

CAN Based Flight Computer and Avionics System for Small Satellite Applications

A project present to
The Faculty of the Department of Aerospace Engineering
San Jose State University

in partial fulfillment of the requirements for the degree
Master of Science in Aerospace Engineering

By

Aric M. Garcia

May 2015

approved by

Dr. Matt Messana
Faculty Advisor



© 2015

Aric M. Garcia

ALL RIGHTS RESERVED

The Designated Project Committee Approves the Project Titled

CAN BASED FLIGHT COMPUTER AND AVIONICS SYSTEM FOR SMALL
SATELLITE APPLICATIONS

by

Aric M. Garcia

APPROVED FOR THE DEPARTMENT OF AEROSPACE ENGINEERING

SAN JOSÉ STATE UNIVERSITY

December 2015

Dr. Nikos J. Mourtos, Committee Chair
Department of Aerospace Engineering

Date

Matt Messana
Skybox Imaging + Google

Date

ABSTRACT

**CAN Based Flight Computer and Avionics System
for Small Satellite Applications**

by

Aric M. Garcia

Spacecraft are nearly full autonomous systems and some are completely autonomous but either way a flight computer and avionics system is always required to maintain the health and safety of the spacecraft and to control all aspects of the various subsystems defined by SMAD. Currently most satellites use an inter-computer communication bus similar to most vehicles except that most of what is used is still not as good as your average sedan. The automotive industry uses controller automated network as the light weight and high speed communication system that carries information throughout the various systems. Also modern day aerospace engineering students and space-tech aspiring high school students don't always have a firm understanding of electronics and programming or how to combine the two. Therefore there is a need for a cheap, affordable, and robust spacecraft flight computer that is can be easily assimilated into any small sat project.

Table of Contents

1. Introduction.....	5
1.1 Motivation.....	5
1.2 Background.....	5

1.3 LEO Radiation Environment: Van Allen Belts.....	8
1.4 Past Small Satellite Flight Computer Designs.....	12
1.5 CAN Bus Background.....	13
2. Theory.....	15
2.1 Basic CAN Design.....	15
2.2 Error Catching.....	18
2.3 Basic CAN Operation.....	18
2.4 System Clock and Advanced Message Priority.....	19
2.5 Subsystem Hardware.....	20
2.6 Subsystem Redundancy.....	22
2.7 Integrated Development Environment.....	23
References.....	24

1. Introduction

1.1 Motivation

I started working at Skybox Imaging in January 2014 and since have learned a lot about spacecraft hardware. Mechatronics and spacecraft have always been my interests and the avionics and command and I have learned that the avionics and command and data handling subsystems are very much what I enjoy about engineering. Other

motivations are my fellow aerospace students who do not have a firm grasp on or do not understand programming and circuit design and how to combine the two. If I have the ability to design a system to help improve the capabilities of aspiring students in spacecraft engineering the opportunity should be capitalized upon.

1.2 Background

Space flight and space exploration is the newest, most extreme, and least explored of the environments known to man. It introduces an environment that is a complete vacuum of extreme cold or intense heat where radiation is abundant. Having the ability to transverse such an expanse is not only a challenge but also a necessity for humanity. Catching a ride to this frontier and traversing it is incredibly expensive and dangerous and therefore requires accurate and precise dynamic control of all aspects of a space vehicle. So it is up to engineers to build systems that not only are resistant to the harsh environment but can maintain absolute control for the safety of the payload (both human and scientific) even in the event that a failure in one or more systems were to occur. In this light it is desirable to design an autonomous and intelligent control system that can resolve any errors or issues that it comes across on its own to ensure complete and consistent operability of the spacecraft. This system is called the spacecraft flight computer and is the primary component of the command and data handling (CDH) subsystem. They have been around in aircraft since the early 1930's and the first use of a computer for space system control was the lunar extraction module during the Apollo moon missions. Since then spacecraft flight computers have been used on nearly every space mission. Flight computers are very expensive because they must be very robust and

reliable. If a computer were to fail on startup or anytime through the mission without recovery the mission would be a bust and the total cost of the mission in time and money would go to waste. Cube satellites, known as cube sats, that are built by college and high school students also require flight computers. However the cost of a powerful flight computer is far too great for most students to afford and is too big and bulky to fit in a cube sat frame. So small unreliable microcontrollers that are prone to failure are generally used for use aboard such spacecraft. Also in general the code complexity is too great for most young engineers, which cause them to tend towards simpler less reliable microcontrollers. This is reason for a cheap, simple, and reliable controller that a student learning about spacecraft and space systems could easily afford and program for a basic satellite project.

