The Use of SPARK in a Complex Spacecraft

Carl Brandon Peter Chapin

Vermont Technical College

{cbrandon,pchapin}@vtc.vsc.edu

Abstract

Building on our previous experience of constructing and operating a successful Earth orbiting satellite, in which the software was primarily written in SPARK, we now describe our new project: writing the flight software for a complex, lunar orbiting spacecraft called Lunar IceCube (LIC). We continue with our use of SPARK for this new mission, extending and enhancing the techniques used in our previous mission. Although this work is ongoing, we have found SPARK to be a manageable technology for us, even in our student-centered development environment.

Categories and Subject Descriptors J.2 [*Physical Sciences and Engineering*]: –-Aerospace; C.3 [*Special-Purpose and Application-Based Systems*]: –-Real-time and embedded systems; D.2.4 [*Software Engineering*]: Software/Program Verification—Programming by contract

Keywords SPARK, Flight Software, CubeSat

1. Introduction

The Vermont Technical College CubeSat Laboratory's first CubeSat, Vermont Lunar CubeSat, was a 1U (10 cm × 10 cm × 10 cm, 1 kg) spacecraft. It was launched on a Minotaur 1 rocket as part of the NASA ELaNa IV, Air Force ORS-3 launch from Wallops Island on November 19, 2013. It was launched into a 500 km altitude, 40.5° inclination Earth orbit (Brandon and Chapin 2013).

Vermont Lunar CubeSat was one of 12 university CubeSats on this launch. Eight of them were never heard from, two had partial contact for less than one week and one worked for four months. Vermont Lunar CubeSat remained functional until reentry, after two years and two days, on November 21, 2015. Its flight software was written, primarily by undergraduate students, in SPARK 2005 while the other CubeSats on the launch were programmed in C. We believe that the use of SPARK allowed our very small software development team to complete the software on time and without fatal flaws. It is believed in the CubeSat community (as related by one of the CubeSat founders, Jordi Puig-Suari of Cal Poly) that most CubeSat failures are software related. The flight software is the hardest part of building a small spacecraft. Every part of the software is mission critical, and the failure of any part will result in a dead spacecraft. While SPARK has been used in commercial and military aircraft, air traffic control and high speed trains, we are the first to use it in space. We believe this resulted in the software successfully accomplishing its purpose during our two year mission. We received much inertial measurement data, several dozen nice photos of the earth, as well as other data.

Because of the success of our mission, attributed to our superior software technology, we were chosen to write the flight software for Lunar IceCube, a 6U CubeSat (10 cm × 20 cm × 30 cm, 14 kg) with an iodine propellant ion drive propulsion system. This spacecraft is manifested on NASA's Space Launch System (SLS) EM-1 flight to the Moon in 2018. This mission is much more complicated. Our first CubeSat was a $50,000 project with a launch valued at $125,000. The development team consisted of two faculty and around three students at any one time. Lunar IceCube is a collaboration of Vermont Technical College with Morehead State University (KY); NASAs Jet Propulsion Laboratory; NASAs Deep Space Network (DSN); NASAs Goddard Space Flight Center, and Busek, Inc. The spacecraft budget is around $15,000,000 and the launch is valued at $10,000,000.

The primary mission of the SLS is to test the Orion capsule on a circum-lunar flight with a high speed reentry at the earth. It is the most powerful rocket ever built, with about 2,000,000 pounds more thrust than the Saturn V. In addition to the Orion capsule, it will carry 13 6U CubeSats. Lunar IceCube will be deployed shortly after trans-lunar injection, on a sun centric orbit. The software will have to control the de-tumbling of the spacecraft so the photovoltaic panels can be oriented toward the sun before the limited battery power runs out (about 30 minutes). Following de-tumbling, the flight software will periodically make radio contact with the DSN for purposes of navigation, and it will receive and process commands transmitted from Earth to guide the spacecraft into lunar orbit. Once in orbit, the science mission, mapping the moon's water vapor, ice and other volatiles over a six month period, will be under the control of the flight software. Commands for the science orbits will be uploaded, and the software must be able to autonomously control the spacecraft for up to 24 hours (3-4 lunar orbits) if necessary, due to timing issues of DSN availability.

