Dynamically Reconfigurable Coprocessor for Floating-Point Arithmetic Capability in Small Satellites

Hezekiah Austin, Chris Major, Zach Becker, Tristan Running Crane, Kris Allick, Brock J. LaMeres

Dynamically Reconfigurable Coprocessor for Floating-Point Arithmetic Capability in Small Satellites

Hezekiah A. Austin
Montana State University
324 Norm Asbjornson Hall
Bozeman, MT 59715
hezekiah.austin@student.montana.edu

Chris Major
Resilient Computing
2952 Technology Blvd W.
Bozeman, MT 59718
chris.major@resilient-computing.com

Zach Becker
Montana State University
324 Norm Asbjornson Hall
Bozeman, MT 59715
bitbytebitco@gmail.com

Tristan Running Crane
Montana State University
324 Norm Asbjornson Hall
Bozeman, MT 59715
tristanrunningcrane@gmail.com

Kris Allick
Montana State University
324 Norm Asbjornson Hall
Bozeman, MT 59715
kris.allick@gmail.com

Brock J. LaMeres
Montana State Univerity
316 Norm Asbjornson Hall
Bozeman, MT, 59717
lameres@montana.edu

***Abstract*— Field Programmable Gate Arrays (FPGAs) are increasing used in small satellite missions for tasks ranging from command & data handling to sensor data processing. An FPGA is a dense system of computing resources, logic gates, memory, look-up tables (LUTs), etc. The size, mass, input/output (I/O), and low power constraints of small satellites prevent designers from taking advantage of the full capability of FPGAs. Implemented yet unused components waste power and FPGA resources. This paper proposes increasing FPGA-based computer capability by implementing a dynamically reconfigurable coprocessor in parallel to the main processor on the FPGA. The coprocessor functions as a hardware accelerator for data intensive operations or operations unsupported by the main processor. This approach minimizes the coprocessor's footprint by reconfiguring the coprocessor in-real time to reuse the same FPGA resources for different stages of a computation. Complex computational operations are broken up into multiple discrete stages with individual coprocessors sequentially performing each stage. This allows the required FPGA resources for the coprocessor to be minimized while taking advantage of the computational boost in FPGA hardware instead of the main processor. A secondary benefit to is this approach is that the coprocessors can support functionality not provided by the main processor. To investigate the feasibility of this approach, a set of floating-point operation were implemented as coprocessors and integrated into a RISC-V soft processor system. The results of this proof-of-concept provide evidence that the use of dynamically reconfigurable floating-point coprocessors has the ability to increase the computational capability of small satellite computers while fitting within the constrained resources of such missions.**

Table of Contents

1. Introduction 1
2. Research 3
3. Conclusions 6
Acknowledgments 6
References 7
Biography 8

1. Introduction

Dynamic partial reconfiguration uses the flexibility of a Field Programmable Gate Array (FPGA) to swap coprocessors during real-time operation while maintaining the functionality of the rest of implemented system. The objective of this research is to improve the capability and flexibility of small satellites without requiring either the size or power to be greatly increased. Research scope and the related fields are shown the research scope Venn diagram in Figure 1. This approach has advantages for small satellites with FPGA-based computers were mass ($< 4kg$), size ($< 3U$), and power ($< 2.5W$) are highly limited resources. Dynamic partial reconfiguration enhances an FPGA-based system with in-flight reprogrammable hardware and real-time adaptive functionality. [1].

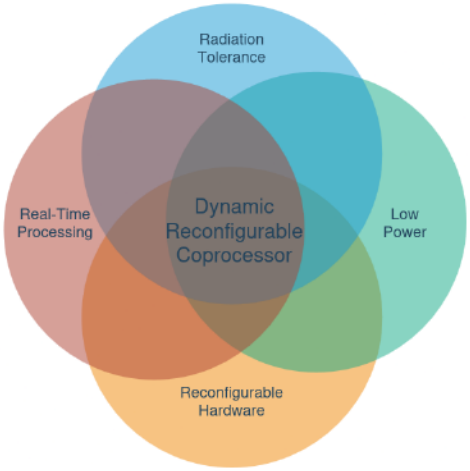


Figure 1. Venn diagram of research scope.

With partial reconfiguration (PR), the FPGA resources can be reused to perform multiple different operations. An advantage of using PR vs full reconfiguration (FR) is the partial bitstreams are only for the reconfigurable region. The smaller partial bitstream size greatly decreases PR latency vs FR latency. [2] New coprocessors were PR'd into the coprocessor block on the FPGA before each operation and PR'd out once the operation is completed. Maintaining the coprocessor in a no operation (NOP) state at the hardware level allows the supervising system to detect radiation faults in the outputs of the coprocessor. Radiation effects, processor selection, and preliminary floating point operation were performed before begin research into the design of the coprocessor.