Flight computers have been in use since the early 1930's in the first of the fly by wire aircraft when the use of electricity to maintain control of a planes attitude and control surfaces became important. Later it became a crucial part of plane operation when control of supersonic aircraft became too difficult for pilot control alone. The soviet Tupolev ANT-20 [16] used the first fly by wire system but the first non-experimental plane that used fly-by-wire was the Avro Canada CF-105 Arrow. Invert wing fighter jets like the Grumman X-29 would be uncontrollable if not for the flight computer [11]. Because the jet had very unstable poles it needed to have a highly computerized fly by wire system to maintain stability.

In the late 1960's when the Apollo missions were beginning to put men on the moon the first spacecraft flight computers were used to maintain control of the spacecraft and the safety of the crew and payload. Many fault alert systems were built into the

spacecraft computers, which in some cases saved the lives of the crew. The computers that were used at the time were iron core memory. Basically it was a system of copper wire that flowed in and out of iron donuts. A donut would act as a transistor in the memory and binary signals could be created that would be used to communicate to other systems and maintain control of the overall system. That kind of computer was very large and bulky but was also resistant to radiation since bit flips are not possible with the iron core memory. [15]

Iron core memory was used in other space missions since but only in part due to the fact that it was extraordinarily large and heavy with far too little computational capacity. The space shuttle used iron core memory in the AP-101B computer because of the early state of semi-conductor



Figure 1. Iron core rope memory

technology and the resistance it proved to radiation. However the space shuttle was constricted to approximately 400kB of memory. Modern spacecraft use integrated circuits that are less resistant to radiation but are radiation hardened. Radiation hardened means that it is made with thicker and bulkier parts that increase radiation resistance or just a shield against the suns radiation with thick covering. [4]

Skybox Imaging of Mountain View California currently is using numbers in hardware and incredible hardware redundancy and powerful fault tolerance logic to combat radiation-induced problems. At the heart of these small satellites is a system of small non-rad hardened computers connected through a high speed CAN bus communication network to create a cheap but effective spacecraft flight computer and

avionics system. It is a higher risk to include non-rad hardened parts but it is cheaper in the long run. This concept has been proven through the satellites at Skybox.

1.3 LEO Radiation Environment: Van Allen Belts

The radiation environment that is viewed by satellites and other spacecraft in orbit about earth is a dangerous hazard. Although usually predictable the radiation cannot be avoided and therefore must be designed for. The source of radiation about earth comes primarily from the Van Allen belt which surrounds earth. The Van Allen belt is an area of highly charged ions or radiation that has accumulated and move very quickly around earth. The Van Allen belts are a result of the earth's magnetic field and its interaction with the sun. There are two belts that surround earth the first of which begins at an altitude of approximately 1000Km and goes to 6,000Km. The second begins at 84,000Km and extends to 36,000Km [6]. The existence of the belts was discovered in 1958 by the explorer 1 spacecraft and was further studied in 2012 by the Van Allen probes also known as the Radiation Belt Storm Probes (RBSP) [10]. It was discovered that high solar winds can cause the upper Van Allen belt to push up against the inner belt and for the lower belt to push lower into earth's plasmasphere. It was also discovered that there exists a region in low earth orbit where inconsistencies in earth's core cause the lower Van Allen belt to drop to around 200Km altitude. This region of space is called the South Atlantic Anomaly (SAA) and is a danger to small satellites that are usually covered in a radiation-unburdened bubble below the first belt. The SAA is located near the South Atlantic Ocean and is situated directly above the lower tip of South America. The Van Allen belt also dips at the north and south poles where earth's magnetic field flows through [10].