2. Spacecraft Description

The physical spacecraft, including both the mechanical and electronic hardware, is being constructed by our collaborators at Morehead State University. The systems controlled by the flight software include the following:

- A photovoltaic (PV) panel orientation drive for aiming the panels (eight panels totaling 120 W) toward the sun, and a power supply for battery charging control and for the generation of multiple voltage levels.
- The science instrument, the Broadband Infrared Compact High Resolution Exploration Spectrometer (BIRCHES), developed by NASA's Goddard Space Flight Center. The instrument must be commanded when to take data and how to set the iris opening so as the ground pixel size will remain constant as the distance

HILT '16 October 6–7, 2016, Pittsburgh, Pennsylvania, USA

ACM [to be supplied]. . . $15.00

to the Moon changes. Data and telemetry from the instrument will need to be received and buffered for later transmitting to the DSN.

- A Blue Canyon attitude determination and control system (ADACS) which has a star tracker camera and three momentum wheels for rotating the spacecraft. During data acquisition, the ADACS will be commanded so that BIRCHES points directly toward the lunar surface. After acquisition, the ADACS will have to be commanded to point the patch antennas toward the DSN dishes on the earth.
- An Iris-2 X-band radio developed by the NASA Jet Propulsion Laboratory which will be used for data transmission to earth and for uploading commands and software updates to the spacecraft. It is also used as a navigation transponder with round trip signal time from the earth and back for distance measurement and Doppler shift of the signal for velocity measurement. Besides controlling the various Iris-2 modes, the flight software will be responsible for forming and decoding the DSN data format packets.
- The Busek BIT-3 iodine propellant ion drive will be controlled as to thrust level (up to about 1.2 mN) and modes. The ion drive will operate for months at a time, requiring about six months just to enter lunar orbit. It is mounted on a two axis gimbal which will be controlled by the flight software so the thrust is pointed in the desired direction. This is coordinated with rotation of the spacecraft by the ADACS. This is the first use of an iodine propellant ion drive.

The flight software will run on a Space Micro Proton-400 dual-core PowerPC, radiation hardened CPU board (Space Micro 2014). We are using VxWorks 6.8 as our embedded, real time operating system (Wind River). VxWorks was used for the Mars Science Laboratory surface operations as well as the very complex Mars atmospheric entry and landing.

3. Tools

The flight software itself is written, to the greatest extent possible, in a restricted dialect of Ada called SPARK. In addition to being a language that is amenable to analysis, SPARK is also a set of tools that can be used to statically prove programs free of certain classes of runtime error. The SPARK language and tools, hereafter simply called "SPARK," can also be used to prove that programs adhere to their specifications as described by embedded pre- and postconditions and other assertions added by the developer. Additional information about SPARK can be found in (McCormick and Chapin 2015) or online (SPARK Team 2014a,b).

We are using the newer SPARK 2014 rather than the older SPARK 2005 that we used during our work on the Vermont Lunar CubeSat project. In addition to supporting a much richer subset of Ada than the earlier SPARK, the current generation of SPARK adds support for Ravenscar-style Ada tasking, a very useful feature for our system as we describe in Section 4.

We also avoid C whenever possible despite the ubiquitous application of that language in spacecraft flight software (see, for example, (Holzmann 2014)). Our verification goals, described in more detail in Section 5, are to at least show the software free of runtime error, and that task is much more difficult with C. We have not investigated the use of any C verification tools such as, for example, Frama-C (Cuoq et al. 2012), although doing so might be an interesting avenue for future work should it become necessary to include significant C components in the Lunar IceCube system.

The software is being written primarily by undergraduate and masters degree students at Vermont Technical College, with some assistance from faculty, and students at Morehead State University. The students are inexperienced developers and most have never used SPARK, or even Ada, until becoming a part of this project. Although some training time is required, it is remarkable how quickly the students can become productive with the SPARK language and verification tools.

4. Software Architecture

In order to simplify the flight software and reduce the testing burden, most flight decisions for the Lunar IceCube mission will be made by controllers on the ground. The core of the flight software thus consists of a command scheduler that plays a script of uploaded commands at appropriate times. The results of these commands, in the form of telemetry data, science data from the BIRCHES instrument, or error responses are buffered for transmission to Earth when the Deep Space Network is available. The flight software has minimal autonomy except for immediately after deployment where it must stabilize and orient the spacecraft in order to establish reliable solar power and communications.

