PolySat's Next Generation Avionics Design

Greg Manyak
Electrical Engineering Department
California Polytechnic State University
San Luis Obispo, USA
gregmanyak@gmail.com

John M. Bellardo
Computer Science Department
California Polytechnic State University
San Luis Obispo, USA
bellardo@calpoly.edu

***Abstract*—The CubeSat platform provides a unique challenge for flight software design due to the incredible size and power constraints. A number of tradeoffs must be made to balance effectiveness, fault tolerance, and cost. These basic requirements have been combined with the lessons learned from Cal Poly's past 8-bit avionics system to design a significant revision based around a 32-bit microprocessor running Linux. This work analyzes both generations of avionics design, including a discussion of major design principles that are relevant to other CubeSat missions.**

***Keywords*-cubesat software design, fault tolerance**

I. Introduction

The CubeSat program was initially developed with universities in mind, enabling cost-effective access to space for educational purposes. This created an interesting combination of requirements. The physical form-factor and power budget of the satellites are extremely constrained. The number of available person-hours to design, build, and operate these satellites is small. The mission demands are always increasing, and the expected turnaround time keeps decreasing. Onboard hardware and software information technology is critical to meeting all the requirements.

This work describes how PolySat's onboard avionics strives to meet these demands, including a system overview of two generations of avionics design. Operational experiences and bus telemetry data are used to evaluate how well the first generation bus met the requirements. A set of design principles for the second bus generation is presented, including some justifications based on experiences from the first design. The second generation design is validated through an external design review process. Finally, other CubeSat designs are surveyed.

The remainder of the paper is organized as follows. Section 2 reviews the major design requirements. Section 3 describes the first-generation onboard avionics design, including an analysis of how well it meets the requirements. Section 4 establishes some basic CubeSat design principles. Section 5 presents the second-generation onboard avionics system. Section 6 reviews systems designed by other organizations. Future directions are discussed in section 7, and section 8 concludes.

II. Avionics Requirements

CubeSat avionics bus requirements are satisfied using a combination of software and hardware. The requirements presented in this section are used to evaluate and compare the different bus designs presented in this paper. Some of these requirements are not universally applicable to all CubeSat missions. For instance, PolySat has flown and intends to continue to fly, multiple missions with the same avionics platform. This results in more emphasis on flexibility requirements that may be unnecessary in planned single-use designs. The requirements are broken down into three broad categories: physical requirements, project personnel requirements, and mission requirements.

A. Physical Requirements

Physical requirements are determined by the CubeSat standard [1]. The smallest "1U" satellite is $10 \times 10 \times 10$ cm^3 (1 liter volume) and a "3U" satellite is 3L. These very small sizes make it extremely important to minimize the space occupied by the avionics. Every additional cubic centimeter occupied by the bus reduces the size of payload that can be flown. Initial CubeSat missions were focused on the educational aspects of designing the satellites and demonstrating the viability of the platform; thus they were not as concerned with the payload capacity. More recent missions still emphasize the educational impact, but also place more importance on the scientific mission. The satellite is also limited to 1.33kg per L. The weight budget does not typically constrain the avionics.

The power budget is also extremely limited. A standard 1U satellite covered with 6 solar voltaic panels has a power budget of roughly 1 watt per hour. It is extremely important that the avionics consumes as little power as possible, saving most of the budget for the payload. Ideally, the same avionics bus should be capable of operating any size CubeSat.

B. Personnel Requirements

The PolySat project operates with unique personnel requirements, most of which are attributable to being in a university setting. The majority of the work is done by student volunteers. These students agree to work at least 10 hours per week on the project, however, this commitment

is inversely proportional to both coursework and facebook usage. A typical volunteer averages 80 hours per three month term. Educating the volunteers about the satellite's design and training them on good engineering practices further reduces the productivity window of a volunteer. Optimistically, an average student can expect to be a productive contributor for 2 years, or a total of about 480 person-hours. 480 person-hours is equivalent to 3 months working 40 hours per week. This necessitates using technologies that favor ease of use and popularity over performance, for instance supporting modern scripting languages.

The shortage of manpower makes modularity and reuse extremely important. It is necessary to use the same overall hardware, communications, and software design in many missions. Ideally all the satellite development effort is spent on creating and integrating the mission specific payload, not on tweaking the avionics to support the next mission.

C. Mission Requirements

The final set of requirements are mission specific. A number of these vary from mission to mission, and are difficult to predict in advance. However, a subset of these requirements are recurring when different mission planners independently develop similar requirements. PolySat has attempted to generalize the requirements to make the avionics design capable of operating a wider range of missions.