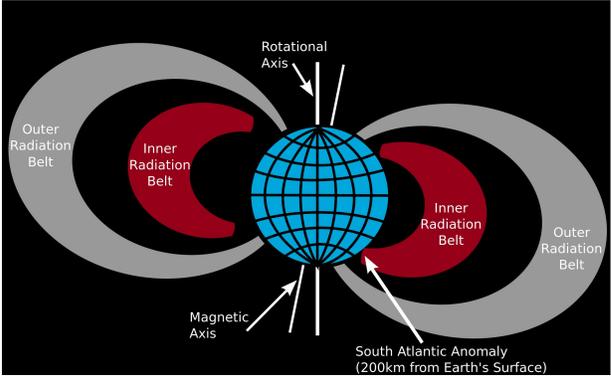


Figure 2. Van Allen belts showing SAA location [6]

6,000Km. The second begins at 84,000Km and extends to 36,000Km [6]. The existence of the belts was discovered in 1958 by the explorer 1 spacecraft and was further studied in 2012 by the Van Allen probes also known as the Radiation Belt Storm Probes (RBSP) [10]. It was discovered that high solar winds can cause the upper Van Allen belt to push up against the inner belt and for the lower belt to push lower into earth's plasmasphere. It was also discovered that there exists a region in low earth orbit where inconsistencies in earth's core cause the lower Van Allen belt to drop to around 200Km altitude. This region of space is called the South Atlantic Anomaly (SAA) and is a danger to small satellites that are usually covered in a radiation-unburdened bubble below the first belt. The SAA is located near the South Atlantic Ocean and is situated directly above the lower tip of South America. The Van Allen belt also dips at the north and south poles where earth's magnetic field flows through [10].

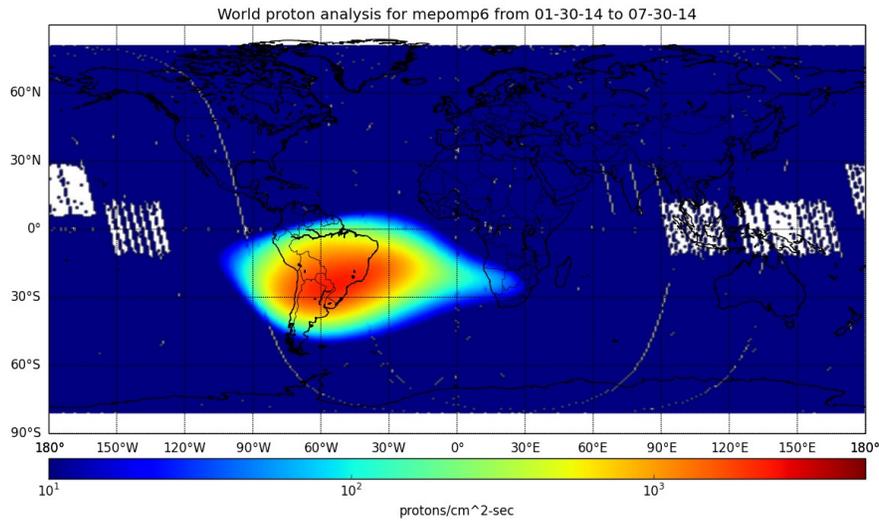


Figure 2. SAA as seen January 2014 [6]

The radiation environment can cause many problems in the electronics that make up the computation and avionics systems. The problems that can occur are but are not limited to: TID, LET, SEE, SEU, SEL, SEFI, SEB, and SEGR. Each is described in Table 1. Most typical forms of radiation issues come in the form of an SEU which depending on where it occurred may require a power cycle of the device affected. [6]

Table 1. Radiation problem types [6]

Issue Type	Description
Total Ionizing Dose (TID)	Material damage caused by ionizing radiation sources. Quantified by deposited energy per mass for a given material with units of Gray (SI) or Rad. Total ionizing dose is an accumulated property so the longer something is in a radiation environment, the more TID it will have accumulated. TID damage will eventually cause devices to fail or go out of spec. Most commercial components can withstand 10-20 kRad(Si) (that's 10,000-20,000 Rad(Si)). The (Si) indicates that we are talking about damage in silicon which is important because the same radiation source will not produce the same TID in different materials.
Linear Energy Transfer (LET)	Rate at which energy is deposited in matter as an ionizing particle travels through. Typical units are MeV/cm or scaled by material density as MeV-cm ² /mg. You can think of LET as the derivative of TID, although a particles LET is also very important in how likely it is to create different SEE's.
Single Event Effects (SEE)	Disruption in function of electronic circuits due to single ionizing particle interaction. Table 2 contains types of SEE events.

