Firecode Receiver for Small Satellite Systems

McGill University

Advised by Prof. Frank F. Ferrie

In Association with **Astro Digital** (formerly Aquila Space, Inc.)

Derek Yu, *260570997*
Markus Suorsa, *260551931*

22 November 2016

Abstract

Within the last decade, the small satellite industry has been able to develop satellites utilizing newer components and systems. As a result, it is likely that satellites will encounter glitches that render the spacecraft inoperable. To re-establish communication and resume operation, an independent system to reset all systems may be used. Markus Suorsa and Derek Yu of McGill University, with association with Astro Digital, are developing the FRS3 as a possible solution. The FRS3 is an open source project with a small footprint and a configurable design. It takes the form of a PCB and antenna acting as an independent system upon a spacecraft. The FRS3 utilizes a low-power Nominal Operation with periodic sleep-RX state to receive and decrypt unique signals before emitting a reset signal to the rest of the spacecraft.

Acknowledgements

Thank you to Kyle Leveque, Gordon Hardman, and Jan King, and all of Astro Digital for their guidance in this project. Also thank you to our supervisor, Professor Frank Ferrie for his insight and review.

List of Acronyms

AES Advanced Encryption Scheme

ASK Amplitude-shift Keyring

CAD Computer Aided Design

CSA Canadian Space Agency

CPU Central Processing Unit

DFN Dual Flat No-leads

DMR Dual Module Redundancy

EIRP Effective Isotropic Radiated Power

ESA European Space Agency

FCC United States Federal Communications Commission

FRS3 Firecode Receiver for Small Satellite Systems

FSK Frequency-shift Keyring [**FSM**] Finite State Machine

GEO Geosynchronous Earth Orbit

GFSK Gaussian Frequency-shift Keyring

GMSK Gaussian Minimum-shift Keyring

GRS Ground Reset Signal

GUI Graphical User Interface

GND Ground

IC Integrated Circuit

IRQ Interrupt Request

I/O Input / Output

LED Light Emitting Diode

LDO Low Dropout

LDCR Low Duty Cycle Reload

LNA	Low Noise Amplifier
LEO	Low Earth Orbit
MEO	Medium Earth Orbit
MCU	Micro-controller Unit
MSK	Minimum-shift Keyring
NASA	National Aeronautics and Space Administration
OOK	On-off Keyring
PC	Personal Computer
PCB	Printed Circuit Board
RCM	RADARSAT Constellation Mission
RF	Radio Frequency
RX	Receive
RSSI	Radio Signal Strength Indicator
SOT	Small Outline Transistor
SPI	Serial Peripheral Interface
SNR	Signal Noise Ratio
SMA	SubMiniature version A
SAR	Synthetic Aperture Radar
TMR	Tri-Module Redundancy
USB	Universal Serial Bus
QFN	Quad Flat No-leads
XML	eXtensible Markup Language.

Contents

Chapter 1

Motivation

With continuing advancements in technology, two things tend to change. First, technological performance increases, providing greater functionality at cheaper costs. Second, as transistor sizes decrease, electronic component footprints tend to shrink. This is the phenomenon behind Moore's Law. Although primarily directed towards the semiconductor industry, the idea is applicable to many different fields of engineering. Until recently, the satellite aerospace engineering industry has resisted such change, instead opting to continue using outdated yet reliable technology to decrease the risk of using newly developed, potentially volatile systems on multi-million dollar satellites. Many base components and systems found on satellites launched in the 90's and 00's are exactly the same as those in the 70's and 80's. Recent developments in the satellite and launch industries have introduced much cheaper platforms for payloads and launches, decreasing the cost involved in satellite development significantly. As a result, more and more satellites are testing experimental components. The risk involved in using these components are less threatening than before, as launch and integration costs have decreases due to smaller satellite platforms. However, the risk still does exist, and because launch costs are still considerable, there must be a contingency in place to mitigate the risk.

In the last decade or so, design paradigm has shifted from a hardware-based approached towards a software-based approach. New and experimental technologies are heavily software based, while the hardware aspects are similar to older iterations with only marginal changes. Thus, when issues arise on spacecraft in orbit, they are much more likely to be software-based than hardware-based. Fortunately, software-based issues may be fixable post-launch, but this varies depending the nature of the problem. Fixes may be uploaded to the spacecraft during a ground pass (please refer to the background for further reading), but if the spacecraft is otherwise unresponsive, it is impossible. To resume nominal operation and regain communication, a simple power cycle or reset would be sufficient. However, due to the location of a spacecraft in orbit, it is not practical to physically hit a reset switch. It is possible to program the spacecraft to reset all systems at the reception of a particular radio message, but it would not work if the flight computer is unable to process it, or if the communication system is the source of the problem. Therefore, a dedicated, independent system that would listen for such a message to reset the spacecraft and be able to do so despite the state of the rest of the spacecraft would be needed. This is the goal of the project.

A need for an emergency reset receiver can be clearly illustrated in the LightSail-1

mission. In 2015, the Planetary Society launched a CubeSat containing an experimental solar sail into orbit, with its primary mission objective to test the solar sail deployment module. The satellite was in the 3U CubeSat format and was developed by Stellar Explorations. Two days after launch, the spacecraft suffered a debilitating software glitch that rendered it unresponsive [2]. For several weeks, LightSail remained inoperable until a "natural reboot" occurred, in which the spacecraft was reset by a charged particle in space hitting the electronics in just the right way to force a reboot. Afterwards, the software was updated from the ground and LightSail was fortunately able to complete its mission.

Although the LightSail-1 mission was ultimately successful, there was a period where the recovery of the satellite was in question. When any mission goes awry as a result of technical problems, it is generally not proper protocol to rely on the occurrence of a "natural reboot," as LightSail did. If LightSail did contain an emergency reset receiver, as soon as it was clear that the satellite was unresponsive, the team on the ground could have emitted the reset signal to the satellite. The receiver would ideally have received the signal and promptly rebooted the satellite, eliminating the problem.

Chapter 2

Background

This section details some areas which may be of relevance regarding the development and context for the FRS3. This is by no means an exhaustive list, and are limited in their scope.

2.1 Low Earth Observation and Applications

The two most popular orbits for earth-orbiting spacecraft are Geosynchronous Earth Orbits (GEO) and Low Earth Orbits (LEO). GEO satellites exist in an altitude of about 36,000km at which its orbital period matches that of Earth's rotation, effectively placing it above constant location on Earth. This is common for telecommunication satellites and less common for remote sensing. If a satellite is said to be in LEO it means that it is at altitude of 200 - 2000km above sea level. LEO satellites travel in fast orbits across Earth, with the average orbital period of within a couple of hours. This makes LEO satellites an attractive option for earth observation platforms, particularly if placed in a polar orbit. In such a configuration, a spacecraft could eventually image the entire surface of Earth. Satellite constellations such as Canada's RCM, and ESA's Sentinel program are used to gather scientific data about the surface of the earth, with notable applications being surface sea temperatures, area covered by rain-forest, and the thickness of polar ice caps.

2.2 The CubeSat Platform

Miniaturized satellite platforms vary in size and weight, ranging from "small satellites" (100 - 500kg) to "femtosatellites" (10 - 100g). The CubeSat platform populates this scale. CubeSats are made up of units U to mean one litre of useful space in a $10 \times 10 \times 10cm^3$ package. Smaller CubeSats are of the $1U$ variety, whereas typical CubeSats are around the $3U+$ range. This smaller size means that costs are kept low in a field where the average cost of delivery into space ranges from millions for small payloads to billions for larger satellites.

2.3 Common Satellite Operations

Satellites travel in trajectories across the surface of the earth in orbits, with each time they travel over the same location considered a ground pass. Due to the curvature

of the earth and their low orbit, they generally have a visual range of a few hundred kilometers, in which they are also able to communicate with a ground station. This small visual range (relative to the size of the earth) couple with the speed at which the satellite travels, means that satellites are only in contact with a permanent base for minutes between passes. While this makes communication difficult, it means that a satellite can traverse a large areas and collect impressive amounts of data.

2.4 RF Communications in Space

Due to the nature of the distance between receiver and the sender, communications in space are typically done at a high frequency, ranging from hundreds of MHz to tens of GHz. The transmission intensity from the ground station to the satellite (uplink) or from the satellite back down to earth (downlink) must be high enough to be able to take into consideration all the disturbances - the directionality, gain of the antenna, the direction of the ground satellite, the Doppler effect when the satellite is approaching versus travelling away, even the condition of the weather such as clouds and particulates in the atmosphere! Units of this intensity are generally given in decibels (dB) which are defined by the formula :

$$10log(\frac{p}{p_0}) \tag{2.1}$$

Where the p is the power by the system and p_0 is some reference power, generally one Watt. This logarithmic scale makes it more convenient to discuss changes in power. An important reference is that a drop of about 3dB is equivalent to a drop in half of the transmitted power. This power plays a key role during calculations of communication both the *power flux density* (or in specific cases, called *intensity*) and then *bandwidth*. The power flux density is the strength of the signal that hits the antenna array. The bandwidth is the range of frequencies for which the signal can be heard.

Keeping track of the variables which could affect the gain and the bandwidth are put together in a document commonly known as the link budget. This is further elaborated upon in Section 4.5.

2.5 Schematics and PCB Layout

The process from a functional block diagram to a completed design is long; it requires a the creation of a schematic, which requires knowledge of the connection between each component associated with the product; and the PCB Layout, where physical challenges such as routing, placing of components, and space efficiency come into play. The complexity of the design increases exponentially with the amount of components and the type of components that are used. The software used for this component is KiCAD, a free and open source software in keeping with the original goal of this project.

2.6 MicroPython

MicroPython is a basic implementation of Python 3 for use in MCUs [1]. The PyBoard, as referred to later, is the official micro-controller board designed and sold by the MicroPython project. It has full support for all MicroPython features.

2.7 AES Encryption

AES is a complicated form of encryption that is vital to day-to-day secure communication. It uses a symmetric key that is shared between two users and hidden from observers to ensure secure public communication. The internals of AES are such that it attempts to deter hackers through diffusion and obfuscation of the original message, the properties of which are much beyond the scope of this paper. The form used in this paper is the following:

$$M_{encrypted} = AES(E_{key}, M_{unencrypted}) \tag{2.2}$$

Where $AES()$ represents the AES function that takes as an argument the symmetric key E_{key}, and the message (or plain-text) that it is expected to encrypt $M_{unencrypted}$, and the output is the encrypted message $M_{encrypted}$. If both the encryption key and the encrypted message is known, than the reverse can be computed through the following:

$$M_{unencrypted} = AES^{-1}(E_{key}, M_{encrypted}) \tag{2.3}$$

The level of encryption of AES depends on the strength of it's key, which range from 128 to 256 bits in size in multiples of 32 bits, and are such that a brute force approach to decryption without the key is technically unfeasible (a sample calculation is provided in Section 6.1.3) with current technologies.