Most missions are interested in a quick turn-around time. In fact, that is one of the main advantages of CubeSats. They are very agile and should be able to react quickly to payload technology. Organizers aren't thrilled at the prospect of waiting two years or more to get the first data back from their cutting edge experiment. A short turn around time eases the training overhead for new volunteers because a single team with fixed composition is able to complete one mission.

Payloads continue to generate increasingly large quantities of data. For this data to be useful, it must be transferred back to earth. This necessitates a robust communication system, starting at the transceiver and extending through the protocols that ensure correctness of transferred data. Not carefully considering all aspects of communication can be a major problem. For instance, using the standard TCP-based file transfer protocol (FTP) results in link capacity being vastly underutilized.

Outer space is a much harsher and less forgiving environment than Earth. As such, missions typically require CubeSats to provide some level of fault tolerance. Transient errors should be handled gracefully. They shouldn't cause the satellite to enter into an unusable, unknown, or uncontrollable state. If possible, errors that corrupt data should be detected and the data points ignored. Depending on the type of experiment, it may be repeated to capture clean data. Components can be selected and the overall avionics designed to minimize the probability that a catastrophic failure occurs in critical memory, for instance selecting radiation-resistant storage to hold the satellite's software.

III. First Generation Bus Design

The first generation avionics bus was primarily designed in 2004 as part of Chris Day's Masters Thesis [2]. This design, with only minor revisions, was subsequently used in five distinct satellites, starting with CP2. The only major changes were in the payload, which is expected to be mission specific. CP1 and CP2 never reached orbit due to launch vehicle failure. CP3, CP4, and CP6 were flown and operated with varying degrees of success. CP5 is in the testing stage of development and has been manifested on the ELaNa VI mission for a launch sometime mid-2012.

A. Hardware Design

The first generation bus was built around two printed circuit boards: the electronic power system (EPS) and the command and data handling board (C&DH). The analog power components were mostly isolated to the EPS board and the digital components on the C&DH board. This physical layout reduced the amount of crosstalk and interference the analog components contributed to the digital components.

The EPS provided voltage rails, via DC-DC converters, to the entire satellite. Each rail is protected by a programmable "smart fuse" that limits the amount of current that can be drawn. The EPS board was also the mounting point for the two lithium-ion batteries, which were charged by the solar cells on the side-panels of the spacecraft.

The C&DH board contained both a single computing and two identical, redundant communication subsystems. The computing subsystem consisted of one 4MHz PIC18LF6720 microcontroller (referred to as the C&DH controller) with two 128KB I^2C EEPROMs for non-volatile storage. Each communication subsystem had the same 4MHz PIC18LF6720. The payload also contained its own, mission specific processor. Most components were interconnected via I^2C. Avionics specific components, such as those for magnetometers and magnetorquers, were connected to the C&DH controller.

B. Software Design

The software was split into three pieces that ran independently of each other on different microcontrollers. The communication software ran on the comms controllers, and was responsible for controlling the transceiver, decoding the data, and forwarding it over the I^2C bus to the C&DH controller. The comms controller could also transmit a packet received from the C&DH and directly respond to a subset of commands without C&DH intervention.

The C&DH responded to all commands for devices that it controlled, for example, transmitting avionics state and energizing the magnetorquers. It was also responsible for

generating the periodic beacons. Commands destined for the payload processor were forwarded on the I^2C bus. Depending on the mission and specific command, the C&DH would occasionally translate the ground command into one understood by the payload before forwarding it. The C&DH was also responsible for relaying results from the payload processor back to the comms.

The C&DH and comms software were mostly reused from satellite to satellite.

C. Fault Tolerance

The C&DH and EPS subsystems both included a number of fault tolerant mechanisms, most notably the fully redundant transmit / receive path. The C&DH microcontroller switched between the two at a regular interval, so that if one were to fail, half the communication would still be successful. If the failure were permanent, a command could be sent up to the satellite instructing it to use the good communication path for the remainder of the mission.

The power system also has a fault tolerant architecture. The two batteries are configured in parallel, each with their own protection circuitry. If one were to fail, the other would still be usable without interrupting power. Three separate DC-DC converters with smart fuses are used to power the C&DH and the two redundant communication chains. The smart fuses provide failure isolation when the load on one exceeds the programmed protection threshold.