979-8-3503-5597-0/25/$31.00 ©2025 IEEE

2025 IEEE Aerospace Conference | 979-8-3503-5597-0/25/$31.00 ©2025 IEEE | DOI: 10.1109/AERO63441.2025.11068533

Radiation Faults

The majority of small satellites operate in Low Earth Orbit (LEO) and High Earth Orbit (HEO) outside the protection of the Earth's atmosphere and magnetosphere. Space computers for small satellites operate in harsh radiation environments which leads to multiple types of failures. Space computers suffer from two board categories of radiation effects, Total Ionizing Dose (TID) and Single Event Effects (SEE), while operating inside this harsh radiation environment. [3] Both of this radiation effects are caused by ionizing radiation striking the substrate of an integrated circuit (IC) and depositing energy. TID occurs when lower energy protons and electrons ($\leq$ 30 MeV/amu) deposit energy and create electron/hole pairs in the IC's substrate layers. [4] When the trapped charge is inside the gate oxide of a transistor, it will alter the threshold voltage and turn the semiconductor either always on or always off. When the trapped charge is inside the isolation region between transistors, it causes leakage current and power consumption until the device is burned out. Modern IC feature size ($< 65nm$) makes most devices naturally resistant to TID since lower energy particles are less likely to be trapped in isolated region or gate oxide of devices. [5], [6], [7], [8] TID failure is a culmination of gradual degradation of the device. Modern COTS process nodes ($< 65nm$) have high levels of TID immunity ($> 100krad$) due to their thin insulation regions reducing the likelihood of charge trapping. [7], [9] See Figure 2 for illustration of the two cases of TID caused damage.

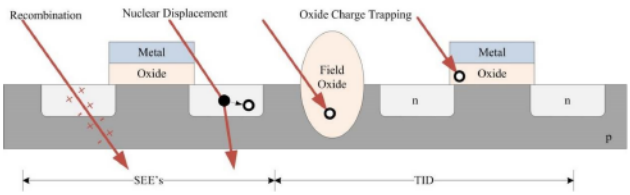


Figure 2. TID and SEE in CMOS cross-section.[10]

SEE occurs when high energy particles and heavy ions strike the diffusion regions of a device as shown in Figure 2. SEEs do not necessarily cause permanent damage. However, the electron/hole pairs created by SEEs cause unwanted logic-level transitions inside ICs. These logic-level transitions cause system faults when the transitions occur on a digical logic line, single event transient (SET), or when the transitions are stored, single event upset (SEU). Failures, single event functional interrupt (SEFI), are caused when an SEU cannot be recovered from by resetting the affected circuit and entire system requires a power cycling, full system re-initialization, or reconfiguration.

RadPC

The softcore processor selected for the coprocessor integrated is RadPC, a custom reduced instruction set computer (RISC-V 32I) designed for radiation tolerance. See Figure 3. RadPC is implemented as a logic design and developed in the VHSIC Hardware Description Language (VHDL). RadPC is an architectural approach to mitigating SEEs on space computers where the detection and recovery is abstracted from the software developer. This computer architecture expands triple modular redundancy (TMR) to quad modular redundancy (QMR) at the central processing unit (CPU) level. RadPC is a QMR microcontroller with a RISC-V architecture which is implemented on an FPGA. [11], [12]. When faults are detected, the internal registers of the faulted CPU are reloaded with correct values. In the event of a complete failure from SEEs, individual CPUs are PR'd and registers reloaded to fully recovery and synchronize the system. [13], [14] These recovery methods combined with memory scrubbers provide resilience from SEE faults. [11], [13], [15] This architectural approach to recovery from radiation induced faults allows the system to be easily expanded to include dynamic reconfigured coprocessors.

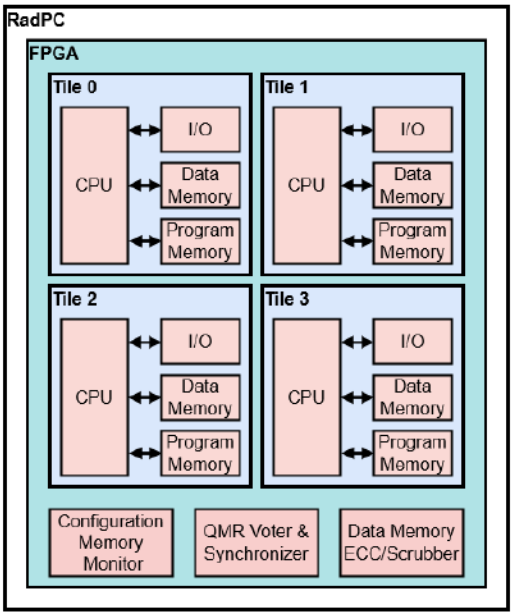


Figure 3. RadPC computer architecture.

