On the Feasibility of Automatically Detecting and Recovering from SEUs in Cyber-Physical Space Systems

Robert G. Pettit IV
The Aerospace Corporation
Chantilly, Virginia, USA
rob.pettit@aero.org

Aedan D. Pettit*
The College of Wooster
Wooster, Ohio, USA
apettit20@wooster.edu
aedan.pettit@aero.org

ABSTRACT

The past decade has seen explosive growth in the use of small satellites. As these small, typically short-lived and risk-tolerant platforms are increasingly adopted for spaceflight missions, there has been a growing trend to place more responsibility on the flight software (versus hardware) and an increasing adoption of consumer-grade microprocessors to satisfy this desire for increased processing capability while still minimizing size, weight, and power parameters. These consumer-grade processors, however, are more susceptible to cosmic radiation and the occurrence of single event upsets (SEUs). In this paper, we present an initial exploration into the feasibility of implementing automated detection and recovery mechanisms to mitigate SEUs within these cyber-physical spaceflight systems.

CCS CONCEPTS

• **Computer systems organization → Dependable and fault-tolerant systems and networks**; • **Hardware → Fault tolerance**; • **Software and its engineering → Embedded software**; **Software fault tolerance**;

KEYWORDS

cyber-physical systems, flight software, single event upset

ACM Reference Format:
Robert G. Pettit IV and Aedan D. Pettit. 2018. On the Feasibility of Automatically Detecting and Recovering from SEUs in Cyber-Physical Space Systems. In *SEsCPS'18: SEsCPS'18:IEEE/ACM 4th International Workshopon Software Engineering for Smart Cyber-Physical Systems , May 27-June 3,2018, Gothenburg, Sweden.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3196478.3196487

1 INTRODUCTION

The past decade has seen explosive growth in the use of small satellites. Industry technology forecasts [3, 9, 10, 18] show no slowing of this trend and, in fact, predict small satellites will continue this growth for the foreseeable future. In particularly the subclass of micro satellites (small satellites less than 50kg in mass) is expected to see substantial growth as shown in Figure 1.

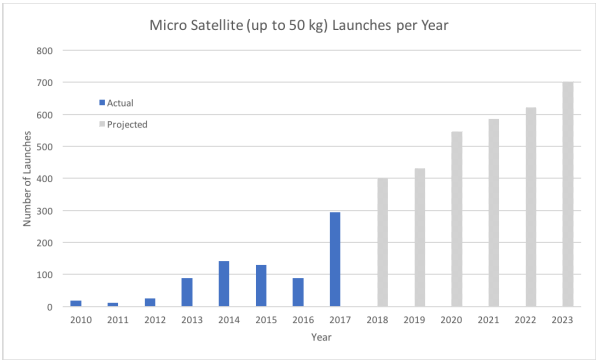

Figure 1: Predicted growth in micro satellite launches.

To achieve the desired on-board computer processing capability while still minimizing size, weight, and power (SWaP) parameters, many of these micro satellite programs are adopting consumer-grade processors rather than the traditional radiation-hardened (but slower and more costly) processors.

While these micro satellite missions tend to be more risk-tolerant than a traditional satellite, frequent/complete data corruption or failure of the on-board processor is still unacceptable. The risk of adopting these unshielded and smaller consumer-grade processors is that they are more susceptible to cosmic radiation and the occurrence of single event upsets.

A single event upset (SEU) is induced by heavy ion radiation [1]. Satellites are constantly exposed to heavy ion cosmic radiation, both from solar and intergalactic sources. This radiation can corrupt on-board memory as well as affecting the flight processor itself. Most commonly, SEUs are manifested as "bit flips" in which the value of a single bit is reversed from 0 to 1 or vice versa. This can trigger anomalies in the flight software in which instructions can be executed out of order; incorrect commands can be transmitted; data can be corrupted; or a processor shutdown/reboot that could lead to mission outage or failure.

Traditionally, satellites employed slower, radiation-hardened processors. With the balance of mission processing being performed in hardware versus software, processing power was not an issue. These satellites were also orders of magnitude larger than micro satellites and were extremely costly; long-lived; and very much risk averse. In contrast, micro satellites are small, comparatively inexpensive platforms with shorter expected lifespans and greater

*Also with The Aerospace Corporation.