Chapter 3

Requirements

The requirements for the FRS3 was that it had to be reliable, not just the functionality, but the physical design to withstand the temperatures of space. The FRS3 was designed as last-case scenario, which assumes that critical components are already in a state of emergency. All of the requirements were in line with Occam's Razor (the principle of least complexity) in order to minimize variables which could reduce the reliability of the component. Below are the other high level technical requirements:

3.1 Adaptability

To increase the impact of the FRS3, the subsystem should be compatible with most kinds of spacecraft, from different clients. As such, it needs to easily interface with many different components. These can be split into two different categories of adaptability: Software flexibility, which enables companies to specify wake-on-wake-off time, keys, and by association, the power consumption, life-time, and responsiveness, and hardware adaptability, which includes how the system integrates with elements such as the power bus, the emergency power time (through battery choices), and mounting possibilities. Given that this is a general application design, the FRS3 can be customized and minimized from it's current debug state into a more appropriate design for each individual client.

3.2 Self Contained

As an emphasis is placed on a system being adaptable for many clients, the FRS3 must be also be system independent. This means that the software running on the FRS3 must be able to run independently of other subsystems on the satellite, and be able to handle it's own exceptions, errors, and on-off operation. In addition, the hardware must also be independent: while a connection to a power bus is required for long term availability, consistent power delivery cannot always be guaranteed. For this reason, the FRS3 must have storage for and be able to handle power delivery from an internal source such as a battery to extend the life-time of the FRS3 in mission-critical scenarios. The independent operation time of the FRS3 should be a minimum of two weeks, or otherwise customizable for the needs of the client.

3.3 Open Source

Given the product's adaptability with various other spacecraft systems and exposure and longevity goals, the product must be entirely open source. Furthermore, all tools should also be open source. This extends from the component integration, custom built footprints, and to driver software that should be available for the any company or any individual to use and customize as required. The decision was made to place all hardware designs and software on a publicly accessible GitHub for the community. Currently, for ease of use, the project it is located under the GitHub account of one of this paper's authors [9]. In the future, it may be moved to the GitHub account of AstroDigital.

3.4 Small Footprint

Given that space constrains on current CubeSats are small, it makes commercial and engineering sense to have every single component be as compact as space-efficient as possible. Since the FRS3 would mark a non-essential component (not mission critical), it has to be such a product have minimal interference with existing spacecraft components. The goal is for the FRS3 to fit within a packed 1U satellite, which would indicate that it could fit within the typical 3U enclosure. There are no specific PCB dimensions given for the general FRS3 application.

3.5 Low Power

Low power usage is one of the most important aspects when it comes to the functionality. Having a device which can operate for extended periods of time opens applications for using a battery to remove power dependency from other components on the spacecraft. Low power usage is an area that will be further elaborated in the Theory of Operations Section 4.3. The target for a low-power component on satellites is around 10mW, a figure given by AstroDigital given their experience in general satellite applications.

Chapter 4

Design

It must be noted that the design process for the FRS3 was very long and intricate. As such, an effort was made to display the broad results and the decisions that were made. The topics presented below are intended to give a broad overview of the research and the construction of the FRS3's major components.

4.1 Frequency

The UHF Band was selected as the primary frequency used. This is because the band, which ranges between 400 - 470MHz is commonly used for satellite telemetry data as well as being a popular amateur radio frequency. This reduces the amount of legal red-tape needed to test the RF Transceivers that have been acquired. Specifically, the decision was made to have the frequency at 433MHz since this frequency band is unlicensed in North America. Elsewhere, this frequency band is primarily used in amateur radio operations.

4.2 Components

4.2.1 RF Transceiver

A critical component of the project would be the choice of transceiver; without one, the system cannot act on signals which it has received. Criteria used to determine eligibility were limited to the following:

- Sensitivity
- Power Consumption (through Standby Current)
- Footprint
- Channel Spacing
- Data Rate
- AES Capability
- Possibility of having an Evaluation board
- Cost per Unit
- Ease of Use

While the list is not in order of importance, it does highlight some of the features that were sought after in receiver design, including low power consumption, having

a small footprint, and having the necessary data rate to handle short bursts of the ground reset signal (GRS). One of the most important features here is the sensitivity, as a low sensitivity receiver dictates a vital section in the link budget, referenced in Section 4.5. Unique to this project is the transceiver's ease of development, which includes factors such as having an evaluation board, being relatively low cost, and being easy to use. The capability of having an AES is a feature that, while not mission critical, is certainly beneficial and reduces overall complexity. Furthermore, any additional features that a transceiver chip has can also be implemented into the FRS3's function, depending on the application. A component which was not considered but certainly an aspect which should be noted was the potential for software challenges, particularly if they depended on updates or patches from the parent company.

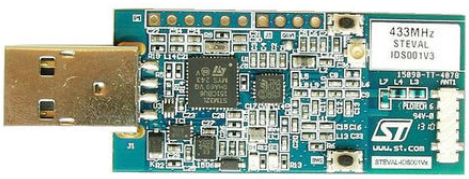
The general approach to finding transceivers was to find unique transceivers from most major microelectronics manufacturers which operated at a 433MHz frequency, and had a small footprint. Options from multiple companies which had similar or identical products where collapsed to form a single representative.

Table 4.1: Transceiver Selection

Transceivers	SPIRIT-1	NRF905	SI4455	AX8052F151
Sensitivity	-118dBm	-100dBm	-116dBm	-116dBm
Standby Current	600nA	32uA	50nA	40nA
Channel Spacing	12.5kHz	100kHz	100kHz	100kHz
Data Rate	1-500kbps	50kbps	500kbps	1-600kbps
AES	128 bit	No	No	No
Eval Board	Yes	Yes	No	Yes
Cost per Unit	$80	$506	$210	$290

The selection process used Table 4.1 to determine which of the receivers would be chosen. Using the matrix decision table, the STMicroelectronics SPIRIT-1, and the OnSemi AX8052151 were selected, the former being the primary component with which to build upon, and the latter being a secondary choice should obstacles be encountered. The main decision factors that influenced this decision was the high sensitivity, a relatively low standby current, coupled within a small factor and a USB-A male port which could directly connect to a PC, contributing to the ease of development. The board itself was photographed after purchase and can be seen below in Figure 4.1.

Figure 4.1: STMicro SPIRIT-1 433MHz module



4.2.2 Micro Controller

Important factors in MCU selection were ease of integration, ease of software development, and reliability. Additional factors include power consumption, footprint size, and memory, but these were deemed less important, as options that excelled in these factors suffered in the previous ones.

Ultimately, the STM32F405RG was the one chosen for the initial version of FRS3. This particular MCU is featured within MicroPython's PyBoard and is known to run it reliably, given its status as MicroPython's flagship hardware. Furthermore, the specifications far surpass any hardware requirements imposed by the software. The schematic of the PyBoard is provided by MicroPython as well, and utilizing this resource simplified the design of the FRS3 schematic.

There are several other boards that MicroPython has been ported to [1], including some that would more closely the fit power and performance specifications more so than the STM32F405RG. However, there are no readily available schematics to refer to, and the MicroPython port may not be as robust as the original port.

4.2.3 Power Delivery

Power delivery is a component of the FRS3 that demands special attention; insufficient power delivery will harm the functionality through potential blackout, brownout, or decreased lifespan through inconsistent power delivery. Through the research design phase, key elements have been identified as being of importance:

- Able to support sustainable battery in orbital parameters
- Able to power from the main power bus during non-critical operation
- Able to use battery to power FRS3 main power bus is offline

It was determined that a component TPS22933 could be used for this exact operation. This component has three inputs; two power inputs, and an enable line. The enable line allows the TPS22933 to choose between which power inputs is used. If the main power bus drops in voltage, than the TPS22933 automatically begins to pull power from the battery.

4.2.4 Antenna

While the antenna is a central part of any RF system such as the FRS3, a particular antenna is not required for a general design. Antenna requirements can vary depending on application, so the general design must be adaptable, whether the antenna is mounted on a surface panel, deployed, or even if it utilizes the main antenna via a switching system. Broadly, antennas matching the profile of this application tend to have an impedance of 50Ω. Hence, for the general application design a small antenna with a 50Ω impedance was chosen. This was a board-mounted 433MHz ISM SMD Antenna from Johanson Technology, part number 0433AT62A0020. The main board utilizes an SMA connector, allowing the board to connect many different antennas via coaxial cable.

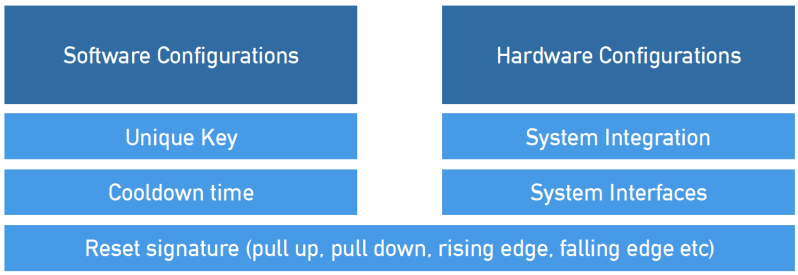
4.3 Theory of Operations

The functionality of the FRS3 can be subdivided into two main areas: pre-launch and post-launch. Pre-Launch concerns itself with regards to the flexibility aspect mentioned in Section 3.1, whereas post-launch can be further subdivided into two main categories to preserve power: Nominal Operation, and Active Operation.

4.3.1 Pre-Launch

Configurations need to be able to be made in two main areas, software and hardware. By being able to partition the two, the client is able to acquire flexibility and compatibility with existing spacecraft systems.

Figure 4.2: Pre-Launch Operation



On the software side, configurations can take the form of the the encryption key, and the message key, as well as the message length, the duty cycle (the effects of these modifications can be seen in the overall power consumption calculations, shown in Section 4.7). These will influence the hardware by changing the size and capacity of the battery required for the minimal two week operational period.

On the hardware side, there are both system integration and system interfaces that need to be made. System integration includes both physical integration, such as having the satellite power bus connected to the power pin on the FRS3, to having the mounting holes in the correct places for that particular satellite, to the correct antenna for that application, with space for that antenna being made. It is also likely that the general application FRS3 deliverable will have to be tailored for each client, meaning certain size and power optimizations can be made to further customize the FRS3 to each application. This also one of the reasons why the hardware design is open-source.

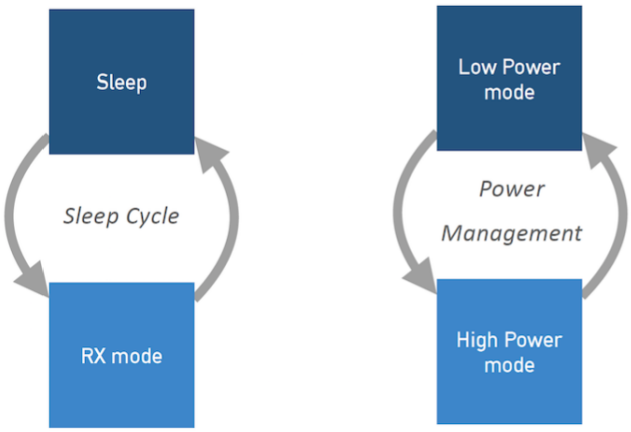
System interfaces is defined as how the FRS3 will communicate the reset signature to the spacecraft. The FRS3's responsibility is to send a pre-programmed reset signal to other components on the satellite which handle a physical reset of the system instead of itself being responsible for resetting components.