Table 2. SEE problem types [6]

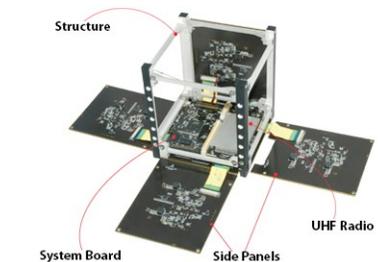
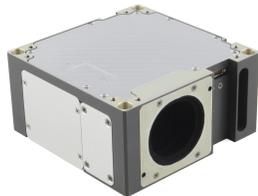
SEE Type	Description
Single Event Upset (SEU)	A "bit flip" or change in state of a digital logic element. These are, by definition, reversible. Generally refer to things like memory (SRAM, DRAM) or registers, etc.
Single Event Latchup (SEL)	Turn on of the parasitic SCR inherent in many CMOS processes. Typically will cause an increase in current draw, heating and potential burnout of the device. For "non-destructive" SEL's the only way to clear the SEL is to remove power from the device.
Single Event Functional Interrupt (SEFI)	Very much like an SEU but in an area of logic or memory that causes a reconfiguration of hardware. An example would be lockup of a microprocessor due to an SEU in a register that causes it to reconfigure some internal hardware (like a PLL) in an unrecoverable way.
Single Event Burnout (SEB)	An effect specific to power control devices, specifically MOSFET's. P-channel devices generally believed to be immune to this effect (but not N-channel!!). Voltage derating of a part helps prevent. Additionally a conducting part generally will not undergo SEB - generally happens when the part is "holding off" power.
Single Event Gate Rupture (SEGR)	Shoot-through of a MOSFET's gate dielectric due to charge injection from high energy particles. Generally exhibits itself as a short from gate to source/drain and is catastrophic for the device.

1.4 Past Small Satellite Flight Computer Designs

The idea of small sat in a box is not a new one. Many enterprises are developing integrated small satellites that can be used with a project. One system is from Blue Canyon Tech called the XB1 high performance 1-U cube sat. The system incorporates in parts made in house for GNC pointing and navigation. The overall design is built for high performance. The structure is made so that modules could be stacked for multiple satellite sizes and can be launched from any conventional cubesat deployment system. The computer has SPI, and 6 12bit analog telemetry channels and 6 discrete inputs. The SPI can be used to control experimental devices and the coding is a flight simulator that is built in house as well [1]. The entire system is built in house. This may be difficult for students trying to program their system since it is new unconventional software that they have likely never worked with.

Another system that is on the market today is a Pumpkin product that incorporates a PIC microcontroller and a TI microcontroller with a Pumpkin cubesat frame. The product is called CubeSat Kit. The coding is done in C code and Pumpkin provides libraries that can be included that incorporate specific spacecraft commands for simplicity. This system is more flexible than Blue Canyons XB1 and is programmed through a very fast and known form of coding. Pumpkin also offers libraries of code that makes it very simple to perform spacecraft specific commands [5]. However with only 2 unprotected controllers, this computation design might not fare well if it were hit by radiation.

One other system that incorporates small satellite in a box is Terran Orbital. The offered options for cubesats are the intrepid and the endeavor modules. These designs are a very integrated cubesat that is already built to spec with power systems and computation systems included. Experimental devices can be included through SPI and other than that every sub system is already built and mounted. The system is even outfitted with an input for a CMOS image sensor like that of a cell phone camera. There is only one computer on board which is the ARM9 based AT91SAM9G20 from Atmel industries [14]. This system is very similar to the XB1 except that it takes more of the freedom of space system design away.



A.

B.

C.

Figure 4. A: XB1 integrated satellite designed for stackability. B: Terranorbital Endeavor general satellite body also designed with all systems attached similar to the XB1. C: Pumpkin cubesat kit, outfitted with a Pic microcontroller on a pumpkin frame and libraries that simplify programmability.