SEsCPS'18, May 27-June 3,2018, Gothenburg, Sweden

ACM ISBN 978-1-4503-5728-9/18/05…$15.00
https://doi.org/10.1145/3196478.3196487

tolerance for risk. Due to their size, they also tend to place greater mission processing responsibility on the software. Then, to achieve the greater processing power while minimizing SWaP, many micro satellites are turning to commercial microprocessors such as the Arm Cortex-A series. Figure 2 illustrates some of the differences between a traditional radiation-hardened processors (BAE's RAD750®) [20] and a commercial/consumer grade Arm Cortex-A series [11].

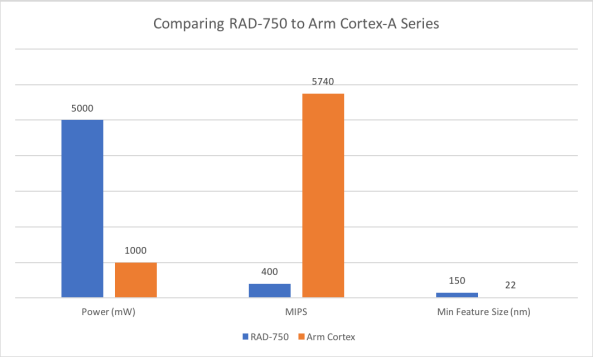

Figure 2: Comparison between traditional RAD750® PowerPC and Arm Cortex-A series processors

From Figure 2, we see that the Arm processor's power consumption is 1/5 of the RAD750® while being capable of delivering more than one order of magnitude greater processing performance. However, the significantly smaller minimum feature size along with the lack of shielding and low critical charge (the amount of charge needed to change the state of a memory location) place the Arm processor at a much greater risk for SEUs induced by cosmic radiation while in orbit.

In a typical satellite system, these processors are integrated into an overall single board computer (SBC). While much attention has been given to the radiation effects and potential for SEUs in other electronic components (e.g. on-board FPGAs), relatively little work has focused on the SEU risk associated with the embedded software or the feasibility of software-centric automated detection and recovery from SEUs. This paper represents our initial research efforts to specifically investigate options for automated detection and recovery from SEUs encountered by micro satellite systems employing modern, consumer-grade microprocessors such as the Arm Cortex series.

1.1 Related Works

The theory of SEUs affecting space-borne electronics was first suggested by Walmark and Marcus in 1962[22]. For the modern era of microprocessors, Akers [1] provides and over on the vulnerability of these processors to SEUs. Specific examples of SEU occurrences with flight systems can be found in O'Neil [15], Bedingfield [2], Underwood [21], and Swift [19].

Current trends on the increasing application of micro satellites can be found in the studies conducted by the Institute for Defense Analysis [10], SpaceWorks [3], and the Nanosatellite and CubeSat Database [9]. These are updated at least annually.

For detecting SEU's affecting software applications, Nicolescu [13, 14] primarily addressed data errors and logic flow. Goloubeva [5] goes into more depth with logic flow and describes a method of using flow assertions to protect against SEUs. In our previous work [16], we applied a combination of data and logic flow checks to determine how SEUs can be detected under ground testing (using electromagnetic pulses) rather than ion radiation). In this paper, we follow on to our previous detection work and examine the feasibility of implementing smart detection and recovery algorithms within the embedded flight software.

2 DETECTING SINGLE EVENT UPSETS

In our previous work [16], we implemented checks for basic instruction errors, control errors, and data integrity. We also implemented a heartbeat signal that would still allow us to detect a SEU even if nothing was triggered in the previous checks. The code was implemented on a Raspberry Pi Model 3 [17] as the Arm Cortex-A53 is representative of those being considered for current space programs [3, 9, 10]. The Raspberry Pi also allowed for a very inexpensive test platform (approximately $30USD) so there was no great cost risk if any units were permanently damaged during test.

To check for memory errors, we used multiple threads implementing a function to compute ten factorial (10!) using iteration. These threads operated in an infinite loop after startup and the results were compared to the known value (3,628,800) after each iteration. The number of threads was adjusted to maximize memory utilization while still allowing the functions to complete in a timely manner. Ultimately, 75 threads were chosen for our final test runs.