Anecdotal evidence suggested that CP4's failure was a result of I^2C bus problems. To address this, an 8-bit CRC was was added to each inter-processor I^2C message[3] in the satellite. This provided three primary benefits. First, most corrupted messages were no longer acted on, preventing the satellite from entering some unintended states. The CRCs, coupled with acknowledgements added to the I^2C bus messages, enabled local retransmission of the message without relying on the Earth station to reissue the command. Finally, the CRC and related software changes enabled the collection of I^2C telemetry data.

D. Design Analysis

The first generation avionics design successfully met all physical design requirements, as evidenced by its multiple flights. The design just barely met the personnel requirements. Experience has shown it took between 18 and 24 months to produce flight ready hardware for a particular mission. This is uncomfortably close to the productive lifetime of a student volunteer. In many cases, primary developers would continue working on the weekends remotely to ensure early satellite operations were as successful as possible. The overall turnaround time should be reduced to better fit mission specific objectives.

Communications failure has been the primary reason the first generation design was not a universal success in meeting mission requirements. The RF and internal I^2C bus have been identified as possible causes for these problems. The next few paragraphs provide mission-specific analysis.

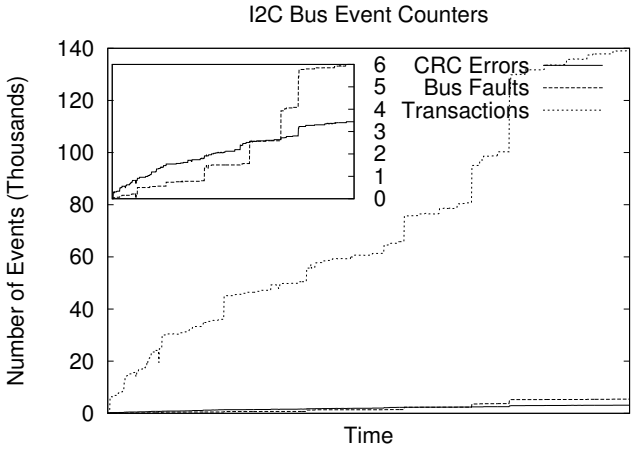

Figure 1. I^2C bus event data from CP6. Inset in upper left shows only the CRC error and bus fault event counters. Time progresses left-to-right, but not proportionally due to periods of time when no data was downlinked.

CP3 was launched on April 7, 2007[3] and can presently still communicate with the Cal Poly earthstation. The receive sensitivity of the communication system is such that very few commands are able to be received by the satellite, precluding extensive operations. However, a substantial amount of telemetry and bus health information continues to be received from this satellite.

CP4 was launched on the same vehicle as CP3 and although it used the same bus hardware, it was not as successful. CP4 suffered a failure soon after launch. An I^2C bus failure preventing communication between the C&DH and the communication controllers[3] is the suspected cause. Only commands sent to the communication processors have been successful, not commands sent to the C&DH.

CP6 was a backup flight model for CP3, with minor modifications. A low-noise amplifier was added to the front end of each RF communication chain, improving the receive sensitivity. As a result, CP6 had much greater operational success. CP6 is also the first satellite to report I^2C bus telemetry data. This capability was added because there was a strong consensus that I^2C problems directly contributed to problems in CP4. This data is shown in figure 1. Note that I^2C errors occur in less than 8% of the bus transactions overall. While this number is higher than ideal, it is low enough that it doesn't adequately justify blaming the I^2C bus in previous missions. These results reinforce the need to gather as much data as possible to focus engineering efforts in areas most likely to make the biggest difference.

CP6 stopped responding for a significant period of time. As of April 2011, only one of the communication processors occasionally responds to simple commands, and there is no response from C&DH commands.

IV. Design Principles

A small set of design principles was developed based on the design analysis of the first generation avionics bus, reviews of other CubeSat designs, and fundamental economic arguments. These principles were used in the design of the second generation bus. This section describes and motivates the principles, so they can benefit other CubeSat projects.

A. Minimize Hardware Modularity

Most engineers tend to design satellite hardware around a set of building blocks, and pick the appropriate combination of these blocks to complete the mission. An example set of building blocks includes a main bus, battery power management, communications module, attitude determination sensors, and a payload processor. These designs tend to use a common interconnect standard to enable vendor independence. PC/104 [4] is found in a number of CubeSat designs. For most terrestrial applications this makes sense because it "right-sizes" the hardware to the mission, minimizing the overall budget impact.

