the PCB was returned to load the remaining parts and fully populate the second board. This provided SSEL with two identical copies of the SFC, one of which is shown in Figure 4.4.

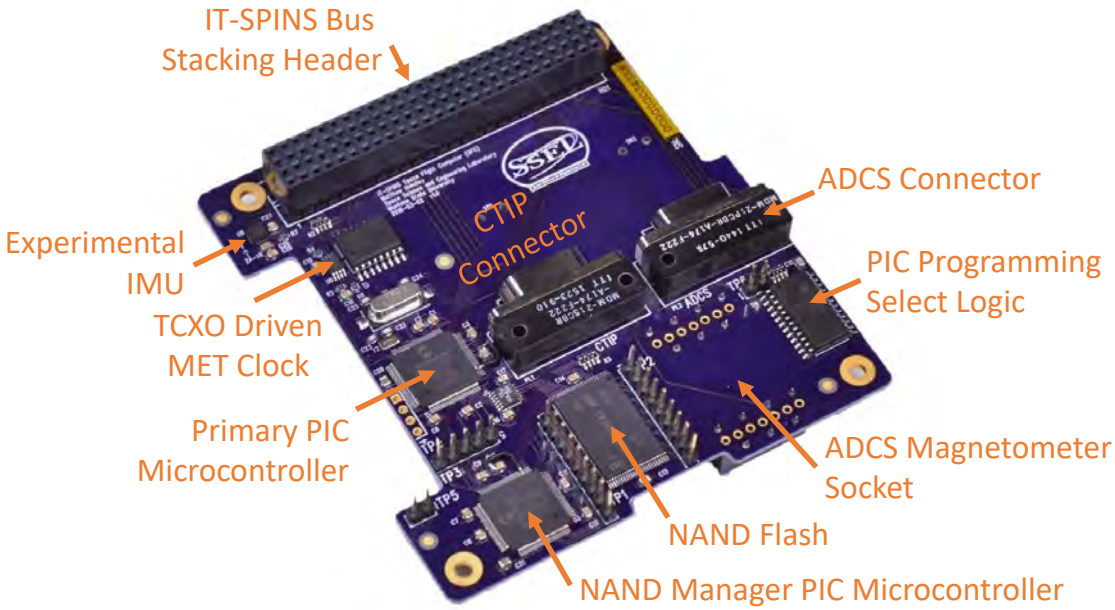

Figure 4.4: Annotated picture of fully assembled SFC showing all major sub-circuits.

SOFTWARE IMPLEMENTATION

The flight software was developed in Microchip's MPLAB v3.60 Integrated Development Environment (IDE) with the XC16 compiler. As mentioned in the Design chapter, Microchip's MCC tool was also used to auto-generate device configuration and hardware peripheral driver source code.

During the code base refactor, a Python tool called `mod-gen` was written to take templates of the new module standard, shown in Appendix B, and generate each module's files. Existing code from the FIREBIRD project was then added function-by-function into the newly generated files and updated to conform to the standard.

Much of the code required by IT-SPINS was already implemented for FIREBIRD. However, it still took significant effort to refactor and reorganize the FIREBIRD software modules into the new standard outlined in the Design chapter. In total, this took approximately 6 weeks for full-time labor to complete the migration of approximately forty-five thousand lines of code. While very time-consuming, this refactor has made the code base much more modular and maintainable for both IT-SPINS and future missions.

Development of new modules for IT-SPINS was fairly straightforward (CTIP, IMU, etc.); however, the MAI-400 software interface was rather challenging. MAI-400 commands were straightforward, as they are required to begin with a two-byte sync character, they are a constant length, and they have a checksum in the final two bytes. The telemetry is divided among a "standard" telemetry packet and two "additional" packets. Each of these three different packets are structured slightly differently, with some containing a "packet type" field, while others use a unique sync character as a type identification. Additionally, one of the "additional" telemetry packets uses a different checksum algorithm from the commands and other telemetry packets. The

solution was to implement the MAI-400 driver with three separate receive buffers, for each of the telemetry types. A state machine is used to determine which packet type is currently being received and push incoming bytes into the appropriate buffer. This approach was used instead of a single receive buffer, as all three packets are usually received in rapid succession and the RTOS would not always service the flag indicating the buffer contained a packet before the next packet needed to be buffered.

Giving each module its own task required each module to have its own CPU stack, which μC/OS-II switches into operation on context switches between tasks. The nature of the PIC24 microcontrollers used required this stack space to be allocated in the lower 32kB of RAM, limiting available stack space. In order to minimize the pre-allocated stack of each task, while avoiding stack overflows, additional telemetry points were added to the System module to monitor stack usage. This was implemented using μC/OS-II's `OSTaskStkChk()` function. Each time a module was implemented its stack would be allocated larger than needed. After running the new module and exercising all of its functionality while monitoring the stack telemetry, a more conservative stack size would be chosen based on the maximum usage under load with some added margin.