To check for basic instruction errors (e.g. executing instructions out of order), we implemented a series of block checks within the main program. Within the main continuous loop, the code was divided into several blocks. At the start and end of each a block, a checking function was called to see if the block had the correct number and state. This ensured that the correct number of blocks of code had been run and that each block had been run completely.

```
//block checking function
void start_block(int block_number,
        char *time_string,
        int *error_count,
        int *r_error_count){

        if(block_number != block_counter){
                //Write error to file...
                (*error_count)++;
                (*r_error_count)++;
        }
        if(status_block != 1){
                //Write error to file...
                (*error_count)++;
                (*r_error_count)++;
        }
        status_block = 0;
}
```

Similar to the block checks for sequential code, we also implemented control checks to guard against SEUs affecting conditional branches. This class of errors can be split into two parts: conditional and unconditional. Conditional control instruction errors occur when the SEU changes a conditional expression in the control code - e.g. a change in the condition within an if statement. Unconditional control instruction errors occur when the SEU changes another part of the control code - e.g. skipping a function call. Detecting errors in the conditional instructions was done by performing each condition twice. If the condition had the same outcome both times then there was no error. If there was a different outcome then an error must have occurred. To detect unconditional control errors, we created a variable that was initialized to one (1), and passed a pointer to that variable to each function that was called. Each function contained code to change the variable to two(2) and this was checked after each function call to verify the function actually completed as designed.

```
//checking conditionals prototype
bool check_ctrl();

//usage
if(check_ctrl()){
        //Some code...
}
else{
        //Write error to file...
        //Some code...
}

//checking unconditionals
ctrl_branch = 1;

//implementing unconditional check
some_function(some args, &ctrl_branch){
        (*ctrl_branch) = 2;
        //Some code...
}

if(ctrl_branch != 2){
        //Write error to file...
}
```

2.1 Testing

Our test system used a Raspberry Pi 3 with a four-core Arm Cortex-A53 processor running at 1.2GHz and configured with 1GB of low-power RAM (LPDDR2). For the operating system, we used a Yocto [4] embedded Linux distribution as a more representative OS than the native Raspberry Pi distribution. We first verified that the software would catch data and processor errors by manually changing values and forcing erroneous jumps in a debugger. Once this was confirmed, we executed our test software with the Raspberry Pi SBC placed adjacent to the coils of an electromagnetic pulse (EMP) generator as showing in Figure 3.

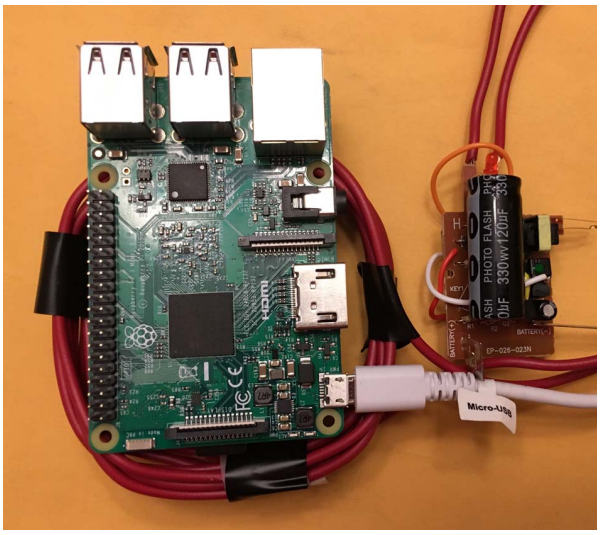

Figure 3: Fry Pi test configuration

Trial and error led us to use 75 threads for the factorial computation. This allowed for the greatest amount of RAM utilization while still allowing for sufficient CPU availability that the threads would continue to complete the factorial calculations without significant delay. Over the course of 100 tests, we only encountered two data errors. No control flow errors were encountered with the processor. However, heartbeat pauses were detected with test, indicating that the system did momentarily pause execution but also continue in the correct order after that pause (typically only one skipped heartbeat). Depending on the application, these skipped heartbeats (<2s) may or may not represent a significant risk.