This is not the case for CubeSats. Assuming it costs roughly \$100,000 [1] to design, build, and fly a 1U CubeSat, the corresponding per-cm^3 cost is \$100. Each PC/104 board is 0.6in thick [4], or 1.524 cm. Adding a single additional PC/104 board consumes $10 \times 10 \times 1.524 = 152.4\text{cm}^3$. This represents a lost payload opportunity cost of roughly \$15,240, and is a reasonable measure of the expense of hardware modularity in CubeSats. This compounds when the base satellite design requires two or three modules to complete the mission, which is common in commercially available "modular" designs. A single board that contains substantially more functionality than necessary represents a better tradeoff, even if the board costs an extra \$500. This is a strong argument for *minimizing hardware modularity* in CubeSats.

B. Minimize Hardware Redundancy

Hardware redundancy is expensive in time, space, and price. Selectively duplicating components sounds appealing, but there are some major drawbacks. First, there isn't enough operational history available to identify the components that are mostly likely to fail. That reduces the effectiveness of redundant hardware, but not the cost. Second, there is a strong economic argument against redundant hardware. Namely, it is not worth increasing the cost or reducing the payload capacity of a CubeSat by \$5,000 to provide hardware redundancy for a small portion of the satellite. Instead, provide redundancy for *all* parts by building a second satellite, which costs substantially less than the first when built at the same time. If the first one fails the second one can be flown, as was successfully demonstrated with CP3 and CP6.

[1] \$100,000 is the current estimate given to potential satellite developers by the CubeSat program.

C. Eliminate Payload Controllers

There is a strong tendency to over-engineer the payload controllers. Specifically, most designs include a dedicated payload processor that monitors the experiment, takes commands from the main satellite processor, and returns data to the main processor. The payload itself is typically composed of a handful of sensors and actuators. This design appears "clean" because the communication interface between the payload processor and main processor is narrow and well defined. For example, the I^{2}C bus and messages used in CP2 – CP6.

The problem with payload controllers is that they substantially add to the development time of a satellite, especially when the payload is being purpose built and has a low likelihood of being reused with other hardware. The satellite developer must now write a number of additional pieces of software. The payload processor needs code to control the experiment, implement the communication bus protocol, and implement the custom messaging protocol on top of the bus. The most common payload processor used, PICs, do not have a reputation for being developer-friendly, which further increases the amount of time required. Finally, matching communication code must also be added to the main processor in order to communicate with the payload processor.

An alternative, streamlined design is to attach all payload sensors and actuators directly to I^{2}C GPIO expanders, A2D sensors, and other required components. From a hardware standpoint the interconnect is just as simple. The biggest difference is in the software. Developing the payload software to run directly on the main processor virtually eliminates all the time spent on developing communication bus and custom messaging implementations. This also increases the complexity of the main processor.

V. Second Generation Bus

The second generation bus was developed to meet the requirements described previously, guided by both the design principles and other past experiences. The resulting avionics system is the basis for two current missions, the 3U LightSail-1 and the 1U CP7. It will also serve all of Cal Poly's missions for the foreseeable future.

A. Hardware Design

The new design consolidates the C&DH and EPS components onto a single board, eliminating two of the three original processors. This necessitates a more powerful main processor. Since this processor is also expected to handle payload control and needs to enable more sophisticated software designs, a very powerful processor meeting the space and power budgets was chosen.

The C&DH is built around a 32-bit ARM9-based Atmel chip, with SDRAM and NAND flash as the primary storage. The detailed specifications are provided as part of