The reset signature is the pulse which is emitted by the FRS3 after successful confirmation of the unique key (see Section 4.4 for more details on the reset signature and the ground reset signal).

4.3.2 Nominal Operation

In order to fit within the power consumption constraints outlined in the Low Power Section 3.5, a Nominal Operation mode had to be established. A trade-off had to be made between power consumed in receive (RX) mode and in sleep mode, and the time spent. By having a sleep-cycle in nominal operation, the average power usage decreases while still being able to receive messages. This nominal operation transition is shown in Figure 4.3. By switching state from sleep to RX, the satellite is able to switch between low current and high current consumption modes respectively. The mechanism for this change is detailed in Section 4.3.4.

Figure 4.3: Nominal Operation



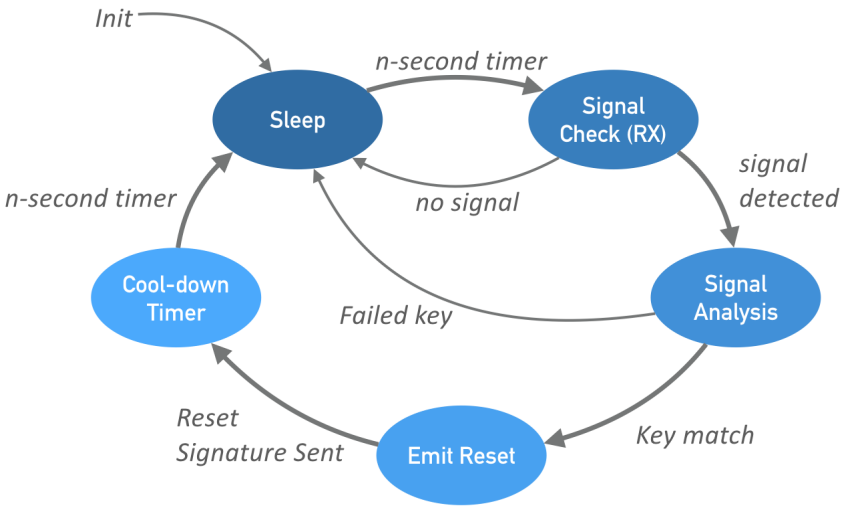
4.3.3 Active Operation

If the signal is detected during the Nominal Operation, the FRS3 goes into Active Operation, in which the state diagram can be seen in Figure 4.4. As discussed in Nominal operation, the FRS3 cycles between sleep and receive (RX) mode continually based on parameters given by the client. Upon a signal detection (a possible Ground Reset Signal), the FRS3 goes into signal analysis, where it uses the AES co-processor on board the SPIRIT-1 to decrypt the package. If the matching key is the correct one, the FRS3 emits the pre-programmed reset signature to the relevant component, and goes into cool-down, where it will ignore all signals until time-out, with which it returns back to nominal operation.

4.3.4 State Switching

In order to be able to change from Nominal Operation to Active Operation requires the FRS3 to be able to make decisions based on whether or not it is able to hear a possible reset signal from Earth. An separate microprocessor was chosen to host the logic necessary to make decisions based on input from the SPIRIT-1 about whether to go into Active Operation once a possible signal was located, and based on analysis of that signal, whether the reset should occur. The software implementation of this state switching can be found in the software section.

Figure 4.4: Active Operation



4.4 Reset Signal

The reset signal is split into four segments:

- Ground Reset Signal (GRS)
- Encryption key (E_{key})
- Matching key (M_{key})
- Reset Signature (RS)

These terms are defined as follows; the ground reset signal is the signal emitted from the ground station, which is the matching key encrypted with the encryption key such that:

$$\begin{aligned} GRS &= AES(E_{key}, M_{key}) \\ M_{keyReceived} &= AES^{-1}(E_{key}, GRS) \end{aligned} \tag{4.1}$$

After the satellite has received the GRS, and decrypts it, it then preforms matching with the Matching key. If $M_{key} = M_{keyReceived}$, then it emits the RS that was specified by the client. This can be as straight-forward as a pull-up/pull-down resistor, or an output pin that is set to 0/1.

In terms of encryption, AES-128 was chosen as the encryption scheme for communications. While this is by no means ideal, it is the limit of the built in co-processor on board the SPIRIT-1. It can be seen in Section 4.2.1 that the SPIRIT-1 is the only transceiver of all the options that were researched with a built in AES co-processor, which enables lower power draw due to the specific hardware accelerated module, rather than attempting to emulate AES through software. The benefits of AES are such that it attempts to reduce the probability of accidental signal lock from an alternate source matching the GRS. This is through a property of AES that a single bit flip in the message will result in a least half of the bits flipped in the final message, which helps to attempt to reduce message collision.

From a security perspective, however this AES does not prevent the case in which a hostile passive observer can record the signal and force unnecessary reset on a target spacecraft. Given that the key does not change, and there is a one-to-one

correlation between the GRS and the RS (minimal disguise on the purpose of the signal), it is not difficult for this property to be abused by hostile actors.

While a better option may be to use an asymmetric key/asymmetric key cryptography, this relies on both components being able to decide on a encryption key to decide secure communication. Since the FRS3 cannot communicate, the resulting option is to rely on a precomputed symmetric key. Making this more challenging is the fact that the GRS must be constantly broadcast with no guarantee of transmission or of receiving. Hence, the key must be kept constant and symmetric. Possible improvements for this are mentioned in the security improvements section in Section 6.1.3.

The exact form of the reset signal has minimal properties; it's maximum length is 3 kilobytes, and the encryption key and matching key have been decided prior to launch.

4.5 RF Link

As mentioned in Section 2.4, a link budget was used to determine the necessary RF sensitivity. This budget can be found in full in the Appendices. However, a condensed version of Table A.1 can be found below in Table 4.2.

Table 4.2: Link Budget Uplink Condensed

Parameter	Value	Units
Ground Station EIRP	31.4	dBW
Isotropic Signal Level at Spacecraft	-128	dBW
Signal Power at Spacecraft LNA Input	-148.0	dBW
Ground Station Receiver Noise Power	-162.4	dBW
Signal to Noise Ratio Power Ratio at S/C Receiver	14.4	dB
System Link Margin	4.8	dB

The EIRP, or the Ground Station Effective Isotropic Radiated Power, is the effective power produced at the ground station. This is the sum of the power at the transmitter, the losses due to the transmission line prior to the antenna, the losses of the connector, filter, and in-line switching losses at the antenna, and finally the antenna gain itself. the Isotropic Signal Level at the Spacecraft is the power in dB that occurs *at* the spacecraft, but is not the same as the amount that is *received* by the spacecraft. The amount that is received at the spacecraft is known as the Signal Power at Spacecraft LNA Input, which is the sum of the Isotropic Signal Level at the spacecraft after accounting for Antenna Pointing Loss, Antenna gain, and again, transmission line loss on board the spacecraft. The Ground Station Receiver Nose Power take into consideration the Effective Noise Temperature of the spacecraft, and the spacecraft Receiver Bandwidth.

The Signal to Noise Power Ratio at the Spacecraft receiver is the difference between the signal power at the spacecraft LNA input and the ground station receiver noise power. Once we have this value, we subtract the required SNR for the telemetry system, which gives the System Link Margin.

This value has very specific meaning: it is defined as the amount of dB attenuation tolerable before signal loss is complete at the spacecraft, and signal acquisition becomes impossible. A margin of 4.8dB is fairly narrow in these applications.

4.6 Modulation

The SPIRIT-1 transceiver is capable of 2-FSK, GFSK, MSK, GMSK, OOK, and ASK. Due to feedback from industry experts at Astro Digital, the GFSK modulation scheme was chosen. Out of all of the options, it is the least susceptible to noise and is the most bandwidth efficient. These qualities are important to the performance of the FRS3.

4.7 Power Consumption

Many of the core requirements set out in Section 3 have to do with reliability and independence. As such, a high priority was made to find power savings which could result in a longer potential lifetime of the FRS3. These can be broken down into two main areas: the PyBoard, and the SPIRIT-1.

4.7.1 PyBoard

The use of the the PyBoard microcontroller to implement state switching and to coordinate the separate modules on the SPIRIT-1 come with increased power consumption for the FRS3 as a whole. Investigations current consumption figures yielded a method of decreasing power consumption by modifying the CPU frequency via the command:

$$\mathrm{pyb.freq(CPU\_FREQ)}$$

Tests run during the PyBoard's development show a significant decrease in current draw by more than 10x when decreasing the CPU frequency and disabling power-hungry I/O ports (such as the microUSB port).

Table 4.3: Current Consumption at varying CPU frequencies [11]

CPU Frequency MHz	Current Consumption mA
168	46
84	26
56	19
42	19
6	3.8

As can be seen from Table 4.3, that 6MHz is clear the best choice. Documentation for the PyBoard in its development phase suggested that the PyBoard may be able to go down to as low as 2MHz, but performance could not be guaranteed. This was the basis for determining the maximum current draw from the PyBoard [11].

The ability to further decrease this current consumption would rely on removing or disabling pull-up/pull-down resistors and reducing parasitic capacitance, all of which can contribute to the reduction of ghost current consumption even when the PyBoard is not preforming critical tasks. In addition, power consumption can also be decreased by putting the PyBoard itself into sleep mode when the SPIRIT-1 is not in listening mode.

Due to the nature of the communication, and the logic that has to be implemented, a CPU frequency of 6MHz was chosen. Initial tests showed that the current consumption at this rate is roughly 3.8mA. A deep sleep mode is enabled by the command:

```
pyb.standby()
```

Power consumption in this mode is roughly 50μA [12], which is high for an microntroller such as this. However with these two figures an average power consumption for the PyBoard can be calculated.

Based on rough software estimates prior to testing, it was determined that a successful acquisition and analysis of a signal could result in the PyBoard being in active mode for up to 10 seconds. Of course, this number could be lowered by increasing the CPU frequency, but this tradeoff was determine to be inefficient. In addition, the Theory of Operations 4.3 shows that the main processing of the signal does not need to occur until signal in the valid frequency and power are detected. Hence, there is no evident interval over which an average current consumption could be constructed. For an upper bound, suppose that the satellite is reset with the FRS3 once a week.

$$\begin{aligned} I_{PyBoard} &= \frac{t_{Active}}{t_{Week}} \times I_{Active|8MHz} + \frac{t_{Week} - t_{Active}}{t_{Week}} \times I_{DeepSleep} \\ &= \frac{10s}{604800s} \times 3.8mA + \frac{604800s - 10s}{604800s} \times 50\mu\text{A} \\ &\approx 50\mu\text{A} \end{aligned} \tag{4.2}$$

There are several assumptions associated with these upper bound calculations:

- Instantaneous change from deep sleep to active requires no current delta
- The satellite is reset once a week
- A reset takes 10 seconds
- Current draw is consistent throughout analysis confirmation
- Current consumption is consistent at all temperatures
- Awaiting for interrupt by the SPYRIT can be can be entirely completed in deep sleep

4.7.2 SPIRIT-1

The SPIRIT-1 also has built in low power consumption abilities. As noted in section 4.2.1, the SPIRIT-1 was chosen for its lower default power consumption, but that can be further enhanced by enabling nominal operation. Observing Nominal Operation (Figure 4.3) together with the SPIRIT-1 Current consumption in different states (Table 4.4) it can be seen that RX mode consumes orders of magnitude current in

Table 4.4: Current Consumption of Different States of the SPIRIT-1

State	Current Consumption
SHUTDOWN	0
READY	400μA
LOCK	$4.4m$A
RX	$9.2m$A
SLEEP	$850n$A

RX mode than in sleep. Hence by switching between SLEEP and RX states, efficient power management can be achieved.