1.5 CAN Bus Background

Controller automated network (CAN) is a high-speed inter-computer communication system. It was designed so that all systems in a vehicle could be directly connected to one another and still be independent. However CAN also provides much lower weight in hardware as compared to traditional communication bus techniques. This lower weight came as a side effect due to the simplicity of CAN. Less need for a complicated wire system decreases weight on a vehicle. CAN was created in 1983 by Robert Bosch and his company when it became evident that traditional inter computer communication techniques were far too slow for the automotive industry to effectively use in automobiles. In 1986 it was presented to the SAE congress in Detroit and in 1993 CAN became an international standard for cars. Today CAN can run at up to 1 Mbit/sec and is used by many mechanics and citizens to diagnose problems with their cars. This system has also been used and implemented with many other vehicles including aircraft and boats. CAN is an international standard and all CAN integrated circuits must comply with the Bosch CAN reference model. Since then many international standardization organization codes (ISO) have been outlined regarding CAN specifications. [3]

In satellites inter-computer communication is an important aspect for the avionics and CDH subsystems. All satellites have a way of handling serial data flow and

commanding between hardware. Small sat projects from colleges or high schools will usually use I2C or SPI communication with hardware and one microcontroller or two working in parallel as an error check. Small sat systems like these while cheap are more prone to error. Some current satellites use CAN as a



Figure 5. SkySats 1 and 2 in the SkyBox clean room.

solution to a cheap robust control system. One is the SkySat satellite designed and built by Skybox Imaging. These satellites are built with two redundant CAN bus systems and over 28 microcontrollers 14 of which are redundant. The entire satellite except for the single non-volatile memory chip is not rad hardened making the satellite far cheaper than an average satellite of its size. The satellite flies through the south Atlantic anomaly approximately 6 times per day and sometimes experiences the normal radiation induced problems in the different controllers but thanks to the CAN system the controllers are able to notify each other easily when one system is down and be able to quickly and efficiently restart it or replace it.

2. Theory

2.1 Basic CAN Design

When Robert Bosch set out to invent CAN in 1983 he intended it to be a fast system that did not require a vast network of cabling and wires as most controller networks were. The main idea is that every system can hear and talk to every other system [3]. The idea of CAN can be thought of as similar to a business conference table

while traditional Controller networks can be thought of as a game of telephone. When a command is sent from the flight computer to a subsystem it must travel through a chain of other systems before reaching it similar to a boss giving an order to the second in command who passes it to the third and so on to the desired person down the chain. Other systems have direct cable attaching each system that needs to communicate directly to each other. Both of these systems can cause a serious mess of wires and be too complicated to trouble shoot. Also this system is far too slow at communicating messages across the bus. Instead CAN uses a conference table type of system. When one person around the table speaks all can hear him. The message gets sent directly to every single controller listening. However only the controllers that care about the message will actually take in the message. The flight computer can be thought of as the boss at the head of the table while the people around the table are all the chains of command all listening at the same time. If the boss or any subordinate wants to talk to any other subordinate around the table all he needs to do is specify his message for that individual. Communication is direct and universal. And as a side effect results in much less need for complicated wiring. [2]

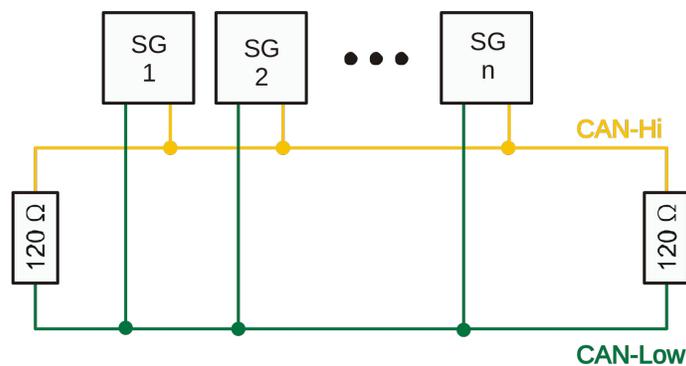


Figure 6. CAN Electrical Setup. Very simple wiring is required on a CAN system. The simplicity cuts down on both weight and cost of the system. [8]

A CAN network is made of two lines, a CAN high (CH) and a CAN low (CL). The CAN controllers flip an electronic switch back and forth very quickly to send a message out across the wires. When there is a high logic voltage on the CL there is a low logic voltage or 0 volts on the CH and vice versa. The quick flipping of these two signal wires is what creates the binary CAN frame that carries the message. All controllers that are to be attached to the CAN have a CH and CL terminals. All CH terminals are attached to the same node and all CL terminals are attached to the same node. All controllers can see the same message because all of their CH and CL terminals are attached. The CL and CH should always be separate but due to the quick flipping it is possible that electrical shorts can occur. To prevent shorts between the CL and CH a 60Ω resistance is placed between the two nodes. Since it is desired to put a resistor on both sides of the bus two 120Ω resistors are placed in parallel on either end of the CAN that equates to one 60Ω resistor. [8]