To date the project has suffered from relatively poorly defined software requirements. Finding a remedy for this issue is ongoing, but is complicated by the large number of geographically and politically dispersed collaborators working on the various subsystems. Most of our collaborators are hardware and spacecraft design experts and not software engineers, making communication about software requirements and other software related issues more challenging. In keeping with our previous experience with the Vermont Lunar CubeSat, the use of SPARK has helped us maintain logical coherence of our software even as it is massively refactored in response to our deepening understanding of its requirements. The rigors imposed by SPARK are thus of great value in an environment that is otherwise not very rigorous, at least from a software development point of view.

Our design is relatively standard. We divide the system into a collection of *modules* together with a support library. Each module and the support library consists of one or more SPARK packages. Modules are distinguished from ordinary library packages in that each module has its own library level task running in an infinite loop. Thus modules are active entities whereas the support library is passive.

Modules communicate in a tightly constrained manner using Ada protected objects. Each module has a single protected object that it uses as a mailbox holding pending messages for the module. Messages are sent by other modules, or even by a module to itself, by calling a protected procedure in the mailbox. A module receives messages by calling a single entry in the mailbox. Some modules (but not all) can meaningfully receive several different types of messages. Since SPARK requires that a protected object have only a single entry, the messages are pulled from the mailbox in that case as XDR encoded (Eisler 2006) octet arrays. Internally the module decodes the message in the octet array at run time before acting on it.

The need for some modules to decode messages adds runtime overhead to message processing, but that overhead is generally small. Also many modules do not require it. Of somewhat greater concern, given our verification goals, is the reduction in type safety introduced by this design. Modules which must decode messages must detect and handle malformed messages at runtime. However, we hope to prove in the final system that malformed messages are never created. If we are successful, we can then justify removing our manually programmed runtime checks on the format of messages. This parallels the normal work flow of SPARK development where proof of freedom from runtime error justifies the removal of compiler generated checks.

Library packages are shared by multiple modules and provide services of general interest to the system. For example, one such library package assists with XDR encoding and decoding of messages. In some cases there is a choice of making a service available as a library package or as a module. The procedures providing the API of the service could be visible procedures of a library package in one case, or protected procedures of a supporting module's mailbox in the other. However, the semantics of a module vs. a library package are quite different due to the highly asynchronous nature of modules. Results from a module are generally returned via a reply message that interleaves with the arrival of messages from other sources.

The core module of our system contains a scheduler component that consists of a task that periodically pools a database of pending commands. When a command comes due it is executed by sending a message to an appropriate supporting module. New commands are added to the database as part of command processing, but most notably by the module communicating with the DSN. Commands transmitted from Earth are the primary driver of the spacecraft's activity.

The science data gathered by the spacecraft will be transferred to the ground over the Deep Space Network using the Consultative Committee for Space Data Systems (CCSDS) File Delivery Protocol (CFDP). This is a general purpose file transfer protocol designed for use with very high latency, unreliable space links. A full CFDP implementation has many features that are not relevant to the Lunar IceCube mission. However, we are creating a partial implementation of CFDP, entirely in SPARK, to support our mission needs. We intend to make our implementation available to other groups as open source software in the hope that it may promote the use of SPARK in other space flight software systems.

As is traditional in most large scale space missions, but unusual for CubeSat missions, we are designing a system where it will be possible to upload software updates to the spacecraft in flight. However, this has been deemed a high risk operation so it is hoped that we never have to actually make such an update. The value of SPARK in this environment is obvious: it is considered a priority that the software work correctly at the time of launch so that updates will be, ideally, unnecessary.

5. Verification and Testing

Our verification goals consist primarily of ensuring the software is free from runtime errors in the sense usually meant by SPARK: no Ada language defined exceptions will ever be raised due to out-of-bounds array access, arithmetic overflow, division by zero, or any other constraint violation. We will also use SPARK to prove as many higher level correctness properties as time and resources allow. We are particularly interested in showing that no invalid messages are ever sent to a module.

In addition to the SPARK verification tools, we execute unit tests using the AUnit test framework. All of our source code is compiled, tested, *and proved* daily in a Jenkins continuous integration (CI) server. This allows us to easily monitor the state of the system and detect failing tests or failing proofs quickly. Our policy is to avoid committing changes that cause unit test failures. However, we do accept failing proofs on a day-to-day basis; doing otherwise presents too great an acceptance barrier for "simple" modifications.