The FIREBIRD flight software had been developed using Apache Subversion (SVN) for code revision control, with the TortoiseSVN tool and a self-hosted instance of WebSVN. However, this provided only simple code revision control. There was a strong desire to switch to a git-based system to handle feature branches and a system that handles issue or task tracking. After reviewing the available options, a self-hosted instance of GitLab Community Eddition (CE) was chosen to handle revision control, issue tracking, collaboration, and Continuous Integration (CI).

TESTING

Standalone Testing

Standard SSEL policy dictates that each PCB board built should be thoroughly tested before applying power and before mating with other hardware. This ensures that a mistake in the design, fabrication, or assembly of the board does not cause damage to other equipment or subsystems. This prevents unnecessary repairs to the device under test (DUT) as well as other subsystems, which consumes excess resources. This standalone testing consists of an unpowered and powered Safe-To-Mate (STM), followed by powered functionality tests for each major subcircuit.

Safe-To-Mate Testing

After receiving each board from assembly, each was tested with the SFC STM procedure, SSEL document number IS-Test-0005. This test involves measuring the resistance of each pin of each connector to ground, with the board unpowered. Resistance measurements were also made between different points of the same net, to verify conductivity. Then, the voltage of each pin was measured with the board powered. This test verifies the board is functioning at a basic level and is safe to connect to other equipment.

This testing revealed the CTIP connector footprint error, mentioned in the Design chapter above. Some of the expected resistance and voltage values in the initial STM test procedure were incorrect, but shown to be acceptable. Other than that, both assembled boards passed their STM tests.

PIC Processor

After passing a STM, each board was programmed with simplified versions of the FIREBIRD CDH and MFIB software, with most functionality disabled. At this point, the IT-SPINS software implementation had not been started. The first time programming was attempted, neither microcontroller would respond to the programmer hardware. Additionally, the main microcontroller's 8MHz oscillator would only intermitently run on power up. After reviewing the schematic, layout, and microcontroller datasheet, it was found that the analog power supply (AV_{DD}) pin was left unconnected. While the analog functions of the microcontroller were not being used, this pin requires power for proper operation, according to the datasheet [21]. After installing a jumper wire to power the AV_{DD} pin, shown in Figure 6.1, each microcontroller could be programmed.

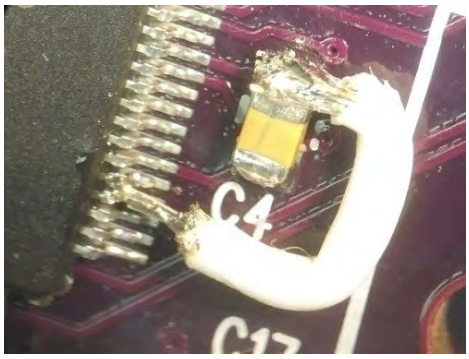

Figure 6.1: Microscope picture of jumper wire connecting the PIC24 microcontroller's AV_{DD} pin to a nearby decoupling capacitor.

The first software and hardware tested was the EGSE UART connection. This provided an interface to the InControl ground station software and worked without issue.

MET Clock Thermal Characterization

Because the TCXO MET clock was a new mission-critical design, there was a strong desire to begin long-term testing early, to gain confidence in the clock and characterize its performance. In order to characterize the TCXO over a range temperatures, one of the SFC EDUs was placed in the SSEL thermal chamber for two weeks. The first week, the chamber was set to 0°C and the second week, the chamber was set to +60°C. In addition to verifying the performance of the TCXO, this 2 week continuous runtime serves as burn-in time for the entire board.

The SFC was mounted to a PCB test stand with simple breakout board mated and a Fluke Type-K thermocouple mounted to the TCXO, as shown in Figure 6.2. USB and power were passed through the chamber feed-through. Two separate 3.3V connections were brought into the chamber so that SYS_3V3 and RTC_3V3 could be reset independently from outside the chamber. During this test, the computer running the InControl groundstation software outside the chamber was being synced to NIST using the nistime-32bit.exe program and considered time "truth."

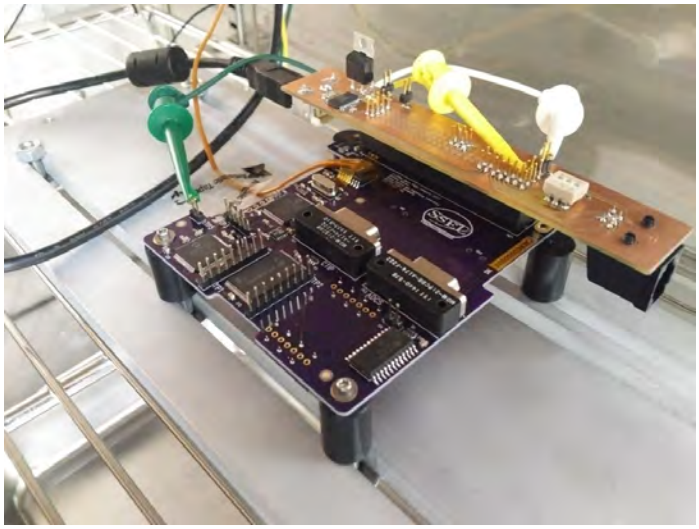

Figure 6.2: SFC board mounted in thermal testing chamber with required power and data connections.