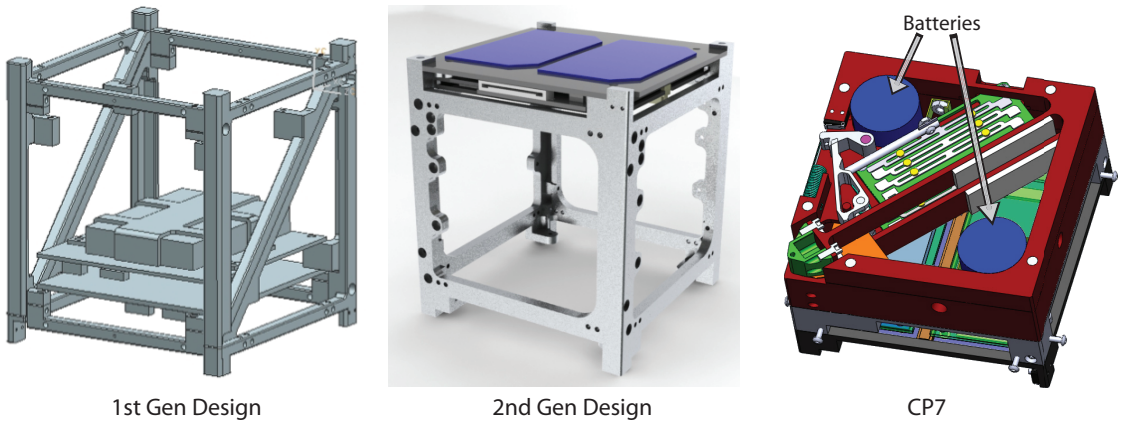

Figure 2. CAD Renderings of different PolySat satellites. The CP7 rendering illustrates how mission-specific batteries were utilized in the payload bay.

figure 4, but the CPU is roughly 2 orders of magnitude faster, and there is 3-4 orders of magnitude more storage. All components, except for the communication subsystem, are on the primary electronics board. The communication system is attached via a low profile daughtercard connector. Even though it is on a separate board, the transceiver is directly controlled from the main processor. There is no dedicated communications processor. I^2C and RS232 are used to connect most external sensors and payloads. A SPI bus connects the remaining components, including the transceiver. Other interfaces, such as USB, LVDS, and RS422, are available if needed in future missions.

The basic structure was redesigned entirely to support the second generation bus. Rather than the printed circuit boards sitting inside the spacecraft structure, the single board with the daughtercards sits underneath the 'top' solar panel. The resulting mechanical structure to support this avionics system requires significantly less volume than it's predecessor, as shown in figure 2. This 2^{nd} generation avionics consumes approximately .1L. When put in the smallest 1U satellite, this is roughly 10% of the volume. In a 3U the percentage is reduced to only 3.33%. These volume percentages assume the satellite's battery can be customized to work within the payload's physical constraints. For example, CP7 is able to utilize cylindrically shaped batteries, which fit well due to the unique mechanical payload, as seen in figure 2. In general the battery technology exists to make this assumption reasonable. This form-factor is small enough to conceivably fit into a 0.25L avionics-only satellite.

B. Software Design

Moving all the processing to a single processor places an increased burden and complexity on the software. Instead of implementing the control software from the ground up, which is a common decision among CubeSat missions, the avionics processor runs Linux. There are a number of benefits obtained from using Linux. First, it handles all the low-level management of hardware, drivers, communication, and processes. This eliminates a large amount of code that would otherwise have been written. More importantly, Linux enables the satellite to run much of the available open source software, including modern languages like Python. This last point is extremely important. Providing modern language support increases the number of developers able to contribute to the project and decreases the amount of retraining that is necessary. None of this would be possible without the generously over-provisioned hardware.

A highly modular and generic software system, centered around the process model of the Linux, was designed. The software architecture contains three distinct layers: processes, abstraction libraries, and drivers. Processes are single-function applications, abstraction libraries provide access to different software and hardware services, and drivers are used by the libraries to access the hardware. An example of this hierarchy is shown in the figure 3, specifically demonstrating that of the communication process.

The processes are the highest level of software in this bus design. Each module in this layer serves a very specific function in the system and heavily utilizes the API provided from the abstraction libraries. Processes communicate between one-another via UDP packets. Each process is assigned a well-known port that is listed by name in the standard services file. This abstracts the port numbers and simplifies adding new processes to a system. Each process runs an event handler system, responding to events from a limited number of sources. For instance, the communication process will respond to events from the transceiver to receive packets, events from the kernel to transmit packets, and timeout events to implement ARQ protocols(s). Timeout events are generated by an internal priority queue. The heap-based implementation always keeps the earliest timeout event efficiently accessible. When the process isn't actively responding to an event, it blocks, to conserve both CPU time and power. All of this functionality is provided by the

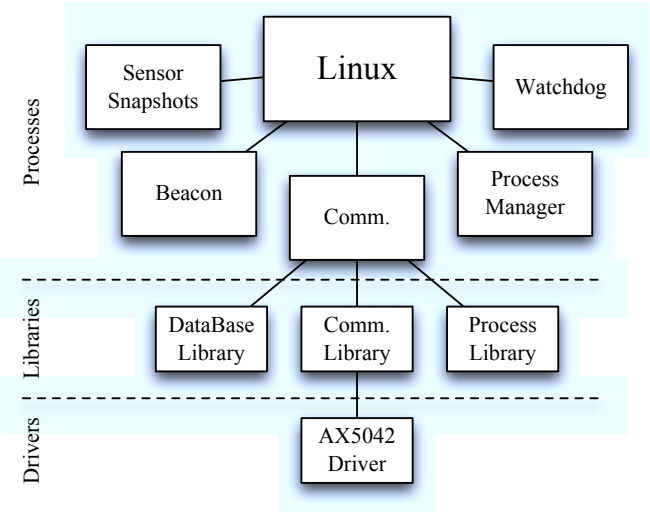

Figure 3. Process Chart with Communication Process Hierarchy

process library, a module of the abstraction libraries.