In order to calculate an upper bound on the power consumption of the SPIRIT-1, one of the key values is to know how long the SPIRIT-1 will be in RX mode. This of course is also an area customizable by the customer as required, since it will influence power consumption. Unlike the PyBoard, which can wait in a deep sleep mode until interrupted, the SPIRIT-1 must constantly shift between RX and SLEEP states in order to observe signals from the home station. When looking at the satellite orbits, it is clear that duration within transmission radius of the home station is not infinite; that is, while the satellite will be able to orbit the planet on an almost hourly schedule, it will only be in receive radius of the home station for minutes. Hence, within this time limit, a guarantee needs to be made that the FRS3 will be listening for at least one cycle. If desired by the client, even this figure can be reduced since the FRS3 has an independent operational period of 2 weeks, there should be ample opportunity for a reset to occur. However, for reliability concerns, this was not entertained.

To provide the greatest reliability, the GRS will need to be constantly broadcast towards the damaged satellite. Given the quick broadcast time per signal, it is projected that the time spent in RX needs to account for the time required to "lock on" to the signal. Signal Analysis, however does not need to be in RX mode as the microcontroller's duty is to record the GRS and to use the AES co-processor (an individual module on board the SPIRIT-1). The AES co-processor can be active in any SPIRIT-1 operating state. Is is assumed that in this state the co-process consumes minimal current consumption relative to RX mode. Therefore the maximum current overhead in this case is only the time taken in RX. The time taken in RX mode is a function of the length of the message, the number of messages before a successful signal is acquired, and the rate at which the signal is being sent, such that:

$$t_{RX} = M_{length} \times N_{messages} \times D_{rate} \tag{4.3}$$

Where an upper bound on the length of a message would be 3 kilobytes, the average number of messages would be 3, and the data rate would be approximately 1kbps. Using these figures, we can determine the average power consumption based

on power consumption per minute.

$$\begin{aligned} t_{RX} &= 3[kb/msg] \times 3msg \times 1kbps \\ &= 9s \\ I_{SPIRIT-1} &= \frac{t_{RX}}{t_{minute}} \times I_{RX} + \frac{t_{minute} - t_{RX}}{t_{minute}} \times I_{SLEEP} \\ &= \frac{9s}{60s} \times 9.2mA + \frac{60s - 9s}{60s} \times 850nA \\ &= 1.4mA \end{aligned} \tag{4.4}$$

The ratio of time spent in active mode (t_{active}, given as t_{RX}) to the total time (t_{total}, given as t_{minute}) is known as the LDCR. A value of 15% for the Load Duty Cycle Reload is low for these applications. However, it is an area which can be configured by the client.

$$\begin{aligned} LCDR &= \frac{t_{active}}{t_{total}} = \frac{t_{RX}}{t_{sleep}} \\ &= \frac{9s}{60s} = 0.15 \end{aligned} \tag{4.5}$$

In calculating the SPIRIT-1 current consumption, there are some assumptions that which simplified the calculation of the total SPIRIT-1 current consumption:

- Instantaneous change from RX to SLEEP requires no current delta
- RX can receive at D_{rate} without bit loss
- RX can 'lock on' to the uplink frequency instantaneously
- RX can 'lock on' to the GRS content within one message of M_{length} length
- Decryption can occur in nominal operation with minimal current draw
- Activation and deactivation of AES co-processor consumes minimal current
- Passing the decrypted transmission to the microprocessor consumes minimal current

4.7.3 Total Power Consumption

Even upper bound estimates cannot replicate the actual power usage, which can only be measured by having a physical component and measuring the current usage in real-life scenarios - rather difficult when real life scenario are in difficult to reach locations. However, by using the equations given in Equation 4.2 and Equation 4.4, we can attempt determine an upper bound on the total current consumption:

$$\begin{aligned} I_{total} &= I_{PyBoard} + I_{SPIRIT-1} \\ &= 50\mu\text{A} + 1.4\text{mA} \\ &\approx 1.4mA \end{aligned} \tag{4.6}$$

Given an upper current consumption, a upper limit power consumption can be calculated as well. Although the FRS3 has an operating voltage of 3.3V, there is no guarantee that the input voltage will be exactly 3.3V as well, since that is determined by the battery that will be used. Hence, a battery voltage of 4V was estimated for this calculation.

$$\begin{aligned} P_{total} &= I_{total} \times V_{battery} \\ &= 1.4mA \times 4V \\ &= 5.6mW \end{aligned} \tag{4.7}$$

It can be seen that a upper limit on power usage compares favourably to the upper limit given in the requirements of 10mW given in Section 3.5.

Knowing the current consumption also confirms the capacity of the battery required for two weeks of nominal and active operation.

$$\begin{aligned} C_{battery} &= I_{supply} \times t_{twoWeeks} \\ &= 1.4mA \times 2 \times 7 \times 24h \\ &= 470mAh \end{aligned} \tag{4.8}$$

From Equation 4.8, it can be see that a potential battery used directly for the general application FRS3 would have to have a capacity of 470mAh, which is high for a coin-cell or lithium cell. This gives much room for improvement. It can be seen that the biggest contributor to the overall current consumption is the LDCR. A shorter message may be preferable depending on client needs, which can in turn reduce SPIRIT-1 average current consumption and hence decrease total power consumption.

Chapter 5

Results and Tests

5.1 Hardware

In choosing the hardware, a constant requirement was that every component had to have tolerance to space conditions, generally given as -40C to +85C. The hardware process was to build a testable prototype that could also be used for software testing and validation purposes. Once this was complete, a general application hardware design could be made.

5.1.1 Prototype

The process began with the purchase of the SPIRIT-1, which was tested to determine whether it fit the specification and the criteria that were used to choose it, particularly with respect to the ease of development and potential software challenges. It appeared that this was an important concern as in the early phases of the project, using the drivers on board the SPIRIT-1 faced challenges because of a required driver update from STMicrocontroller, which did not occur until late into Q2 2016.

Once the SPIRIT-1 was properly updated and the drivers were operational, it was determined to be the selected RF Transceiver, and a miniaturized SPIRIT-1 was created by using the block diagram supplied by STMicrocontroller. This featured not insignificant changes such as using a SMA connector to interface with an antenna system and replacing the default standard USB-A male connector with I/O pins. This intermediary step was called the SPYRIT, a SPIRIT-1 that was compatible with a PyBoard. Two of these SPYRIT's were manufactured for testing, one of which can be seen in Figure 5.1. Due to the symmetric nature of the PyBoard, the SPYRIT can be placed on either edge of the board with the antenna pointing away from the microprocessor.

5.1.2 Discrete Components

Table 5.1 contains a list of each of the components that were used in the process of PCB layout.

Figure 5.1: SPYRIT, PyBoard and the Antenna with SMA Connector

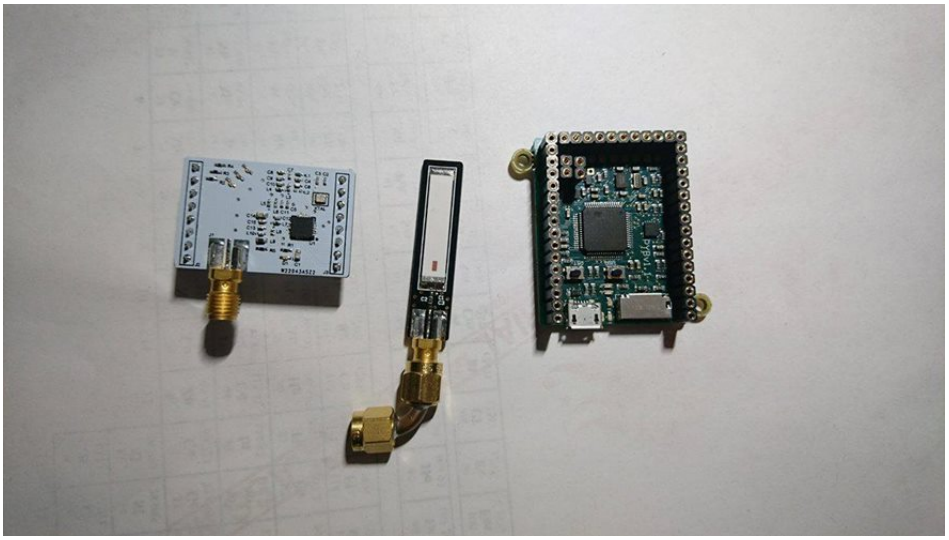


Table 5.1: Discrete Component Choices

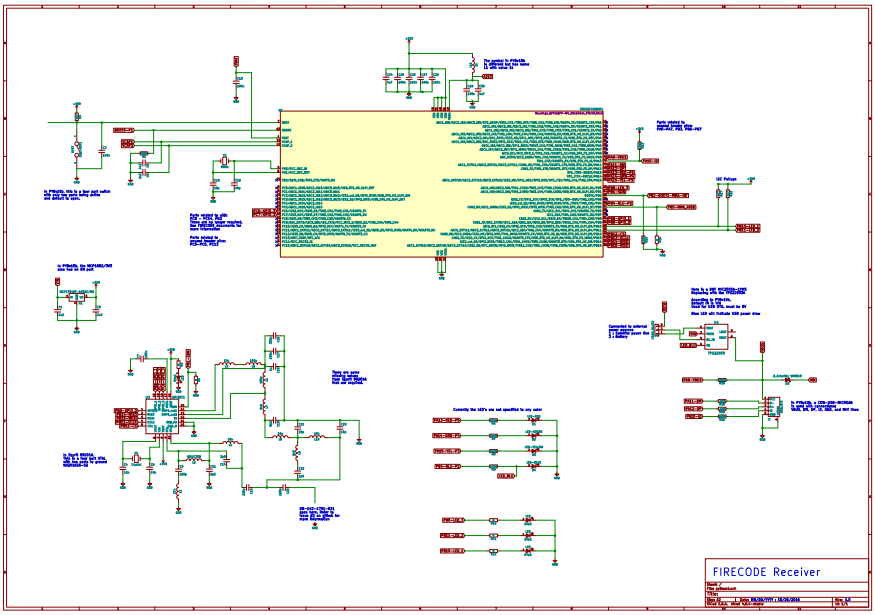
Component	Package Size	Product Name
Resistors	0603	
Capacitors	0603	
Inductors	1206	
Oscillators	SMD 5032	
Button Switches	SMT	Omron SPST 3BU 3000P-B
Voltage Regulator	SOT89-3	MCP1754ST-1802E/MD
RF Transceiver	QFN 4x4mm	ST SPIRIT-1
Power Multiplexer	μQFN 1.5x1.5mm	TPS22933
Microcontroller	LFP64	STM32F405RGT6
Shottky Diode	DO-41	1N5819

5.1.3 Schematic

Having knowledge that the PyBoard and the SPYRIT work together, the next step was to begin miniaturization of the two components. Since the two products were developed on different software (The SPYRIT was developed on a professional grade software known as Altium, whereas PyBoard, an open source project was developed in a paid program EagleCAD). In order to keep in sync with the original requirement of using only open source software, both of these products had to be manually ported into KiCAD, a time-intensive process. Having both of the components on one schematic allowed an optimization through a reduction of unnecessary components on the PyBoard, and direct wiring of the components on the PyBoard to the components on the SPYRIT rather than using external pin connectors. The detailed schematic for this can be found in Figure A.2, and briefly, in Figure 5.2.