Figure 6.3 shows the results of the MET clock testing in the thermal chamber. The clock incremented for 1 week while "cold" (0°C) and another week while "hot" (60°C). Excel's linear fit tool was used to approximate the rate at which the clock was incrementing during each temperature period.

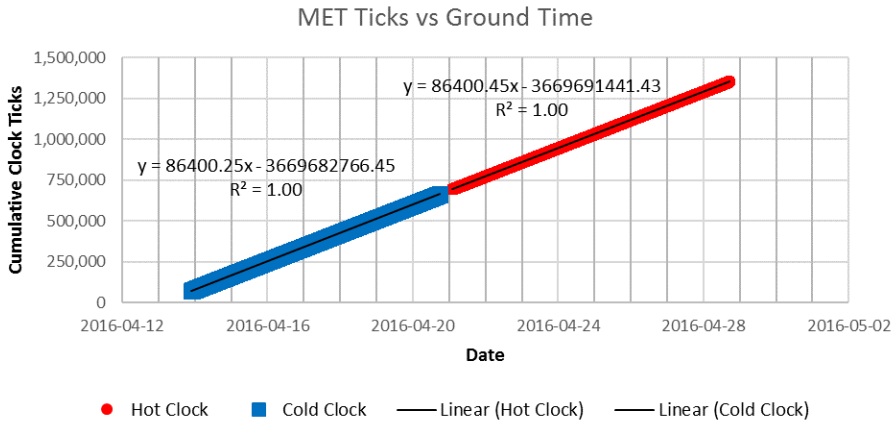

Figure 6.3: Plot of MET cumulative ticks versus elapsed time at hot and cold temperatures with linear fits.

Table 6.1 summarizes the above data. Ideally the clock would always increment exactly once per second, resulting in 86,400 ticks per day. When cold, the clock ran 2.9ppm fast and when hot it ran 5.2ppm fast. This thermal dependent error is within specifications for the TCXO [19]. Another result from this test is that the SFC was able to run for 2 weeks continuously without any latch-ups (however this was with a very simple software load).

NAND Manager

Testing of the NAND Manager and NAND flash hardware was fairly straightforward. The NAND Manager microcontroller was initially tested by attempting to program it. Once the AV_{DD} jumper had been installed on this microcontroller, as described above for the primary microcontroller, the device could easily be

Table 6.1: Summary of MET thermal test data.

	Average Ticks per Day	Absolute Error (ppm)	Estimated Drift (sec. per month)
Cold Tick Rate	86400.247758	2.8676	7.5360
Hot Tick Rate	86400.451940	5.2308	13.7464
Relative Error	0.204182	2.8676	7.5360

programmed.

Special care was taken in each revision of code flashed to the NAND Manager microcontroller to avoid performing write operations to the NAND flash. The NAND flash comes from the factory blank; however, defective blocks of memory are marked by the manufacture. Before performing any write operations, read operations were performed on the entire flash, to detect any manufacture-marked bad blocks marks. No manufacture-marked bad blocks have been detected on the SFCs.

Once the NAND Manager microcontroller code bad been sufficiently developed, long term tests were performed to verify the NAND Manager microcontroller and NAND flash could support several telemetry IDs being written at flight-like cadences for several days consecutively. No significant issues have been found. Note that this test has only been performed on the engineering model (EM) SFC, as write/erase cycles will be limited on the flight model (FM) SFC, to allow a longer on-orbit life. Readback and downlink of data has also been performed; however, the ground station software has not yet been sufficiently developed to characterize the validity and efficiency of these downlinks.

Experimental IMU

Because the experimental IMU is not mission critical, only minimal testing has been performed. Readback of the 3-axis accelerometer, 3-axis gyroscope, and temperature sensor have been verified through telemetry. Only simple tests rotating the SFC by hand have been used to verify the response of these devices. One issue that has not been resolved is the readback of the 3-axis magnetometer, which is only intermittently working.

Inter-Subsystem Testing

Once stand-alone testing was completed and the required software had been implemented, the SFC was integrated and tested with all other subsystems on the satellite one at a time. The SFC has passed all tests performed with the CTIP, ADCS, EPS, and COMM subsystems. However, at this time the SCE has not been built, and therefore, has not been tested.

CTIP Payload

CTIP was one of the first external subsystems tested with the SFC, since a CTIP EM was available early in the development of the mission. A custom cable was made to interface between the SFC and CTIP, which resolved the footprint error on the SFC's CTIP connector. This interface proved to be functional, with all CTIP commands and telemetry working well.

MAI ADCS Development Unit

In order to test the MAI-400 interface on the SFC, a cable was made to connect the SFC to the MAI-400 development unit, which contains only the main circuit board and processors, with no sensors or actuators. This interface proved to be

functional, with most commands and telemetry working well. There were several bugs in the MAI-400 interface found, including an invalid temperature telemetry point and invalid packet checksum. These bugs have workarounds implemented for now, and MAI intends to resolve them in their final firmware release.

Electrical Power System

Testing of the SFC with the EPS was delayed, as a flight-like fully functional EPS was not complete until very late in the mission design cycle. However, once an EPS had been built, all software and power interfaces to the SFC proved to be functional. This interface was tested by mating an EPS and SFC through the system bus connector, and verifying that SFC could read and write to all command registers of the I2C interface and that the commands turned on or off the desired rail.