The abstraction libraries provide both hardware and software-based services to all of the processes. Low-level tasks are abstracted into methods that are meaningful to the process developer and hide the specific hardware components of the avionics system. These libraries are dynamically-linked at runtime so that they can be upgraded without recompiling every process that utilizes them. The libraries also reduce development time by providing a single implementation for commonly needed functionality.

The drivers are at the lowest level of the software hierarchy, containing code that interacts directly with the components of the system. The components may be included on the avionics board itself, in the solar side panels, or in the payload. The drivers may run within the Linux kernel or as a separate user-level process with special permissions to directly talk to a device. User-space drivers are desirable because of their inherent ease of development, isolation from other software components, and their ability to use any programming or scripting language. Most devices are interrogated so infrequently that the extra overhead from running in user space isn't noticeable.

C. Fault Tolerance

Rebooting the satellite is the primary mechanism for recovering from detected, transient faults. Although a complete restart is heavy-handed, it helps minimize the probability that another, undetectable fault impacts the mission. Some of the more common transient errors are SDRAM bit-flips or bus errors. In certain critical circumstances, a bus read operation is retried when a fault is detected. Watchdogs and checksums are used to detect corruption.

A hierarchical watchdog system is used to detect transient errors that result in external misbehavior of a process. The high level watchdog is implemented in software at the process level and watches each process, expecting a specific range of behaviors. This watchdog is robust enough to dynamically reconfigure its set of expected behaviors, however, care is taken to eliminate the possibility that a runaway process can inadvertently reconfigure the watchdog and avoid detection. The high level watchdog verifies behavior on the order of every 1-5 minutes. If any inconsistencies are found a restart is initiated.

The low level watchdog utilized is the Linux kernel's software watchdog, which is "tapped", unconditionally, at a much higher frequency than the higher level one. The timer interval for this watchdog is fixed, and the timeout behavior is a soft reboot, which will enable a refresh of SDRAM upon boot-up. The processor's built-in hardware watchdog and an external watchdog are used to recover from failures in the shutdown and boot-up sequence.

Checksums are used to verify the integrity of persistently stored data. The primary checksum used is MD5, which can run quickly on the main processor and has a much lower probability of random collision than standard IP and CRC checksums. To reduce the likelihood of getting into an unintended state, commands are given even stronger guarantees. A command is a ground-initiated message that triggers any kind of function in the satellite, such as taking a sensor reading or reconfiguring a process's behavior. An MD5 checksum alone is insufficient to ensure a rogue process doesn't issue commands that reconfigure the satellite.

To protect against that scenario, and to provide better operational security, commands sent to the satellite are digitally signed. Since the signing key will not be present on the satellite, this guarantees that any valid message must have originated from a trusted off-satellite source. The signatures also enable corrupted commands to be identified. Standard public/private key cryptography and X.509 certificates are used. The satellite is pre-loaded with a small set of trusted certificates, from which a chain of trust can be established to validate future commands. Each certificate lists the set of commands the holder is permitted to sign, enabling fine-grained operational security. Commands can be stored indefinitely in volatile or non-volatile memory and the signature properties still hold.

Special attention has been given to add robustness to the early stages of system boot-up. The Linux kernel and small initial set of processes is stored in SPI Flash, implement with non-volatile phase change memory. Phase change memory of this type has been shown to be radiation tolerant up to 30 MRad against permanent errors [5]. Multiple copies of both the kernel and initial files are kept in SPI Flash, and a custom boot loader validates checksums before transferring processor control to the kernel. To better tolerate transient bit errors, the boot loader will read the same data multiple times if the checksum doesn't validate. Again, MD5 checksums are used for extra robustness. Persistent bit errors in SPI