Since RadPC uses PR for recovery from radiation strikes, the hardware and softcore architecture is easily adaptable to using PR to dynamically swap coprocessors. RadPC only supports integer operations inside its ALU. Incorporating a floating point unit (FPU) as a coprocessor proved the feasibility of using dynamic partial reconfiguration to expand the computation capability of the system.

The primary reasons for RadPC's selection for integration with the floating point coprocessor is that the development hardware and system architecture already support partial reconfiguration. Furthermore, RadPC is an integer processor makes the floating point coprocessor a valid test case for the expansion of capabilities.

Preliminary Coprocessor Work

As proof of concept, the floating point operations were implemented on an FPGA development board (Nexys A7). This preliminary work implemented nine (9x) of the most common floating point operations (addition, subtraction, multiplication, multiply-and-accumulate, divide, modulo, remainder, divide-by-2, and reciprocal). These functions provided a balance between useful capability and timing constraints from long combination logic paths. This feasibility study focused on two goals, determining the requirements for the floating point operations and determining practicality of integrating a floating point coprocessor into RadPC.

Testing showed that the main issue with the floating point operations were the long combination logic paths. Since each operation had different logic paths, the timing for a completed operation was slightly different. To counter this, the input and output were routed directly from and to data memory.

Table 1. Individual Floating Point Operation PR and Compute Times

Name	Function	Test Operation	PR Time	Compute Time
NOP	No Operation	N/A	250 ms	N/A
ADD	Addition	10.75 + 4.25 = 15.00	250 ms	13.65 ms
SUB	Subtraction	10.75 - 4.25 = 6.50	250 ms	13.84 ms
MUL	Multiplication	6.5 * 4.25 = 27.625	250 ms	14.12 ms
MAC	Multiply/Accumulate	319.33 / 6.5 = 49.12769	250 ms	15.04 ms
DIV	Division	rem 15 / 4 = 3	250 ms	14.12 ms
DIVBY2	Division-by-2	16 mod 5 = 1	250 ms	15.51 ms
MOD	Modulus	1 / 5 = 0.2	250 ms	13.11 ms
REM	Remainder	19.33 / 16 = 19.958125	250 ms	13.56 ms
RCP	Reciprocal	2 * 5 + 0.75 = 10.75	250 ms	10.68 ms

The CPU performing data management for the coprocessor is by default much slower than the coprocessor and allowed the coprocessor to avoid timing problems by updating the memory faster than the CPU was reading it. PR regions were also adjusted to allow individual functions similar compute times.

Implementation of the floating point operations together with RadPC demonstrated the FPGA would not support the RadPC computer and all nine operations simultaneously. The most efficient solution for the system was a coprocessor which implemented a single floating point operation at a time and swapped operations dynamically. The results from this testing in PR time and compute time are shown in Table 1.

Furthermore, the floating point operations are combinatational logic which would be operating within a 16 MHz computer. Reading the output with an oscilloscope showed a delay or buffering was needed a computer clocked at 8 MHz to 32 MHz. A delay was build into all the mathematical operations to ensure that the output had time to update before being read. Changing the operating frequency requires that the delay be tuned to match. Altering the delay requires that RadPC with the integrated FPU be resynthesized, implemented, and bitstreams regenerated. This issue will need to be addressed by committing to a 16 MHz clock or adjusting the coprocessor to accept a delay.

The preliminary investigation for the coprocessor demonstrated the key requirements for designs were standardized the input/outputs (I/O) with support for all nine of the floating point operations, a VHDL wrapper of the coprocessor component to support PR, single floating point operation implementation, and read/write capability to the data memory for loading data directly into the coprocessor.

2. RESEARCH

The preliminary work found a coprocessor based on the FPU was possible within the RadPC computer architecture and that swapping the floating point operation via PR was possible. Based on this work, a floating point coprocessor for the nine (9x) floating point operations was developed in a three-step process. Firstly, the data storage for the coprocessor was developed by testing different setups, direct memory access, First-In-First-Out (FIFO) storage, peripheral memory addresses, registers, etc. Secondly, floating point coprocessor was developed. The I/O for the coprocessor was standardized to allow for floating point operation and development of future coprocessors. Various PR region locations for the floating point coprocessor within RadPC's architecture were tested. Finally, the floating point coprocessor was integrated within RadPC and implemented on an FPGA development board for verification.