However, it can also be seen that there are still components who's purpose in a final launch product is unnecessary; these include components such as debug/notification LED's, microUSB headers, and switches for manual reset operation. These form the basis for a general application deliverable.

Figure 5.2: schematic



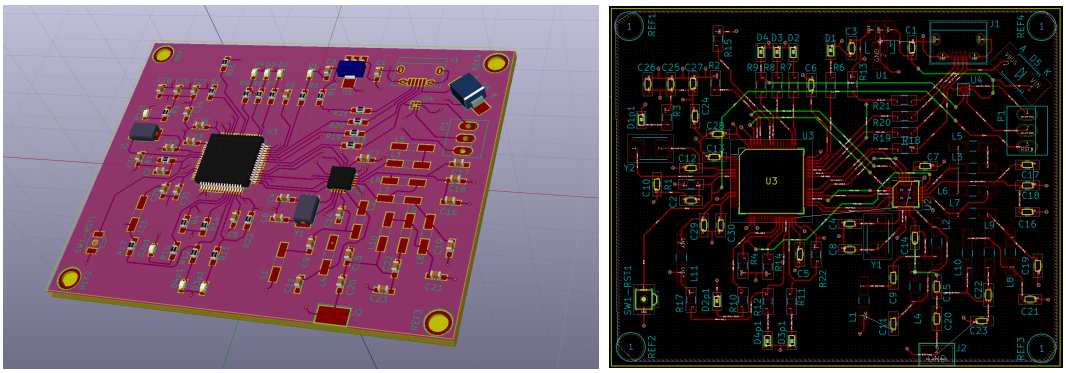
5.1.4 PCB Layout

Laying out the PCB was much more complex than anticipated. Since the general application product had have more features as compared to a custom mission-ready FRS3, there were some unexpected learning curves and challenges.

The board itself is a four layer board, laid in the default format of Signal, 3V3, GND, Signal, from top to bottom. This decision was made so that RF signals could be isolated at the bottom of the PCB rather than possibly influencing signals on the top of the PCB. This was also an inevitable requirement due to the way that the components were routed. It is possible that better component placement would result in a reduction in the number of layers, but the inclusion of RF isolation forces a four layer construction. Dimensions of this board were not explicitly stated in the original requirements, but were made to be as compact as possible. Since the shippable design was made to be closer to a breakout board rather than a fully optimized product, some leeway was given to the PCB layout.

Component selection and footprint challenges were a minor aspect of the layout. Components were chosen to be an appropriate balance of miniaturization and ease of use. Some components, such as resistors and capacitors, were of size 0603, which is smaller than the default 0805, but still are relatively easy to be hand soldered during component manufacturing and testing. Additional components, such as button switches, and footprints for the SPRIT-1 microprocessor and the PyBoard microprocessor had to be found using the public KiCAD libraries on GitHub. Certain components, such as the μQFN 1.5mm x 1.5mm for the power switching module TPS22933, simply did not exist in the community open-sourced KiCAD libraries. This component specifically had to be custom built for this project. Additionally other features for the general pupose application had to be added, such as mounting holes, debugging points, microUSB connector, switches, etc.

More detailed diagrams can be found in the appendices under Figure A.4 and Figure A.3. The current PCB layout and size suggest a footprint of approximately



(a) 3D rendering PCB Layout (b) 2D PCB Layout

Figure 5.3: PCB Layout in 2D and 3D based on renderings

$54mm \times 70mm = 3,780mm^2$, which is slightly less to the summation of the SPYRIT and the PyBoard, (which have a footprint of $22.5mm \times 30mm = 675mm^2$ and $33mm \times 40mm = 1,320mm^2$ respectively) and is relatively comparable considering it not only includes components of a much large size, but also includes debug points, microUSB ports, and hand-solderable components.

5.2 Software

The software design of the project constitutes of two main parts. First is the SPIRIT-1 drivers, which allows a MicroPython controller to use an RF transceiver. Second is the embedded system code that implements the drivers in a way that implements the desired behavior of FRS3.

5.2.1 SPIRIT-1 Drivers

The drivers for the SPIRIT-1 transceiver are provided for microcontrollers running in C [10]. However, all embedded systems originating from Astro Digital are developed using MicroPython. As a result, selected functions and features were ported from C into Python. These drivers were tested and developed using the SPYRIT prototype board.

All features of the SPIRIT-1 are controlled by registers. There are read/write, read/erase, and read-only registers. They are manipulated via SPI (serial peripheral interface). The two main functions used in SPI are read and write to a specific register address or in a sequentially increasing address burst mode. Using these two functions, every other functionality can be configured by interpreting the input and writing the corresponding registers or parsing the register values and returning the appropriate result.

The initial version of the drivers involved writing functions for all of the SPIRIT-1 features in MicroPython. However, this proved to be problematic. Many of the functions performed calculations that occupied all of the memory of the microcontroller, causing it to crash. Additionally, the microcontroller clock speed was decreased to limit power consumption, so the operations tht did work took too much time to complete. The solution devised was to transfer all of these functions to a GUI in

CPython (Python used for normal computers), where memory limitations would not be a factor. The preferred configurations are then be calculated there and output to an XML file, which are then be loaded onto the microcontroller. This XML file then acts as a non-volatile configuration of the registers states. The microcontroller will then write all the register values listed on the table upon initialization. This GUI is named "SPIRIT-1 App". Besides manipulating the SPIRIT-1 register table in a user-friendly way, the SPIRIT-1 App can also send commands to the SPIRIT-1. This allows the SPIRIT-1 to transmit and receive packets in a live testing environment. However, this function is not critical and will be completed at a later date.

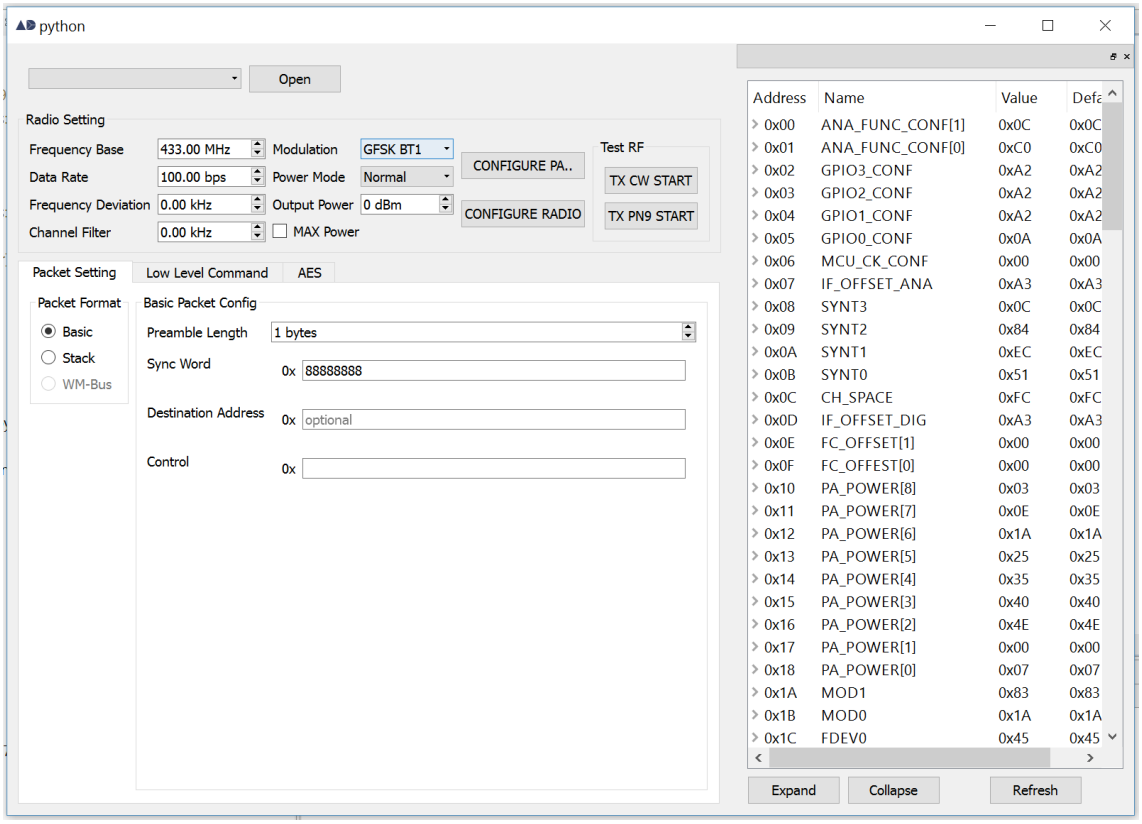


Figure 5.4: Screenshot of the SPIRIT-1 App

5.2.2 Embedded Software

The section explains the software design decisions made to achieve the behavior outlined in Section 4.3.

The behavior of a nominal and active states is present in the LDCR (low duty cycle reload) feature of the SPIRIT-1. In LDCR, the SPIRIT-1 alternates between the RX state and the SLEEP state based on a wakeup timer and RX timer configured in the registers [13]. However, the RX state can be sustained if a parameter or combination thereof are fulfilled. The RSSI (radio signal strength indicator) level can be higher than a configured threshold, the SYNC word of the packet can be detected, or the radio signal quality indicator can be above another threshold. An

and/or relationship between these paramters can be configured in the registers, and if the correct combination of them are fulfilled, the SPIRIT-1 will delay the RX timeout to continue listening for a complete packet.

The SPIRIT-1 has the capability to provide an interrupt request to the MCU. This feature is critical for all the state transitions outside of the LDCR. During LDCR, the MCU is in a low-power state, waiting for an interrupt. As soon as an interrupt request is received, it wakes up and reads the interrupt request registers on the SPIRIT-1. The interrupt registers indicate which interrupt has occurred. From there, the MCU can take appropriate action. The first interrupt is the RX Data ready interrupt, which occurs when the SPIRIT-1 has received a complete packet. When received, the MCU reads the RX output queue on the SPIRIT-1, where it then writes it to the AES IN registers. The contents are then decrypted by the SPIRIT-1's AES co-processor. When the process is complete, the SPIRIT-1 then sends another interrupt request, indicating the end of the AES operation. After this, the MCU reads the AES OUT registers. If the decrypted contents match the reset code, then the MCU sends the reset signal, and then goes back to sleep.