The computational mechanism that consists of the traffic in a CAN bus is called a CAN frame. The frame is the binary signal that encapsulates the message, prioritization, and error catching algorithms. When a message begins it sends a start of frame bit followed by the 11-bit arbitration field. To arbitrate is to settle a dispute and the arbitration field is meant to settle the dispute between CAN frames sent at the same time. The arbitration field that makes up the highest binary number will win priority of the bus to send data. The other CAN nodes will back down and wait until after that frame is done before trying again. A node knows when a frame is done by observing the end of frame period signified by 7 consecutive bits being held high. The arbitration field also acts as the id or address of the message. A controller will always see every message but it can

choose to ignore the message based upon its id or arbitration field. This way priority and destination of a message are inter-linked and have to be considered when designing the control system of a CAN based spacecraft. Other areas of the CAN frame are the RTR, extension, and error bits such as the CRC field and the ACK field. [8]

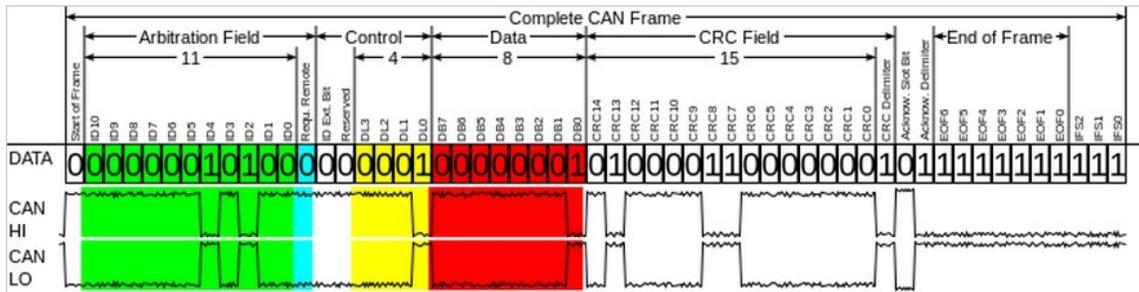


Figure 7. Standard CAN message frame.

2.2 Error Catching

The CAN system has a lot of methods for finding and catching errors in both the hardware and the transmission/receiving methods. In the CAN frame there are two fields that pertain to catching CAN errors, the CRC field and the ACK field. The CRC is a cyclic redundancy check also known as a check sum. In a CRC the total number of high bits from everything preceding the field is added up to a value and placed in the CRC field. The receiving node will take the same message and find the CRC using the same algorithm. If the receiving and transmitting numbers do not match then there has been a problem while transmitting. The ACK field is just a field that will be a one or a zero depending on if any other node has heard the message. If a dominant bit was not received from any other receiving node then there has either been an error with the received message or no nodes on the CAN were actually listening or exist. Other errors pertain to the hardware itself and will fault if there are errors that occur in the node itself. These

errors are called TEC and REC errors, which stand for transmitting/receiving error counter. These errors have different levels of severity and if an error count becomes too high then interrupts will be generated on key pins, which will shutdown all activity on that particular node. Usually the only way to recover from such errors is by giving the hardware a hard reset. [8]

2.3 Basic CAN Operation

Setting up a CAN network is very simple. Basic commands and C code libraries already exist that simplify the use of a node. When initializing a CAN system the speed of the transmission must be the first thing to occur. CAN is capable of transmitting at up to 1 megabyte per second but not all controllers on the market are up to the task. The clocks on the CAN must be similar speeds but must at least be able to operate on the same or faster speed than the CAN bus. The CAN can be configured for lower speeds if necessary but computational speed is important when operating a vehicle network. Other initialization parameters involve setting the mode of the device to normal operation. This involves simply setting a few specific bits in the node and making sure that it is always in normal mode. When receiving messages code can be set up to always check for a pending message and if one is found it needs to be received to any available buffer in the node. Where it can then be read and evaluated if it's id is really one the particular controller cares about. Transmitting involves defining the id and other frame parameters. The message at most can contain 8 bytes of data and each byte must be defined individually. Once the message has been successfully created all that remains is to simply send the message. [8]