Satellite	Organization	Main Processor	Mhz	Volatile Memory	Non-Volatile Memory	Operating System
1^{st} Gen Bus	Cal Poly	PIC18F6720	4	3.75 KB	256 KB	custom
2^{nd} Gen Bus	Cal Poly	AT91SAM9G20	400	64 MB	528 MB	Linux
Cute-1.7+APDII	Tokyo Institute of Technology	ARMV4I	400	32 MB	128 MB	Windows CE.NET
MEROPE	Montana State	MC68HC812A4	16	1 KB	132 KB	custom
QuakeSat	Stanford University	ZFx86 486 Clone	100	16 MB	1 MB	Linux
AAUSAT-II	Aalborg University	AT91SAM7A1	40	2 MB	8 MB	*unavailable*
ITUpSAT	Istanbul Technical Unv. (Pumpkin Design)	MSP430	7.4	10 KB	50 KB	Salvo 4
STUDSAT	India - 7 Academic Institutions	AT91SAM9260	180	*unavailable*	*unavailable*	VxWorks

Figure 4. Comparison of processing specifications and operating systems from CubeSat missions. Chosen to demonstrate variability. Some designs, such as Pumpkin and PIC-based (like PolySat's 1^{st} generation design), have flown many missions. Data was gathered from public documents.

Flash can be corrected once an operational copy of the kernel is loaded.

NAND memory is used for high capacity non-volatile storage. The NAND specification permits a small number of permanent, unrecoverable errors to increase yield and decrease cost. The Linux NAND drivers already track these bad blocks and stop using them. It is expected that this existing mechanism will work well in orbit to recover from faults, however, the number of faults may be higher than on Earth.

D. Design Validation

The second generation avionics system hasn't flown a mission yet, so operational experience can't be the analysis basis for understanding how well the design requirements were met. However, the design has been validated by a diverse group of experienced engineers from the Aerospace industry, from companies such as JPL and the Aerospace Corporation. This provides external confidence that it will meet the mission requirements.

The first review, a peer design review (PDR), was completed in the very early stages to assess the feasibility of the concept. The reviewers consisted entirely of industry professionals with significant experience, who listened to the project leaders present their initial design goals and schedules. At this time, little was decided about the specific hardware and software architecture of the new avionics system, except for general goals such as consolidating the subsystems. Unfortunately this design review was not a resounding success. A number of the complaints focused on the high risk nature of the CubeSat platform, and didn't address the proposed architecture.

A second, much more informal peer review took place a few months after the first. The participants were just a few engineers from industry, in combination with the entirety of the LightSail-1 team. Many of the design details were developed before the second review, and were discussed thoroughly. The feedback directly altered some of the design for both hardware and software, such as the inclusion of a hard reset of the avionics and adding security mechanisms to the uplinked commands.

The final, critical design review occurred nearly 6 months after the initial PDR. It was executed in a similar fashion, with a number of industry professionals. The avionics system was presented in detail, both the hardware and software designs. Fortunately, this review was much more successful than the PDR and the design was well-received by the reviewers. No significant changes were required as a result of the third review.

VI. Other CubeSat Designs

Nearly 70 CubeSats have been or are manifested to launch. This section attempts to review both the most common designs and the unique ones, to provide a sampling of the overall design space. Some missions keep the design details of their avionics proprietary. Table 4 summarizes the computational power and operating system software flown on some of the CubeSat missions.

A large number of the CubeSat designs are based around multiple PIC-class microcontrollers interconnected with an I^2C bus. This is roughly the same design as PolySat's first generation avionics platform. Most satellites using this design will share the same set of benefits and drawbacks.

The QuakeFinder project [6] designed, built, and operated QuakeSat [7]. Some of their design motivations and outcomes were similar to those presented here. For instance, QuakeSat was built around a general purpose Diamond Systems Prometheus PC/104 board running Red Hat 9 Linux. Mission operators seemed generally pleased with the extra flexibility afforded by the Linux-based solution.

The UWE-1 [8] satellite bus was built around a Hitatchi H8S/2674R microprocessor running μClinux [9]. Their primary mission was to test the performance of various IP-based protocols over the poor quality radio links. Linux was

chosen to leverage the existing protocol implementations and to enable standard server software to work out of the box. This is an early demonstration of the added flexibility and capability provided by flying a full-fledged operating system. The μClinux distribution was chosen to fit within the processing and power limitations of the microprocessor.

Cute-1.7+APDII's [10] main bus consisted of two Hitachi NPD-20JWL PDA logic boards running Windows CE 4.1. Peripherals were interfaced using a combination of USB and RS232. This platform was chosen because the electronics were readily available and the software programming environment was more familiar to a wider group of developers. The latter motivation is a driving factor in PolySat's new avionics design. Due to the use of off the shelf PDAs, Cute-1.7+APDII was a non-standard 3L satellite.