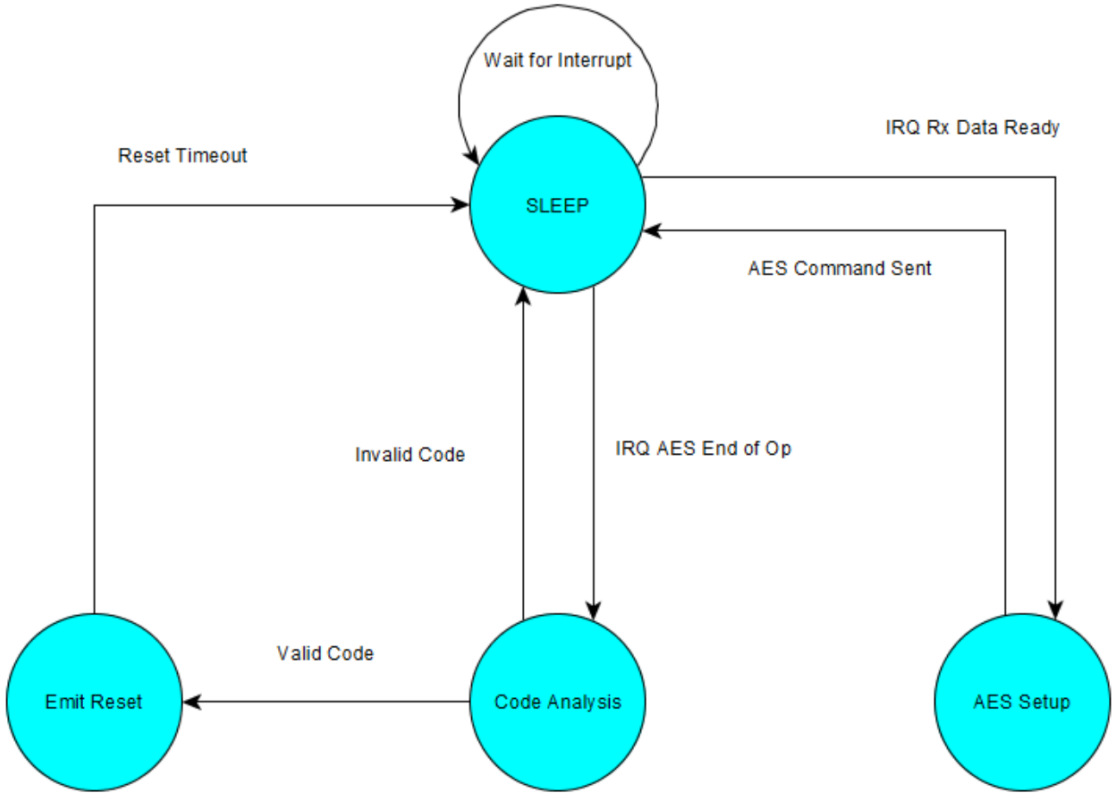


Figure 5.5: FSM for MCU operation

Chapter 6

Next Steps

6.1 Design Improvements

Despite the deliverable being being a general application design, there are still directions in which the design can improve in. The sections below are by no means complete; for there are an large number of ways to continually improve an embedded design project such as this. These are those directions in which the authors feel are the most natural and obvious solutions.

6.1.1 Hardware

The primary goal of hardware is to enable the software to preform as smoothly as possible while minimizing power and space constraints while preventing unintended operation.

The miniaturization of the overall footprint is possible through miniaturization of the basic components, if the requirement to build the component by hand is unnecessary. Capacitors, Inductors, and Resistors all can go smaller than the 0603 reference size, i.e. 0402 or even 0301. A consequence of this is lower power draw as smaller components require less power. In addition, other components such as the LDO Regulator were chosen to be an accessible footprint (SOT) instead of a smaller form factor such as DFN, which is smaller but KiCAD footprints for are less common. Given more time, each component could be made find the smallest and lowest power equivalent in an optimization phase.

The SMA connector (while the footprint is on on Figure 5.3a, a good example of would be the one connected to the prototype in Figure 5.1) in the general application FRS3 is included so that a change of antenna is possible depending on what the client prefers, whether it be a more powerful omni-directional antenna, or perhaps a custom antenna designed for that particular satellite. But in the case that the client wants only the smallest package possible, the antenna portion can be incorporated onto the PCB itself, thus decreasing overall footprint.

An additional component that can be added is the addition of a DMR/TMR at the output pin would prevent a reverse Lightsail-1 type of error (see Chapter 1 for details) where a photon strikes a control pin and accidentally resets the spacecraft that does not need to be reset. It is important to note that the signal emitted by the FRS3 does not physically reset hardware components, rather it sets a reset control line to high/low depending on the client needs. If it were to be directly connected to

hardware components, the output pin would also require impedance matching and the components necessary to drive all reset pins to reset mode. This was beyond the scope of the project and was not considered in the design phase.

The microprocessor, while chosen for its guaranteed functionality through it's use in the PyBoard manufacturing and marketplace usage, is not the most efficient of components. Indeed, it can be seen from the PyBoard's deep sleep mode that the deep sleep power consumption is $50\mu A$, much higher than the SPIRIT-1's deep sleep mode of 850nA. However, there is a lower bound on the type of microprocessors that can be used because they will still have to be able to run MicroPython.

6.1.2 Software

Embedded software often runs on real-time computing restraints. At the time of writing, these restrains are not yet defined in the requirements of the project, and the actual response time of state transitions within the software has not been tested. It would be helpful to potential clients to know these specifications to ensure that the FRS3 would be appropriate for their systems.

The FRS3 is capable of performing other tasks other than a satellite-wide reset. After the received message is decoded and it is verified to be the right recipient, the message can contain other commands that the MCU can interpret and perform. Examples could include resetting specific systems upon the spacecraft and enabling or disabling other subsystems.

To make the FRS3 resistant to signal anomalies in orbit, it can contain a multiple layers of signal verification to ensure that the correct command is sent. For example, the reset message can be sent in multiple parts, and for each part received the MCU would assert one output, and if all outputs are asserted, the reset signal would propagate.

6.1.3 Security

As mentioned in Section 4.4, there is nothing preventing a hostile actor from recording the GRS and playing it back to reset the spacecraft. Meaning that there is no need to decrypt the GRS at all. This problem can by solved by having a changing symmetric key with every reset signal. A proposed solution to this is to have the initial key being used to encrypt the initial reset signal, which contains two sections: The reset message, and the encryption key for the *next message* that is used. Hence, the message content can be observed as such:

$$GRS_{new} = AES(E^{i}_{key}, (M_{keySent}, E^{i+1}_{key})) \tag{6.1}$$

Where E^{i}_{key} is the initial key and E^{i+1}_{key} is the next key. This does require further computation from the ground station, since once communication with the satellite is established, both the client and the FRS3 need to change the encryption key to use the new implicitly agreed upon encryption key. This is not an established methodology as of yet. The benefit of this methodology is that security from a hostile actor is dependent on the strength of the AES key, which in itself has 3.4×10^{38} possible combinations, requiring the fastest supercomputer (as of Dec 2nd, 2016) the Sunway TianhuLight, with a 93 pentaflop computing limit, an optimistic 1000

flops (floating point operations) [14] per combination check, and only requiring to search though half the keys on average to obtain the correct key:

$$\begin{aligned} C_{checks/s} &= \frac{93 \times 10^{15}}{1000} \frac{[flops/s]}{[flops/check]} \\ &= 93 \times 10^{12} [checks/s] \\ t_{s/yr} &= 60 \times 60 \times 24 \times 365 = 31,536,000 s/yr \\ t_{years|AES128} &= \frac{1.7 \times 10^{38} [checks]}{93 \times 10^{12} [checks/s] \times 31,536,000 [s/yr]} \\ &= 5.79 \times 10^{16} yr \end{aligned} \tag{6.2}$$

For a single decryption of a single key, it would take the TianhuLight 5.79 *quintillion* years of constant computation. Since the key is always changing; it is technically infeasible to decipher the key to reset the spacecraft after the key has been changed. These AES keys would act as ephemeral keys, where they are only permanent until communication has been made. In addition, this paves the way for an expansion of secure communication though the FRS3 if required.

6.2 Application Directions

6.2.1 SPYRIT

The SPYRIT RF skin that was developed to work in conjunction with the pyboard is the first of it's kind, and can be made into an independent project. For this, some more work needs to be completed. To be made truly independent, two separate projects can be made - one where the skin should house it's own antenna (a basic one will do), with an impedance matching circuit; and another which has the SMA connector with ranges for impedance that it will support. Both of these would also need to come with a complete set of documentation, including antenna and SPYRIT radio performance, and most importantly, a completed port of the SPIRIT-1 drivers into MicroPython. The full-range MicroPython drivers still have yet to be tested, which may require further hardware iterations.

6.2.2 AstroDigital

In the future, it would be a great project to be able to take the general application design to produce a final space-worthy board for the company. This would involve a few more hardware iterations, a practical power consumption test, and more day-in-the-life testing, as well as choosing an appropriate omni-directional antenna for AstroDigital's LEO satellite. As mentioned in the Hardware Improvements Section 6.1.1, smaller components can be used, a more efficient microprocessor, and better space efficiency can all be achieved with further testing.

Chapter 7

External Impact

7.1 Use of Non-Renewable Resources

Given the constraint of low-power, low-footprint design, use of any materials is kept at a minimum. The FRS3 will not be mass-produced, and will be manufactured on a case by case basis. Given its small footprint, the impact of its construction on the environment will be marginal. The majority of material used would be in the PCB. Thus, the makeup would be mostly resin, trace amounts of copper, plastic and silicon for the ICs. Aluminum would likely be used for the antenna.

As for the renewable aspect of the aforementioned resources, some of have renewable alternatives. Renewable bio-resin plastic for PCB use does exist [3], but it is not yet mainstream within the industry. However, for aerospace applications, it is necessary to use the most robust option available, so conventional PCB material is to be used. The copper used for the PCB connections will be non-renewable as copper is finite resource, and in this application it will also not be recyclable. Any ICs, such as the transceiver chip, are made of plastic, metal, and silicon components. These items can not be considered renewable either. The battery will likely be composed of a lithium ion polymer and is not recyclable or renewable.

The powering of the receiver itself can be considered renewable. Satellites are primarily powered by solar cells, and the receiver will receive power from the spacecraft's power bus.