3 RECOVERY OPTIONS

Recovery options can be divided into two areas: data recovery and processor recovery. We will address thoughts on each of these in the following subsections.

3.1 Data Recovery

Data recovery is the least difficult of the recovery options. While hardware solutions could employ error checking and correction (ECC) memory, this would require additional space on the SBC for the extra memory chips as well as requiring some additional power. From our tests, it appears that software error checking is a feasible approach. In this case, additional threads/processes can be added to check and correct critical computations. As opposed to the 75 concurrent threads that we used, typical spaceflight software employs no more than 10 concurrently executing real-time processes. With the multiple cores and increased processing speed of microprocessors such as the Arm Cortex family in comparison to the traditional radiation-hardened processors, this can be accomplished without overly taxing the processor or memory utilization.

3.2 Processor Recovery

Automated processor and SBC recovery is a more difficult task. While we verified in a debugger that our code should catch flow control errors, we only experience momentary heartbeat lapses under EMP testing. However, if we look at the scalability of a software detection and recovery approach by augmenting a single program, it is questionably feasible. To explore a reasonable example of flight software, we analyzed the NASA core Flight System (cFS) [12]. The cFS is a ready-to-use framework for spaceflight software and includes a sample payload application. For that payload software alone, there are over 5,000 functions written in C/C++. Of those, most are relatively simple with an average cyclometric complexity (V(g)) of 3.5. However, expanding that through all the functions would lead to a V(g) > 18,000, meaning that there are over 18K independent paths that would need to be instrumented. This is certainly infeasible for manually implementing the software checks and would at least need to be automated. Yet even if the code was automatically instrumented, the increased lines of executable code would likely affect processing speed and timing between concurrent tasks.

A more feasible approach appears to be one in which redundant SBCs are used. With the low power of an Arm processor compared to the PowerPC (see Figure 2), Arm-based SBCs could conceivably be used to implement *n*-mode redundancy and still remain well below the power consumption of traditional radiation-hardened SBCs. In this case, legacy software can still be used but with the addition of independent tasks that are responsible for error checking and recovery.

Traditional satellite systems tend to use a dual-mode, fail-over approach in which the active computer is rebooted while a secondary computer is started to take over operations. While such an approach is feasible for our class of small satellites, it does involve loss of mission time and does not capitalize on the efficiencies in SWaP gained from employing modern microprocessors. Taking advantage of an Arm-based (or comparable) SBC, we feel that it is a better approach to consider an active *n*-mode redundancy scheme in which data is processed by *n* active SBCs simultaneously with final results being determined by a voting scheme. Active triple-mode redundancy could be achieved with an Arm-based SBC while still only consuming 60% of the power required by a single PowerPC-based SBC.

4 CONCLUSIONS AND FUTURE WORK

In this paper we present our initial investigation into the smart detection and recovery from single event upsets in micro satellites adopting modern consumer-grade processors. It is our position that smart data recovery is certainly feasible within the software for relatively little cost in terms of resource utilization. Processor recovery appears to be feasible with redundant hardware and can still be achieved with a lower SWaP profile than traditional radiation-hardened SBCs. However, this trade-off certainly requires more development cost/effort.

Moving forward, we plan to test whether our EMP testing is in fact representative of the SEU risks in space. To achieve this, we would subject our Raspberry Pi to ground radiation testing representative of the space environment such as described by Irom [7] and Koga [8]. We then plan to conduct further experiments with multiple SBC configurations both with the Raspberry Pi units as well as other commodity processors.

Finally, we also plan to investigate protective measure in hardware casing as well as orientation of the SBCs. Preliminary testing with the EMP testbed indicates that aluminum foil may give sufficient protection to the point that a single SBC can be used in all but the most critical applications. However, the EMP disruption is from magnetic waves versus energized particles, so this approach would need to be verified under radiation tests. Additionally, we plan to test whether altering the orientation of redundant SBCs within the satellite can increase the resilience to directional particles encountered in space.

REFERENCES

[1] LD Akers. 1996. Microprocessor Technology and Single Event Upset Susceptibility. (1996).