Pumpkin manufactures and sells a CubeSat Kit™ [11]. Their kit provides the ability to customize the main bus via the PC/104 [4] interconnect standard. The main board, and most of the expansion boards, are based around PIC-class embedded processors. Use of the PC/104 standard results in noticeably larger volume requirements for the bus. The CPU daughter boards do not provide nearly the same level of performance that PolySat's integrated design does. The CubeSat Kit™isn't powerful enough to run Linux. Instead it runs a custom real-time operating system, Salvo 4, which presents more of a learning curve for most developers.

Though not a specific avionics design, Klofas *et.al.* [12] provides a survey of CubeSat communication subsystems, including a description of the range of radio hardware and related parameters used in satellites through 2008. It also provides limited insight into three different bus designs and proposes experience-based best practices for communication subsystems. In contrast, this work focuses on the main bus requirements and design, of which communication is one piece.

VII. Future Work

Upon completing the avionics implementation, telemetry data from ground simulated operations and actual flight operation needs to be gathered and analyzed to determine how well the new avionics design is able to meet the mission requirements.

VIII. Conclusion

This work focused on presenting and evaluating two generations of PolySat's avionics technology against a set of general requirements. Deficiencies in the first generation design were identified, and steps were taken in the second generation design to avoid the reoccurrence of the problems. The external design review process for the second generation avionics was used as an initial measurement against specific mission requirements.

This work also presented a short survey of other CubeSat avionics. This background helps illustrate some of the design differences and similarities between PolySat's second generation bus and other avionics platforms. In addition, general design guidance that is widely applicable to many CubeSat missions was presented and justified.

Acknowledgment

We would like to thank Austin Williams who has led the design of the new avionics system. Furthermore, we are especially grateful for Dr. Jordi Puig-Suari's guidance and constantly forward-looking advising to the PolySat program that has made all of this work possible.

References

[1] "Cubesat specifications." [Online]. Available: http://www.cubesat.org/images/developers/cds_rev12.pdf

[2] Christopher Alan Day, "The design of an efficient, elegant, and cubic pico-satellite electronics system," Dec 2004. [Online]. Available: http://polysat.calpoly.edu/PublishedPapers/ChrisDay_thesis.pdf

[3] Keith McCabe, "Enhancements to the cpx i2c bus," Dec 2007. [Online]. Available: http://polysat.calpoly.edu/PublishedPapers/KeithMcCabe_srproj.pdf

[4] PC/104 Consortium, "Pc/104 specifications." [Online]. Available: http://www.pc104.org/pc104_specs.php

[5] A. Gasperin, N. Wrachien, A. Cester, A. Paccagnella, F. Ottogalli, U. Corda, P. Fuochi, and M. Lavalle, "Total ionizing dose effects on 4mbit phase change memory arrays," in *Radiation and Its Effects on Components and Systems, 2007. RADECS 2007. 9th European Conference on*, Sep 2007, pp. 1 –8.

[6] "Quakefinder – earthquake research," http://www.quakefinder.com/.

[7] Tom Bleier and Paul Clarke and Jamie Cutler and Louis DeMartini and Clark Dunson and Scott Flagg and Allen Lorenz and Eric Tapio, "Quakesat lessons learned: Notes from the development of a triple cubesat." [Online]. Available: http://www.quakefinder.com/services/quakesat-ssite/documents/Lessons_Learned_Final.pdf

[8] Y. Aoki, R. Barza, F. Zeiger, B. Herbst, and K. Schilling, "The cubesat project at the university of wurzburg: The mission and system design." [Online]. Available: http://www.stec2005.space.aau.dk/getpdf.php?id=107

[9] "μclinux – embedded linux/microcontroller," http://www.uclinux.org/.

[10] M. Iai, Y. Funaki, H. Yabe, K. Fujiwara, S. Masumoto, T. Usuda, S. Matunaga, J. Katoka, and T. Shima, "A PDA-Controlled Pico-Satellite, Cute-1.7, and its Radiation Protection," in *In Proceedings of the 18th AIAA/USU Conference on Small Satellites*, Aug 2004.

[11] "Pumpkin cubesat kit," http://www.cubesatkit.com/.

[12] Bryan Klofas and Jason Anderson and Kyle Leveque, "A Survey of CubeSat Communication Subsystems," in *Proc. of CubeSat Developers' Workshop*, Apr 2008.