7.2 Environmental Benefits

When spacecraft "go dark", or become unresponsive while in orbit, they lose valuable time when they should be performing their mission. If the satellite has a terminal issue, then the spacecraft becomes space debris; a waste of all costs associated with the design, integration, and launch of the satellite. The FRS3 attempts to minimize such situations by recovering spacecraft when they suffer from issues that could be solved by a hardware reset. As a result, there would be minimized scenarios where replacement satellite would be needed, minimizing costs and materials used in the development of an additional spacecraft. Many spacecraft components are non-renewable, so launching fewer of them can be considered to be beneficial to the environment.

Contrary to common belief, space itself, or more accurately Earth orbit, can be considered an environment that is susceptible to pollution. When spacecraft

complete their mission or become terminally inoperable, the physical satellite stays in space. Since there is yet to be a reliable mechanism to bring them down, the satellite can remain in orbit for extremely long periods of time. These satellite "corpses" can accumulate into a space "graveyard" in a popular orbit around Earth, and will eventually pose a risk to other satellites that are still operating, and possibly other space missions. If and when collisions occur, the space debris expands and the space environment becomes a more hazardous and toxic environment.

The FCC requires CubeSats and other small satellites to re-enter the atmosphere 25 years after then end of their mission, otherwise satellites at the higher end of orbits can spend up to 150 years there [8]. However, CubeSats commonly operate within an altitude range where they de-orbit naturally over several years, with the average LEO satellite de-orbiting in around five years [5]. There are technologies being developed to accelerate a de-orbit of a CubeSat at the end of its mission. A university in Finland has developed a deployable plasma brake to accomplish this goal [6]. There are a number of other experimental technologies such as deployable balloons and tethers, and new method are being constantly developed. A natural progression of the FRS3 project would be to also develop some kind of standard de-orbiting device to fulfill the FCC's requirements. This way, in addition to having a capable reset system, it would also fulfill any de-orbiting requirements, further increasing its usefulness. Also, other larger satellites that are placed into higher orbits such as MEO or GEO are now required to contain a certain amount of propulsion to place the spacecraft into a "graveyard orbit," or an orbit where it would pose no threats to collisions or to interruptions to active satellites. This is the solution when propulsion requires are too high to completely de-orbit the spacecraft.

7.3 Safety

Much of the danger or risk involved in a component such as FRS3 is related to its integration. The component itself will be tested thoroughly to ensure proper operation, but improper application could create monetary or health risk to parties involved. Firstly, the integration of electronic components into space or flight hardware requires an extensive ruggedization process. One notable difference between flight electronics and normal electronic components is the use of different kinds of solder. Unleaded solder is commonly used for everyday purposes, but it is prone to "whiskering" [7]. The brittleness in unleaded solder can lead to broken connections, as hardware is treated violently during the rocket launch. Leaded solder, conversely, is more malleable and is less prone to breaking. However, applying leaded solder can be harmful if done in an improper environment. Additionally, here are many other industry procedures that must be followed for safe use of flight electronics that are not covered in this document.

There are many pre-launch configurations that must be determined for the specific application, as mentioned in Section 4.3.1. If these configurations are not correct, it could cause a variety of issues post-launch.

7.4 Benefits to Society

For the satellite aerospace industry to progress, experimental technologies must be flown to test their usefulness and reliability. If satellite manufacturers were more willing to fly these technologies, the industry would progress at a faster rate. It is hopeful that the FRS3 project will urge more companies to take the risk and use these prototyped technologies, knowing that FRS3 would mitigate some of the risk. With the progression of satellite technology, society would benefit from all the missions undertaken, whether it be new remote sensing data, telecommunication, or even exploration.

Chapter 8

Report on Teamwork

8.1 Markus

Markus's responsibilities began with the creation and maintenance of all technical design diagrams involved in this project. His efforts went into the software development of the drivers and MicroPython required in order to make the FRS3 able to communicate with a base station. All things software-side were under his responsibility for this project. Also, given his past affiliation with Aquila Space, was responsible for the collaboration efforts and resource acquisition from Aquila Space.

8.2 Derek

Derek's responsibilities began with research investigation into the RF transceiver. Culminating into significant effort into the design presentation made at the end of the first semester. In the second semester, he was responsible for the hardware-level design and implementation of the FRS3. Using KiCAD from block diagram, to combining hardware prototypes, to designing custom footprints, to laying out the PCB was his section of this project. He was also responsible for all cryptographic material related to the project.

8.3 Overall

As can be seen over the course of this report, there was a steep learning curve associated with the transfer of the knowledge base regarding aerospace requirements, RF components, and the general design process. Aerospace hardware design has much stricter requirements than many other industries and much research was necessary to ensure compliance. RF communication was a challenge as well, as neither Derek or Markus had any prior experience in the field. Many hours were dedicated to understanding the multiple facets of this topic. In addition, the engineering design project requirements laid out by McGill University required the design of posters, presentations and this report, both of which were team efforts by Derek and Markus.

Chapter 9

Conclusion

A complete concept design of the FRS3 has been completed as a result of this project. Due to time constraints, a working prototype of the final board was not completed. However, an initial version of the hardware design is complete, and fabrication of the PCB is imminent. The software design is complete, and only lacks testing on the final board. The hardware design is complete, but requires manufacturing and physical board testing in order to determine next steps as laid out in next steps section of this document.

Working on the project has provided a greater appreciation for readable and complete documentation. The primary components in the project have a large number features, and reading the data sheets and related documents make integration and design involving the components much easier. For example, the SPIRIT-1 data sheet briefly explains how many of the features work, but to implement them correctly in the drivers, much more research and testing had to be done. There is a steep learning curve involved in working with the components, especially among those related to the RF systems.

Since this project only constituted completing the concept design, more work will go into properly integrating and testing it within a proper spacecraft. This involves fabricating the hardware up to aerospace standards and testing in a large range of RF environments. Also, there is a potential to add features other than a reset capability. After decrypting the packet, it can provide other commands that the microcontroller can interpret and perform. One such function could be resetting individual systems, or activating deployment of main antennas or other systems.

Chapter 10

References

Bibliography

[1] "MicroPython - Python for Microcontrollers", *MicroPython.org*. N.p. 2016. Web.

[2] J. Davis, "Software Glitch Pauses Lightsail Mission", *Planetary.org*, 2015. [Online]. Available: `http://www.planetary.org/blogs/jason-davis/2015/20150526-software-glitch-pauses-ls-test.html`

[3] A. Sunija, "A Survey on Bio-Based Materials for Printed Circuit Boards", *International Journal of Science Technology Engineering*, vol. 2, no. 5, pp. 178-180, 2015.

[4] STMicro, Low data rate, low power sub-1GHz transceiver (SPIRIT-1), DocID022758(SPIRIT-1) datasheet, Feb. 2012 [Revised Dec. 2015]

[5] D. Oltrogge and K. Leveque, "An Evaluation of CubeSat Orbital Decay", in *Small Satellite Conference*, Logan, UT, 2011.

[6] O. Khursid et al., "Accommodating the plasma brake experiment on-board the Aalto-1 satellite", Aalto University, Espoo, Finland, 2014.

[7] D. Kostic, "Lead-free Electronics Reliability - An Update", GEOINT Development Office, 2011. [Online]. Available: `http://nepp.nasa.gov/whisker/reference/tech_papers/2011-kostic-pb-free.pdf`

[8] B. Hardy and J. Fuller, "Systems and methods for a self-deploying vehicle drag device," U.S. Patent 8 616 496, August 3 2011.

[9] M. Suorsa, "markusc90/spyrit," in Github, 2016. [Online]. Available: `https://github.com/markusc90/spyrit`. Accessed: Dec. 05, 2016

[10] I. Dmitrichenko, "errordeveloper/mist," in GitHub, 2016. [Online]. Available: `https://github.com/errordeveloper/mist/tree/master/platform/mist-SPIRIT-1/dev/SPIRIT-1`.

[11] D. George, "Power Consumption of the board," in Kickstarter, 2013. [Online]. Available: `https://www.kickstarter.com/projects/214379695/micro-python-python-for-microcontrollers/posts/685745`. Accessed: Oct. 13, 2016.

[12] D. George, "functions related to the board," in MicroPython Docs, 2016. [Online]. Available: `https://docs.MicroPython.org/en/latest/pyboard/library/pyb.html`. Accessed: Dec. 04, 2016.

[13] STMicroelectronics, "Low duty cycle operation with the SPIRIT-1 transceiver," STMicroelectronics, Geneva, Switzerland, 2015. [Online]. Available: `http://www.st.com/content/ccc/resource/technical/document/application_note/47/50/91/3a/f5/fd/4d/c1/DM00068699.pdf/files/DM00068699.pdf/jcr:content/translations/en.DM00068699.pdf`. Accessed: Nov. 14, 2016.

[14] M. Arora, "How secure is AES against brute force attacks?," in EE Times, 2012. [Online]. Available: `http://www.eetimes.com/document.asp?doc_id=1279619`. Accessed: Dec. 01, 2016.

Appendix A

Appendix

Table A.1: Link Budget Uplink

Parameter	Value	Units
Ground Station		
Transmitter Power Output	100.0	W
Transmitter Power Output	20.0	dBW
Transmitter Power Output	50.0	dBm
Transmission Line Losses	-2.6	dB
Connector, Filter, or In-Line switch losses	-1.0	dBm
Antenna Gain	16.0	dBiC
Ground Station Effective Isotropic Radiated Power	31.4	dBW
Uplink Path		
Ground Station Antenna Pointing Loss	-1.0	dB
Antenna Polarization Losses	-4.0	dB
Path Loss	-148.9	dB
Atmospheric Losses	-3.0	dB
Ionospheric Losses	-2.5	dB
Rain Losses	0.0	dB
Isotropic Signal Level at Spacecraft	-128	dBW
SNR Computation		
Spacecraft Antenna Pointing Loss	-3.0	dB
Spacecraft Antenna Gain	-15.0	dBiL
Spacecraft Transmission Line Losses	-2.0	dB
Spacecraft LNA Noise Temperature	150	K
Spacecraft Transmission Line Temperature	270	K
Spacecraft Sky Temperature	700	K
Spacecraft Transmission Line Coefficient	0.6310	
Spacecraft Effective Noise Temperature	691	K
Spacecraft Figure of Merit (G/T)	-45.4	dB/K
Signal Power at Spacecraft LNA Input	-148.0	dBW
Spacecraft Receiver Bandwidth	6000	Hz
G.S Receiver Noise Power (Pn = kTB)	-162.4	dBW
Signal to Noise Power Ratio at S/C Receiver	14.4	dB
System Desired Data Rate	1024	bps
System Desired Data Rate	30.1	dBHz
Telemetry System Required SNR	9.6	dB
System Link Margin	4.8	dB