COMM

Testing of the SFC with the AstroDev COMM subsystem was also delayed, as only a flight model COMM board was purchased due to budget constraints. Additionally, the Li-1 radio order arrived several months late and turned out to be a Li-2, with different mechanical, electrical, and software interfaces, which has yet to be defined by the manufacture. This forced a resign of the carrier board for the radio, since it had already been designed and manufactured to meet the published specifications of the AstroDev Li-1 radio. Basic testing of the UART command and telemetry interface of the Li-2 proved the device could properly receive radio transmission and pass them to the SFC, as well as transmit data provided to it. The configuration registers could also be written to and readback via the UART interface.

Software Testing

Throughout the implementation of the flight software code base, its stability was tested regularly. Each time a significant feature was implemented, the SFC would be run for several days continuously with as many subsystems connected and in use as possible. This verified the new feature or change had not adversely affected other modules and was performing as expected. Added commands and telemetry would be tested with the InControl ground station software.

CONCLUSION

Timely and accurate prediction of space weather storms are extremely important to maintaining today's space and ground based infrastructure. Current models of the Earth's ionosphere and its interaction with the Sun as a coupled system lack the in situ measurements needed to improve predictions. The IT-SPINS mission will provide line-of-sight measurements of 135.6-nm nightglow emissions from O^+/electron recombination from a 3U CubeSat in LEO. Post processing will yield 2D altitude versus in-track images of the O^+ density. These 2D images will further our understanding of the TTR, as well as mesoscale structures like EPB, and polar patches and will provide weighting parameters for space weather models.

The IT-SPINS Space Flight Computer has been designed, built, and tested at the Space Science and Engineering Laboratory at Montana State University. The SFC hardware and software designs leverage heritage from SSEL's past CubeSat missions, while fulfilling the unique command and data handling requirements of the IT-SPINS mission. All tests to date have shown the SFC to meet its subsystem requirements, derived from the top-level mission requirements.

Lessons Learned

While the PIC select DEMUX design works as intended, as other subsystems were added to the avionics stack, each with their own copy of the circuit, several flaws became apparent. The static power consumption of the DEMUX, while only a few milliwatts, was duplicated on each board in avionics stack. Additionally, this circuit is only used for programming on the ground, so it acts only as a complicated pull-up resistor on orbit. For future designs it is suggested to move the DEMUX circuit out of the satellite to the external EGSE, and route the output of the DEMUX through

the EGSE harness. This would reduce power consumption, simplify the design, and yield more layout space.

As mentioned in chapter 3, the NAND flash architecture was found to not be a good fit for storing the small commands sequences and system state information. This is because the NAND flash used requires half-megabyte block of memory to be erased at a time. The solution was to add a NOR flash, identical to the one used on FIREBIRD's CDH, to the SFC with a small daughterboard.

My Contribution

Expect where noted, the described work of designing, building, software implementation, and testing for the IT-SPINS SFC is my own. However, I did receive guidance, review, and assistance from the staff and students of the MSU SSEL throughout the process.

Future Work

Because the SCE hardware has not yet been built, the SCE software module has not been fully tested. Currently the SCE software module has been developed based on the FIREBIRD FISCE software while incorporating lessons learned from the IT-SPINS EPS software. Once the SCE has passed all stand-alone tests, it should be integrated and tested with the SFC.

While the radio interface software on the SFC has been tested, the InControl ground station processing of stored data downlink has not been completed. Due to several changes in the packet structure of stored data downlinks, InControl's IT-SPINS mission interface will need to be updated to accept the new structure.

The specifics of the mission concept of operations (data collection timing,

required ADCS data for tomographic reconstruction, etc..) still need to be defined. Once defined, this will allow the flight command sequences and custom telemetry IDs to be developed and tested in software.

Finally, there are two known bugs in the software which should be resolved before flight. First, the NAND Manager's bad block management will incorrectly flag some blocks as bad during reformatting of the NAND flash. Second, the system has been seen to freeze after running for several days with many telemetry IDs configured to be sent out of EGSE at high cadences; however, this is not a major issue, as the EPS's WDT will reset the system every 12 hours.

REFERENCES CITED

[1] D. Klumpar, "Cubesat lessons learned - two launch failures followed by one mission success," in *Proceedings of the 26th Annual AIAA/USU Conference on Small Satellites*, Aug. 2012.

[2] D. Klumpar, L. Springer, E. Mosleh, K. Mashburn, S. Berardinelli, A. Gunderson, M. Handley, N. Ryhajlo, H. Spence, S. Smith, *et al.*, "Flight system technologies enabling the twin-cubesat firebird-ii scientific mission," in *Proceedings of the 29th Annual AIAA/USU Conference on Small Satellites*, Aug. 2015.

[3] J. Hanson, A. G. Luna, R. DeRosee, K. Oyadomari, J. Wolfe, W. Attai, and C. Prical, "Nodes: A flight demonstration of networked spacecraft command and control," in *Proceedings of the 30th Annual AIAA/USU Conference on Small Satellites*, Aug. 2016.