2.4 System Clock and Advanced Message Priority

An option sometimes used in CAN systems with high amounts of traffic is setting a system clock. This is one of a few system designs where there is a master controller above the rest. This controller can send out a start of session message that will tell all the other controllers what kind of messages can be sent, primarily aimed at the priority of a message. This is essentially splitting up available time on the bus into timed sections that can be used for different priorities of messages. CAN is already designed to allow for higher priority messages to get through first but if a system has a very high amount of traffic with a lot of very high priority messages then the lower priority messages may get pushed back continuously until there is time to send it. Delaying any message for too long can be bad for the system. For example if there is an error in the payload system, critical navigation telemetry may not get through because it is not as important as the payload error data being transferred. Therefore setting allotted time for different priority messages may be necessary. However doing this means taking away a part of the freedom and most useful abilities of the CAN system. Priority is already taken into account and setting a system clock would only slow down transmission of all messages. Doing this means creating caches of messages on each controller so that the messages can be stored until the time to send those priority messages arrives. This adds complication to all of the subsystems on the CAN system. A smaller satellite may not require a lot of traffic to be sent around so it may be more beneficial to not use a system clock. This is one design parameter that must be carefully considered and if necessary implemented. [8]

2.5 Subsystem Hardware

A satellite's hardware like reaction wheels and gyros are all similar in the fact that they are hardware that talk through an SPI or I²C relationship with a controller. Most products on the market have similar startup and shutdown procedures and provide telemetry and receive commands for control and configuration. Therefore it may be possible to generalize the procedures that control any piece of hardware. [12] [13] This is an important aspect to the overall system and will be the backbone of the satellite subsystem architecture. Digging through data sheets to understand basic system operations and commands will become important for designing specific commands for configuration and control. Also CAN IDs need to be created for each controller individually that reflect message priority and priority of a certain system. No two IDs can be used to send different messages so in some cases it may be necessary to have generic messages that could emanate from any system carrying the same message. [2]

The since some systems critically depend on certain kinds of telemetry it will become critical to design a ID allocation protocol based on message importance and what kind of systems need to have the most fluid communication. For example the NAV system would need to have a strong communication with the ACS system for critical pointing efficiency and accuracy since the control algorithms will depend on each other. The systems that need to be taken into account in a basic micro satellite would be among the following: power (PWR), navigation (NAV), attitude control system (ACS), communication (COMM), and experimental system (EXP). The experimental system will be the component that moderates and controls any on board experiments that a cube

satellite may contain. Below is an example of ID allocations for the CANopen integrated system. [8]

Index (hex)	Object
0000	<i>not used</i>
0001 – 001F	Static Data Types (for reference only)
0020 – 003F	Complex Data Types (common to all devices)
0040 – 005F	Manufacturer Specific Data Types (device specific)
0060 – 1FFF	<i>reserved</i>
2000 – 5FFF	Manufacturer Specific Profile Area
6000 – 9FFF	Standardised Profile Area
A000 – FFFF	<i>reserved</i>

Figure 8. CAN open Object Dictionary Structure. Example list of ID allocations for the CAN open system.

Dynamic control theory combined with CAN operation will be required in all subsystems. For example when the ACS controller receives navigation information from the NAV controller it needs to be able to decipher the information and make it useful. This is where dynamic control solutions come into play. Since the hardware for this system is supposed to be able to be moved into any desired position multiple types of dynamics system setups need to be considered as possibilities and checks on controllability of the system designed ensured through checks in ACS system’s code.



Figure 9. Sinclair Star trackers and Reaction wheels similar to what is flown on SkyBox’s SkySat’s. These devices contain many sensors for health and safety which is a desirable trait to have in a device.

2.6 Subsystem Redundancy

Besides the primary function of the flight computers subsystem setup there must be algorithms that catch errors and respond with an appropriate resolution to right the issue. Already CAN is designed to catch errors in transmission but other techniques must also be used to correct faulty hardware or a radiation hit. Typically the correct response to a radiation hit or an out of sync piece of hardware is to do a power cycle, which will clear out the memory of the system and may possibly fix the issue. If it persists it may have an uncorrectable error. In this case it must either use a radiation resistant Non-volatile memory to reload its own code or simply fail over to a duplicate system. Therefore it may be more desirable to have multiples of the same controllers for redundancy in the case of an SEB or SEGR (refer to table 2). Other protocols that need to be designed for are in the event of communication loss, repetitive system failure even after reset, and safe orientation failsafe. All of these protocols need to be automated so that in the event of a major event the health and safety of the satellite can still be ensured.