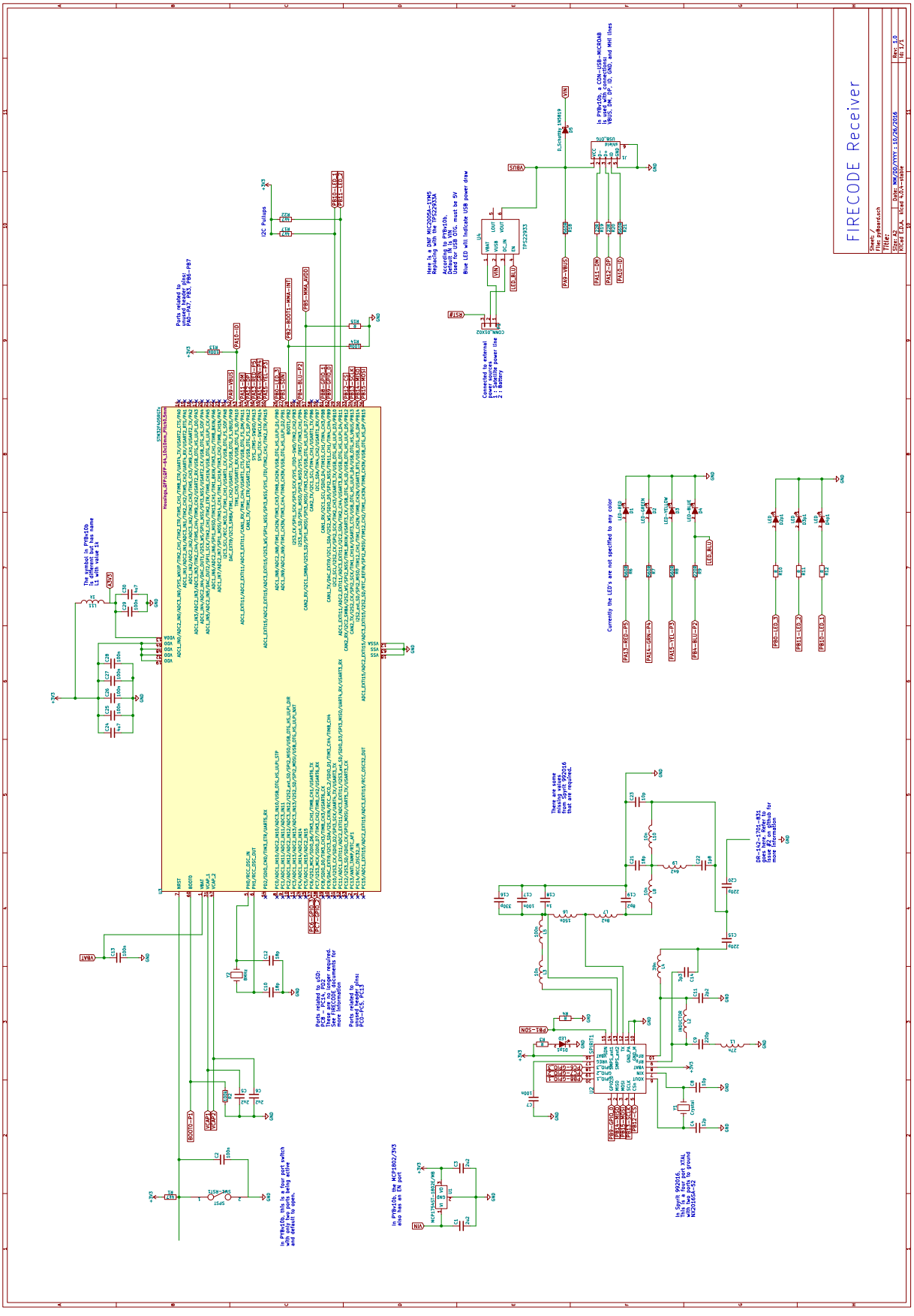


Figure A.2: FIRECODE schematic

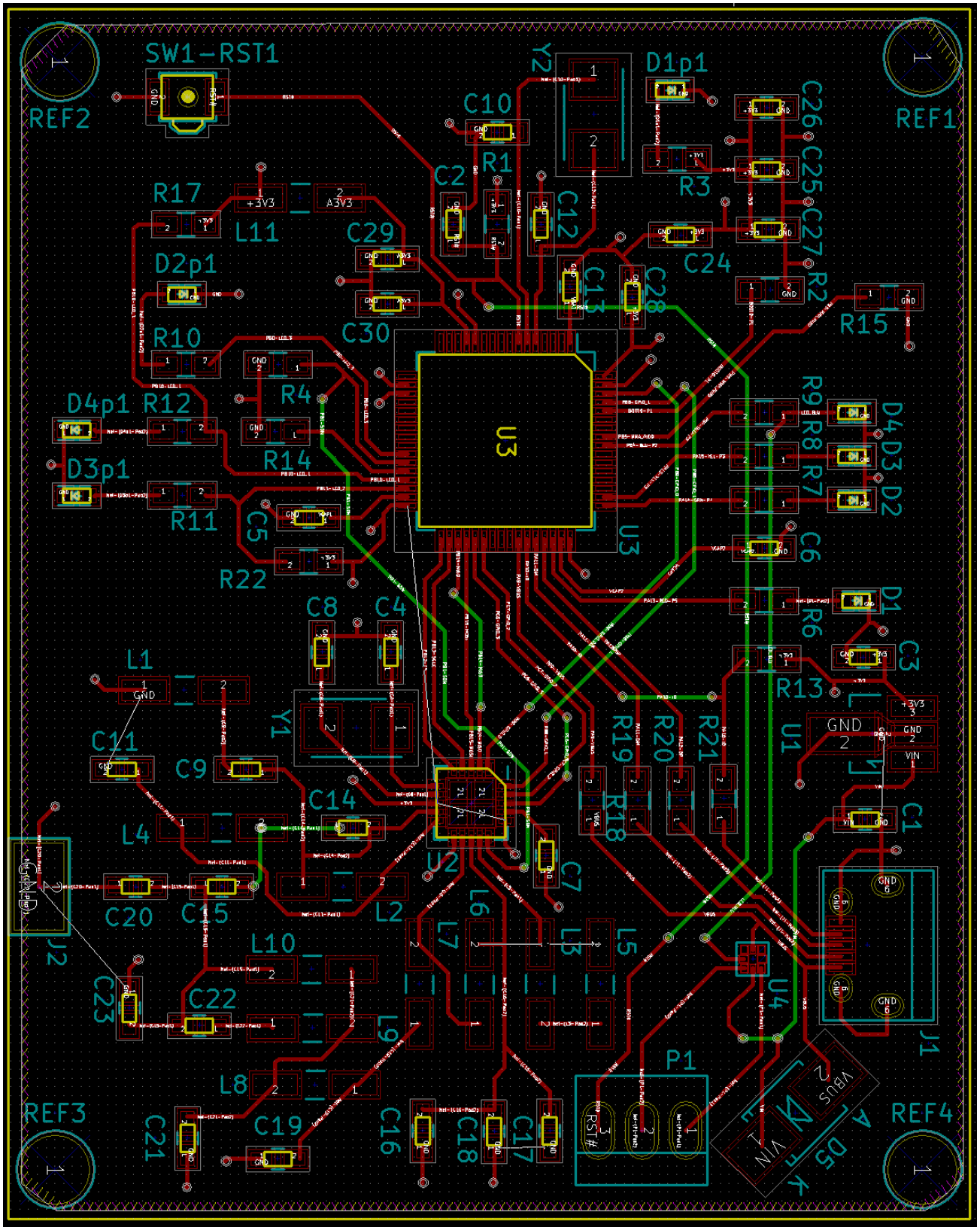


Figure A.3: FIRECODE 2D Layout

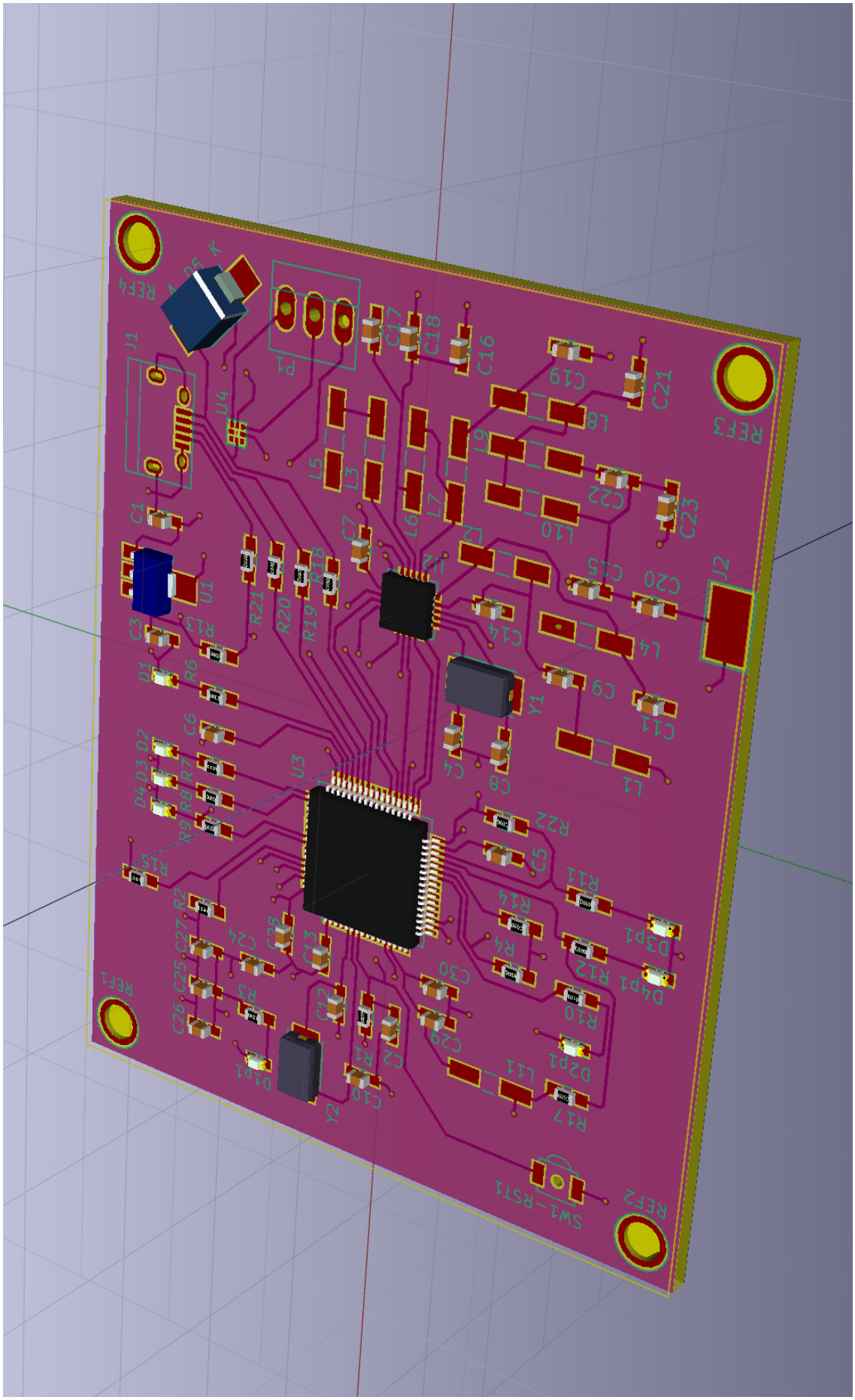


Figure A.4: FIRECODE 3D Layout