However, the CI server often detects proofs that start failing unexpectedly in an apparently unrelated part of the system from where a small change was made. A good example is when the addition of a precondition on a subprogram causes proof problems at some of that subprogram's call sites. It is much easier to understand the cause of those problems and to correct them when they are detected early. We strongly recommend the use of *continuous proof* in SPARK development environments.

We are using Microsoft Windows as our development platform. We compile our source natively for the sake of executing unit tests. We also compile our source with a cross compiler that generates PowerPC code for VxWorks. The PowerPC version of our system is then loaded onto a development board that is similar to the flight computer (the main difference is the lack of radiation hardening and more convenient physical interfaces). This allows us to run tests of the system in a more realistic context.

We are making use of a NASA Jet Propulsion Laboratory provided hardware emulator for the Iris-2 radio. One side of the emulated radio is connected to the development board. The other side is connected to a Linux system running NASA provided software to emulate the DSN network. This allows us to send simulated radio traffic to the radio emulator and for our flight software to interact with the radio emulator, and indirectly with the emulated DSN. Finally, we are using simulation software provided by NASA that will do the physics calculations necessary to create simulated flight conditions.

At the time of this writing we have not completely configured this test bed, but we do have various components of it working now and anticipate having it fully operational in a few weeks. With this combination of hardware and software we are able to run our flight software through full scale mission simulations to explore its behavior in a variety of scenarios. The use of SPARK will not decrease the rigor of this testing, but it will, hopefully, save time by eliminating the need to find and fix low level errors, such as out-of-bounds array accesses. It will also help us direct the testing effort to areas where it is most needed by exploring the sections of the software where the SPARK proofs are still failing.

6. Conclusion

Hopefully, a successful Lunar IceCube mission will be further proof after our successful Vermont Lunar CubeSat of the value of SPARK to both the CubeSat community and also the general space community. There have been many failures of both CubeSat missions (83% failure rate on our ELaNa IV launch) and much larger and expensive spacecraft. We hope that the methods we have used, and are using, and will make available to the CubeSat and general space community, will help bring this failure rate down. Spacecraft software is much more difficult than many of the groups attempting spacecraft assume. After all, it *is* rocket science.

References

C. Brandon and P. Chapin. *A SPARK/Ada CubeSat Control Program*, pages 51–64. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013. ISBN 978-3-642-38601-5. doi: 10.1007/978-3-642-38601-5_4. URL http://dx.doi.org/10.1007/978-3-642-38601-5_4.

P. Cuoq, F. Kirchner, N. Kosmatov, V. Prevosto, J. Signoles, and B. Yakobowski. Frama-C: A software analysis perspective. In *Proceedings of the 10th International Conference on Software Engineering and Formal Methods*, SEFM'12, pages 233–247, Berlin, Heidelberg, 2012. Springer-Verlag. ISBN 978-3-642-33825-0. doi: 10.1007/978-3-642-33826-7_16. URL http://dx.doi.org/10.1007/978-3-642-33826-7_16.

M. Eisler. *RFC-4506: XDR: External Data Representation Standard*. Internet Engineering Task Force, May 2006. http://tools.ietf.org/html/rfc4506.html.

G. J. Holzmann. Mars code. *Commun. ACM*, 57(2):64–73, Feb. 2014. ISSN 0001-0782. doi: 10.1145/2560217.2560218. URL http://doi.acm.org/10.1145/2560217.2560218.

J. W. McCormick and P. C. Chapin. *Building High Integrity Applications with SPARK*. Cambridge University Press, 2015.

Space Micro. Proton 400k single board computer. http://www.spacemicro.com/assets/datasheets/digital/slices/proton400k.pdf, May 2014. Accessed: 2016-09-15.

SPARK Team. *SPARK 2014 Reference Manual*. AdaCore, New York and Paris, 2014a. URL http://docs.adacore.com/

`spark2014-docs/html/lrm/`. Available at `http://docs.adacore.com/spark2014-docs/html/lrm/`.

SPARK Team. *SPARK 2014 Toolset User's Guide*. AdaCore, New York and Paris, 2014b. URL `http://docs.adacore.com/spark2014-docs/html/ug/`. Available at `http://docs.adacore.com/spark2014-docs/html/ug/`.

Wind River. Vxworks. `https://windriver.com/products/vxworks/`. Accessed: 2016-09-15.