[4] J. G. Kappenman, "Geomagnetic storms and their impact on power systems," *IEEE Power Engineering Review*, vol. 16, no. 5, p. 5, 1996.

[5] C. J. Mertens, B. T. Kress, M. Wiltberger, S. R. Blattnig, T. S. Slaba, S. C. Solomon, and M. Engel, "Geomagnetic influence on aircraft radiation exposure during a solar energetic particle event in october 2003," *Space Weather*, vol. 8, no. 3, 2010.

[6] "Heliophysics: The new science of the sun-solar system connection. recommended roadmap for science and technology 2005-2035," Tech. Rep. NASA/NP-2005-11-740-GSFC, PB2009-102795, NASA, Washington, DC United States, January 2005.

[7] R. A. Heelis, W. R. Coley, A. G. Burrell, M. R. Hairston, G. D. Earle, M. D. Perdue, R. A. Power, L. L. Harmon, B. J. Holt, and C. R. Lippincott, "Behavior of the o+/h+ transition height during the extreme solar minimum of 2008," *Geophysical Research Letters*, vol. 36, no. 18, pp. n/a–n/a, 2009. L00C03.

[8] S. A. Gonzlez, M. P. Sulzer, M. J. Nicolls, and R. B. Kerr, "Solar cycle variability of nighttime topside helium ion concentrations over arecibo," *Journal of Geophysical Research: Space Physics*, vol. 109, no. A7, pp. n/a–n/a, 2004. A07302.

[9] M. R. Hairston, W. R. Coley, and R. Heelis, "Mapping the duskside topside ionosphere with cindi and dmsp," *Journal of Geophysical Research: Space Physics*, vol. 115, no. A8, 2010.

[10] J. Comberiate and L. J. Paxton, "Coordinated uv imaging of equatorial plasma bubbles using timed/guvi and dmsp/ssusi," *Space Weather*, vol. 8, no. 10, 2010. S10002.

[11] S. Basu, K. Groves, S. Basu, and P. Sultan, "Specification and forecasting of scintillations in communication/navigation links: current status and future plans," *Journal of Atmospheric and Solar-Terrestrial Physics*, vol. 64, no. 16, pp. 1745 – 1754, 2002. Space Weather Effects on Technological Systems.

[12] C. G. Sondecker and M. J. Kim, "Sense: The usaf smc/xr space environmental nanosatellite experiment," in *30th Space Symposium*, 2014.

[13] R. Roble and E. Ridley, "A thermosphere-ionosphere-mesosphere-electrodynamics general circulation model (time-gcm): Equinox solar cycle minimum simulations (30–500 km)," *Geophysical Research Letters*, vol. 21, no. 6, pp. 417–420, 1994.

[14] A. B. Crew, H. E. Spence, J. B. Blake, D. M. Klumpar, B. A. Larsen, T. P. O'Brien, S. Driscoll, M. Handley, J. Legere, S. Longworth, *et al.*, "First multipoint in situ observations of electron microbursts: Initial results from the nsf firebird ii mission," *Journal of Geophysical Research: Space Physics*, vol. 121, no. 6, pp. 5272–5283, 2016.

[15] J. C. Springmann, B. P. Kempke, J. W. Cutler, and H. Bahcivan, "Development and initial operations of the rax-2 cubesat," in *ESA/CNES Small Satellites Systems and Services Symposium*, 2012.

[16] A. K. Gunderson, "Design, fabrication, and implementation of the energetic particle integrating space environment monitor instrument," Master's thesis, Montana State University-Bozeman, College of Engineering, 2014.

[17] Texas Instruments, *High Temperature 3V LVDS Differential Line Receiver*, 2013. DS90LT012AH datasheet.

[18] Linear Technology, *60V Fault Protected 3V to 5.5V RS485/RS422 Transceivers*, 2011. LTC2865 datasheet.

[19] Maxim Integrated, *32.768kHz Temperature-Compensated Crystal Oscillator*, 2007. DS32kHz datasheet.

[20] Maxim Integrated, *I2C, 32-Bit, Binary Counter Clock with 64-Bit ID*, 2007. DS1372 datasheet.

[21] Microchip, *64/100-Pin, 16-Bit Flash Microcontrollers with USB On-The-Go (OTG)*, 2010. PIC24FJ256GB210 family datasheet.

APPENDICES

APPENDIX A

SFC SUBSYSTEM LEVEL REQUIREMENTS

Table A.1: IT-SPINS SFC System Interface Requirements

Requirement Number	Baseline Requirement	Source
SFC-1	The SFC shall interface with the CTIP payload utilizing a dedicated 5V RS-422 link for command and telemetry. The RS-422 signals will be converted to UART to interface with the SFC.	Derived
SFC-2	The SFC shall interface with the MAI-400 ADCS utilizing a dedicated 3.3V UART link for command and telemetry.	Derived
SFC-3	The SFC shall interface with the AstroDev radio utilizing a dedicated 3.3V UART link for command and telemetry.	Derived
SFC-4	The SFC shall interface with the EPS utilizing a 3.3V I2C link for command and telemetry.	Derived
SFC-5	The SFC shall interface with the NAND Manager utilizing a 3.3V SPI link for command and telemetry.	Derived
SFC-6	The SFC shall interface with the solar cell experiment utilizing a 3.3V I2C link for command and telemetry.	Derived
SFC-7	The SFC shall interface with the ground support equipment utilizing a dedicated 3.3V UART link for command and telemetry.	Derived