[2] Keith Bedingfield, Richard D Leach, Margaret B Alexander, et al. 1996. Spacecraft system failures and anomalies attributed to the natural space environment. (1996).

[3] B Doncaster, C Williams, and J Shulman. 2017. SpaceWorks Nano and Microsatellite Market Forecast 2017. *SpaceWorks Enterprises, Inc.(SEI), Atlanta, GA* (2017).

[4] Linux Foundation. [n. d.]. Yocto Project. ([n. d.]). https://www.yoctoproject.org

[5] Olga Goloubeva, Maurizio Rebaudengo, M Sonza Reorda, and Massimo Violante. 2003. Soft-error detection using control flow assertions. In *Defect and Fault Tolerance in VLSI Systems, 2003. Proceedings. 18th IEEE International Symposium on*. IEEE, 581–588.

[6] David Honess and Oliver Quinlan. 2017. Astro Pi: Running Your Code Aboard the International Space Station. *Acta Astronautica* 138 (2017), 43–52.

[7] Farokh Irom. [n. d.]. *Guildeline for Ground Radiation Testing of Microprocessors in the Space Radiation Environment.* Technical Report.

[8] R Koga, WA Kolasinski, MT Marra, and WA Hanna. 1985. Techniques of Microprocessor Testing and SEU-rate prediction. *IEEE Transactions on Nuclear Science* 32, 6 (1985), 4219–4224.

[9] Erik Kulu. 2018. Nanosatellite Database. (2018). http://www.nanosats.eu

[10] Bhavya Lal, Elena de la Rosa Blanco, Jonathan R. Behrens, Benjamin A. Corbin, E.K. Green, Alyssa J. Picard, and Asha Balakdrishnan. 2017. *Global Trends in Small Satellites.* Technical Report P-8638. Institute for Defense Analysis.

[11] Arm Limited. 2017. Arm Cortex-A. (2017). https://developer.arm.com/products/processors/cortex-a

[12] NASA. [n. d.]. core Flight System (cFS). ([n. d.]). https://cfs.gsfc.nasa.gov

[13] Bogdan Nicolescu, Yvon Savaria, and Raoul Velazco. 2004. Software detection mechanisms providing full coverage against single bit-flip faults. *IEEE Transactions on Nuclear science* 51, 6 (2004), 3510–3518.

[14] Bogdan Nicolescu and Raoul Velazco. 2003. Detecting soft errors by a purely software approach: method, tools and experimental results. In *Embedded Software for SoC*. Springer, 39–51.

[15] PM O'Neill and GD Badhwar. 1994. Single event upsets for space shuttle flights of new general purpose computer memory devices. *IEEE Transactions on nuclear science* 41, 5 (1994), 1755–1764.

[16] Robert G Pettit IV and Aedan D Pettit. 2018. Detecting Single Event Upsets in Embedded Software. In *Real-Time Computing, 2018. ISORC'07. 21st IEEE International Symposium on*. IEEE.

[17] Raspberry Pi. 2016. Raspberry Pi 3 Specs. (2016). https://www.raspberrypi.org/magpi/raspberry-pi-3-specs-benchmarks/

[18] Armen Poghosyan and Alessandro Golkar. 2017. CubeSat evolution: Analyzing CubeSat capabilities for conducting science missions. *Progress in Aerospace Sciences* 88 (2017), 59–83.

[19] Gary M Swift and Steven M Guertin. 2000. In-flight observations of multiple-bit upset in DRAMs. *IEEE Transactions on Nuclear Science* 47, 6 (2000), 2386–2391.

[20] BAE Systems. 2016. RAD750™radiation-hardened PowerPC microprocessor. (2016). https://www.baesystems.com/en-us/download-en-us/.../1434555668211.pdf

[21] CI Underwood and MK Oldfield. 1998. Observed radiation-induced degradation of commercial-off-the-shelf (COTS) devices operating in low-earth orbit. *IEEE Transactions on Nuclear Science* 45, 6 (1998), 2737–2744.

[22] IT Walmark and SM Marcus. 1962. Minimum Size and Maximum Packing Density of Non-Redundant Semi-Conductor Devices. *Proceedings of the IRE* (1962).