2.7 Integrated Development Environment

After the final design of the system is built it needs to have the ability for simplistic programming for the first time satellite builder. The best option for making a basic programmer is to base it off of Matlab since the program is already built for heavy mathematics and is capable of solving complex dynamics and optimization problems. However implementing Matlab into the design means integrating a computer with an operating system capable of utilizing it. Integrating this into the CAN might be difficult and an entire project of its own. This is why it might be more desirable to use another

microcontroller as the main controller and using a C based system. While C is much faster it is also much more complicated to use. Also using Matlab would mean that the overall cost of the system could greatly increase since Matlab is very expensive and so is a micro cpu that could run it. Matlab will first be attempted and if it proves too complicated for the project timeline it will be abandoned for a simpler and cheaper C controller. The programmer called an IDE or integrated development environment will need to have functions that can be easily used by the programming engineer to easily use all of the hardware in a useful way. Functions like quaternion hold could utilize both the NAV and ACS controllers to help maintain a quaternion in space. Similar functions can be created for various other commands needed by the spacecraft that would cover all of the basic commands that might be needed while programming the behavior of a satellites autonomous function.

References

- [1] bluecanyontech,. 'Precise 3-Axis Stellar Attitude Determination In A Micro-Package'. N.p., 2015. Web. 2 May 2015.
- [2] Can-cia.org,. 'CAN In Automation (Cia): CAN Protocol'. N.p., 2015. Web. 2 May 2015.
- [3] Can-cia.org,. 'CAN In Automation (Cia): CAN History'. N.p., 2015. Web. 2 May 2015.
- [4] Cpushack.com,. 'CPU History - Computers And Cpus In Space'. N.p., 2015. Web. 2 May 2015.
- [5] Cubesatkit.com,. 'Cubesat Kit Home'. N.p., 2015. Web. 2 May 2015.
- [6] Dyer, J. (2014, January 1). Background on Radiation Effects. Retrieved February 1, 2015.
- [7] Khurram, M., S. Muhammad, and Y. Zaidi. 'CAN As A Spacecraft Communication Bus In LEO Satellite Mission'. *Proceedings of 2nd International Conference on Recent Advances in Space Technologies, 2005. RAST 2005.* (2005): n. pag. Web. 2 May 2015.

- [8] Lawrenz, W. (1997). *CAN System Engineering From Theory to Practical Applications*. New York, NY: Springer Science Business Media.
- [9] Markley, F. Landis et al. 'Maximum Torque And Momentum Envelopes For Reaction Wheel Arrays'. *ntrs*. N.p., 2015. Web. 2 May 2015.
- [10] NASA,. 'NASA's Van Allen Probes Spot An Impenetrable Barrier In Space'. N.p., 2015. Web. 2 May 2015.
- [11] NASA,. (2015). *NASA Armstrong Fact Sheet: X-29 Advanced Technology Demonstrator Aircraft*. Retrieved 4 May 2015, from <http://www.nasa.gov/centers/dryden/news/FactSheets/FS-008-DFRC.html#.VUbsdDvF8gA>
- [12] Sinclairinterplanetary.com,. 'Reactionwheels - Www'. N.p., 2015. Web. 2 May 2015.
- [13] Sinclairinterplanetary.com,. 'Star Trackers - Www'. N.p., 2015. Web. 2 May 2015.
- [14] Terran Orbital Corporation,. 'Intrepid Platform - Terran Orbital Corporation'. N.p., 2015. Web. 2 May 2015.
- [15] Tomayko, J. (1985). NASA's Manned Spacecraft Computers. *IEEE Annals Hist. Comput.*, 7(1), 7-18. doi:10.1109/mahc.1985.10009
- [16] Spearman, M. (1985). *Some comparisons of US and USSR aircraft design developments*. *Ntrs.nasa.gov*. Retrieved 4 May 2015, from <http://ntrs.nasa.gov/search.jsp?R=1986000673>