Table A.2: IT-SPINS SFC Time Management Requirements

Requirement Number	Baseline Requirement	Source
SFC-8	The SFC shall utilize a system clock with less than 8ms measurement resolution and temperature stability less than TBD parts per million.	Derived

Table A.3: IT-SPINS SFC Data Handling and Storage Requirements

Requirement Number	Baseline Requirement	Source
SFC-9	The SFC shall have non-volatile, partitioned data storage capacity for both mission science and spacecraft telemetry.	Derived

Table A.4: IT-SPINS SFC Command Processing Requirements

Requirement Number	Baseline Requirement	Source
SFC-10	The SFC shall receive, authenticate, process and distribute commands from GSE, COMM and stored command sequences to individual subsystems and the science payload.	Derived
SFC-11	The SFC shall be able to reject erroneous commands.	Derived
SFC-12	The SFC shall utilize a stored command sequencer for autonomous start-up (including activation of deployables) following deployment from the launch canister.	Derived

Table A.5: IT-SPINS SFC Telemetry Requirements

Requirement Number	Baseline Requirement	Source
SFC-13	The SFC shall collect, packetize and store time-tagged telemetry from all subsystems and the science payload in real-time.	Derived
SFC-14	The SFC shall transmit current-value telemetry to GSE, COMM or non-volatile storage on demand and at a configurable rate.	Derived
SFC-15	The SFC shall transmit specific stored telemetry to GSE or COMM upon request	Test Plan
SFC-16	The SFC shall utilize spacecraft telemetry monitors with configurable trigger conditions to execute autonomous corrective or protective action against anomalous system behavior.	Derived

APPENDIX B

MODULE FILE TEMPLATES

```
1  /*****************************************************************************
2   *
3   *              Space Science and Engineering Laboratory
4   *                      Montana State University
5   *
6   *                          Flight Software
7   *
8   * Filename       : <ModuleName>.h
9   * Description    : Header file for <ModuleName> module
10  * Authors(s)     : {author}
11  * Date Created   : {date}
12  *****************************************************************************/
13
14  // This is a guard condition so that contents of this file are not included
15  // more than once.
16  #ifndef <ModuleName>_H
17  #define <ModuleName>_H
18
19  /*****************************************************************************
20  *                                    INCLUDES
21  *****************************************************************************/
22  #include "os/ucos_ii.h"
23
24  /*****************************************************************************
25  *                                    DEFINES
26  *****************************************************************************/
27  #define <ModuleName>_STK_SIZE     256u
28  #define <ModuleName>_MAX_MSGS     SYS_NUM_PKTS
29
30  #define <ModuleName>_FLAG_MSG_Q   0x01
31
32  /* Commands */
33  enum E_<ModuleName>_COMMANDS
34  {
35      E_<ModuleName>_NOOP                     = 0,   /* Noop command */
36      E_<ModuleName>_RESET_SOH                = 1,   /* Reset SOH Struct command */
37      E_<ModuleName> */
38  };
39
40  /* Module state of Health */
41  typedef struct _<ModuleName>SoH_
42  {
43      INT8U    cmd_cnt;
44      INT8U    inv_cmd_cnt;
45      INT8U    last_cmd;
46      INT32U   bytes_sent;
47      INT32U   bytes_rcvd;
48  }_<ModuleName>SoH;
49
50  /*****************************************************************************
51  *                                    EXTERNALS
52  *****************************************************************************/
53  extern OS_STK            <ModuleName>Stk[];
54  extern OS_EVENT        * <ModuleName>Queue;
55  extern OS_FLAG_GRP     * <ModuleName>Events;
56  extern void            * <ModuleName>MsgArray[];
57
58  extern _<ModuleName>SoH;
59
60  /*****************************************************************************
61  *  PROTOTYPES
62  *****************************************************************************/
63  void     <ModuleName>Task            ( void * pdata );
64  void     <ModuleName>SetFlag         ( INT8U flags );
65  INT8U    <ModuleName>QPost           ( void * msg );
66
67  #endif /* <ModuleName>_H */
```

```
1  /*****************************************************************************
2   *
3   *              Space Science and Engineering Laboratory
4   *                      Montana State University
5   *
6   *                          Flight Software
7   *
8   * Filename       : <ModuleName>_task.c
9   * Description    : Task file for <ModuleName> Module
10  * Authors(s)     : {author}
11  * Date Created   : {date}
12  *****************************************************************************/
13
14 #include "os/ucos_ii.h"
15 #if TASK_<ModuleName>_EN
16 #include "modules/<ModuleName>.h"
17 #include "modules/<ModuleName>_driver.h"
18 #include "modules/system/system.h"
19
20 OS_STK       <ModuleName>Stk[<ModuleName>_STK_SIZE]; /* Task Stack */
21 void         * <ModuleName>MsgArray[<ModuleName>_MAX_MSGS]; /* Task Message Array*/
22 OS_EVENT     * <ModuleName>Queue;
23 OS_FLAG_GRP * <ModuleName>Events;
24
25 extern void <ModuleName>ProcessLocalMsg  ( PACKET * msg );
26
27 extern void <ModuleName>Init             ( void );
28
29 void <ModuleName>Task ( void * pdata )
30 {
31     /* local vars */
32     INT8U err;
33     PACKET * msg;   /* message variable for task queues */
34     OS_FLAGS flags; /* Variable to get Events from the OS */
35
36     /* Create event flag group */
37     <ModuleName>Events = OSFlagCreate(0x00, &err);
38     if ( <ModuleName>Events == (OS_FLAG_GRP*)0 )
39     {
40 #ifdef __DEBUG
41         __builtin_software_breakpoint();// ERROR
42 #endif
43     }
44
45      /* Create Message queue */
46     <ModuleName>_MAX_MSGS);
47     if ( <ModuleName>Queue == (OS_EVENT*)0 )
48     {
49 #ifdef __DEBUG
50         __builtin_software_breakpoint();// ERROR
51 #endif
52     }
53
54     /* init <ModuleName> hardware */
55     <ModuleName>Init();
56
57     /* Never leave this loop */
58     while ( 1 )
59     {
60         /* Wait forever for any of these events */
61         flags = OSFlagPend(<ModuleName>Events,
62                            /* wait for flag */
63                            <ModuleName>_FLAG_MSG_Q,
64                            OS_FLAG_WAIT_SET_ANY | OS_FLAG_CONSUME,
65                            0, /* wait forever */
66                            &err);
67
68         /* If valid event flag(s) was read, handle each one */
69         if ( err == OS_NO_ERR )
70         {
71             /* Message queue event */
72             if ( flags & <ModuleName>_FLAG_MSG_Q )
73             {
74                 /* Read and process all messages in queue */
75                 msg = (PACKET*)OSQAccept(<ModuleName>Queue, &err);
76                 while(err == OS_ERR_NONE)
77                 {
78                     <ModuleName>ProcessLocalMsg(msg);
79
80                     /* Get the next message, if there is one */
81                     msg = (PACKET*)OSQAccept(<ModuleName>Queue, &err);
82                 }
83             }
84         }
85         else
86         {
87 #ifdef __DEBUG
88             __builtin_software_breakpoint();// ERROR
```

```
89  #endif
90          }
91      }
92  }
93
94  void <ModuleName>SetFlag( INT8U flags )
95  {
96      /* local vars */
97      INT8U err;
98
99      /* Set the OS flags */
100     OSFlagPost(<ModuleName>Events, flags, OS_FLAG_SET, &err);
101
102     /* check for error */
103     if(err != OS_ERR_NONE)
104     {
105 #ifdef _DEBUG
106         __builtin_software_breakpoint();// ERROR
107 #endif
108     }
109 }
110
111 /*****************************************************************************
112 * Function      : INT8U <ModuleName>QPost ( void * msg )
113 * Description   : Send a message to a message queue and set an event flag
114 * Arguments     : void * msg   - message being posted to the task
115 * Returns       :
116 * Remarks       :
117 *****************************************************************************/
118 INT8U <ModuleName>QPost ( void * msg )
119 {
120     INT8U err;
121
122     err = OSQPost(<ModuleName>Queue, msg);
123     if (err == OS_ERR_NONE)
124     {
125         /* Set an event */
126         (void)OSFlagPost(<ModuleName>_FLAG_MSG_Q, OS_FLAG_SET, &err);
127
128         /* check for error */
129         if(err != OS_ERR_NONE)
130         {
131 #ifdef _DEBUG
132             __builtin_software_breakpoint();// ERROR
133 #endif
134         }
135     }
136     else
137     {
138 #ifdef _DEBUG
139         __builtin_software_breakpoint();// ERROR
140 #endif
141     }
142     return (err);
143 }
144 #endif
```

```
1  /*******************************************************************************
2   *
3   *              Space Science and Engineering Laboratory
4   *                      Montana State University
5   *
6   *                          Flight Software
7   *
8   * Filename       : <ModuleName>_cmds.c
9   * Description    : Commands for <ModuleName> Module
10  * Authors(s)     : {author}
11  * Date Created   : {date}
12  *******************************************************************************/
13
14 #include "os/ucos_ii.h"
15 #if TASK_<ModuleName>_EN
16 #include "modules/<ModuleName>.h"
17 #include "modules/<ModuleName>_driver.h"
18 #include "modules/system/system.h"
19
20 /***********************************
21  * Local Variables for the System
22  ***********************************/
23 _<ModuleName>SoH;
24
25 /*******************************************************************************
26 * Function              : void <ModuleName>NoOp ( void )
27 * Description           :
28 * Arguments             :
29 * Returns               : none
30 * Remarks               : none
31 *******************************************************************************/
32 void <ModuleName>NoOp ( void )
33 {
34     ;
35 }
36
37 /*******************************************************************************
38 * Function              : void <ModuleName>ResetSoH ( void )
39 * Description           :
40 * Arguments             :
41 * Returns               : none
42 * Remarks               : none
43 *******************************************************************************/
44 void <ModuleName>ResetSoH ( void )
45 {
46     <ModuleName>SoH.cmd_cnt = 0;
47     <ModuleName>SoH.inv_cmd_cnt = 0;
48     <ModuleName>SoH.last_cmd = 0;
49 }
50
51 /*******************************************************************************
52 * Function              : void <ModuleName>Init ( void )
53 * Description           :
54 * Arguments             :
55 * Returns               : none
56 * Remarks               : none
57 *******************************************************************************/
58 void <ModuleName>Init ( void )
59 {
60     <ModuleName>_Init();
61 }
62
63 /*******************************************************************************
64 * Function      : void <ModuleName>ProcessLocalMsg ( PACKET * msg )
65 * Description   : Process a message from another module to this module
66 * Arguments     : PACKET * msg - message from another module
67 * Returns       :
68 * Remarks       :
69 *******************************************************************************/
70 void <ModuleName>ProcessLocalMsg ( PACKET * msg )
71 {
72     BOOLEAN msg_valid = OS_TRUE;
73
74     /* Verify destination */
75     if ( GetINT8U(msg+DESTID_OFFSET) == E_MODULE_ID_<ModuleName> )
76     {
77         /* Process the packet depending on the packet type */
78         if ( GetINT8U(msg+CMDTLM_OFFSET) == E_CMD_PKT )
79         {
80             <ModuleName>SoH.last_cmd = GetINT8U(msg+FUNCID_OFFSET);
81
82             /* Handle specified command */
83             switch ( GetINT8U(msg+FUNCID_OFFSET) )
84             {
85                 /* COMMANDS */
86                 case E_<ModuleName>_NOOP:
87                     <ModuleName>NoOp();
88                     break;
```

```
89
90                      case E_<ModuleName>_RESET_SOH:
91                          <ModuleName>ResetSoH();
92                          break;
93
94                      case E_<ModuleName>_INIT:
95                          <ModuleName>Init();
96                          break;
97
98                      default:
99                          /* Invalid command */
100                         msg_valid = OS_FALSE;
101                         break;
102                 }
103             }
104             else if ( GetINT8U(msg+CMDTLM_OFFSET) == E_TLM_PKT )
105             {
106                 ;
107             }
108             else
109             {
110                 /* Invalid command */
111                 msg_valid = OS_FALSE;
112             }
113         }
114         else
115         {
116             msg_valid = OS_FALSE;
117         }
118
119         if ( msg_valid )
120         {
121             <ModuleName>SoH.cmd_cnt++;
122         }
123         else
124         {
125             <ModuleName>SoH.inv_cmd_cnt++;
126         }
127
128     #if TASK_SYSTEM_EN
129         /* always release the message */
130         SystemReleasePacket((void*)msg);
131     #endif
132     }
133     #endif
```

```
1  /*****************************************************************************
2   *
3   *              Space Science and Engineering Laboratory
4   *                      Montana State University
5   *
6   *                          Flight Software
7   *
8   * Filename       : <ModuleName>.h
9   * Description    : Header file for <ModuleName> driver
10  * Authors(s)     : {author}
11  * Date Created   : {date}
12  *****************************************************************************/
13 #ifndef <ModuleName>_DRIVER_H
14 #define <ModuleName>_DRIVER_H
15 /*****************************************************************************
16  *                                    INCLUDES
17  *****************************************************************************/
18
19 /*****************************************************************************
20  *                                    DATATYPES
21  *****************************************************************************/
22
23 /*****************************************************************************
24  *                                    EXTERNS
25  *****************************************************************************/
26
27 /*****************************************************************************
28  *                                    PROTOTYPES
29  *****************************************************************************/
30
31 /*****************************************************************************
32  *                                    DEFINES
33  *****************************************************************************/
34
35 /*****************************************************************************
36  *                                    Macros
37  *****************************************************************************/
38
39 /*****************************************************************************
40  *                                    PROTOTYPES
41  *****************************************************************************/
42 void    <ModuleName>_Init                       ( void );
43
44 #endif /* __<ModuleName>_driver_h */
```

```
1  /*****************************************************************************
2   *
3   *              Space Science and Engineering Laboratory
4   *                      Montana State University
5   *
6   *                          Flight Software
7   *
8   * Filename       : <ModuleName>.c
9   * Description    : Driver for <ModuleName>
10  * Authors(s)     : {author}
11  * Date Created   : {date}
12  *****************************************************************************/
13
14  #include "os/ucos_ii.h"
15 #if TASK_<ModuleName>_EN
16 #include "modules/<ModuleName>.h"
17 #include "modules/<ModuleName>_driver.h"
18
19  /*****************************************************************************
20  * Function            : void <ModuleName>_Init ( void )
21  * Description         : Initialized the <ModuleName> Driver
22  * Arguments           : None
23  * Returns             : None
24  * Remarks             : none
25  *****************************************************************************/
26 void <ModuleName>_Init ( void )
27 {
28      //TODO: implement
29 }
30 #endif
```