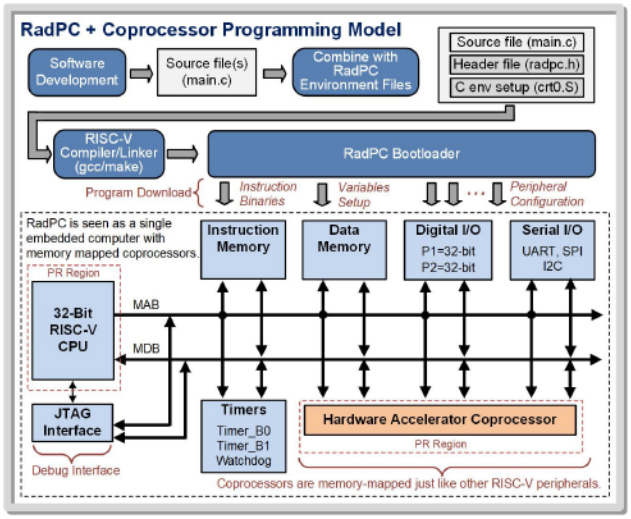


Figure 4. RadPC+coprocessor integrated model.

Coprocessor Data Storage

During operation, the coprocessor needed to receive input data from and return results to the CPU. Input data was constrained to float32 format and required access by a user on the software level. Results required storage in memory until read by the CPU and passed to the software level for delivery to the user. The majority of solutions for this coprocessor data memory and transfer are custom to the CPU. Tightly coupled coprocessors with specific opcodes to peripheral coprocessors with a dedicated interface. [16], [17], [18] Three options were tested for the memory storage, separate data memory for the CPU and coprocessor, shared data memory for the CPU and coprocessor, and fast storage for the coprocessor with dedicated data memory locations.

Separate data memory for the CPU and coprocessor was functional, but inefficient. The sequential data movement from CPU data memory to coprocessor data memory resulted in high latency. Total operation time was dominated by data transfer between the two data memory spaces. Adding

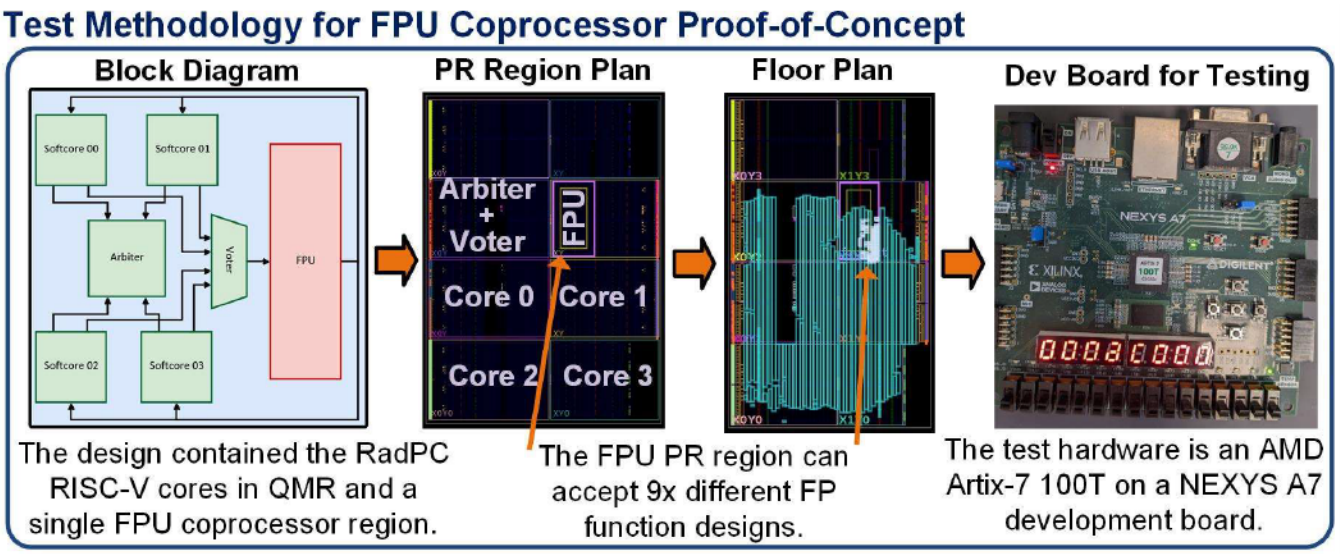


Figure 5. RadPC with FPU implementation showing FPGA layout, operation, and testing of the ADD on physical hardware (Xilinx 100T FPGA): Courtesy of Resilient Computing

the second data memory of usable size for the coprocessor required more fast onboard FPGA RAM than was available for the targeted FPGAs (Xilinx 100T and 200T).

The shared data memory was functional in terms of available onboard FPGA RAM. However, there were issues with the CPU and coprocessor being in contention when accessing the memory. For example, the CPU would read the memory address for the results before the coprocessor had completed writing. Swapping the coprocessor via PR contributed to the contention as PR was unpredictable for the CPU and caused memory access contentions.

Fast storage for the coprocessor with dedicated data memory locations was the solution to the issues of data transfer latency and memory access contention. Registers for the coprocessor were implemented inside the CPU architecture. From the coprocessor's perspective, these registers were continuously feeding data and accepting results. From the CPU's perspective, registers were accessible in the same manner as a standard peripheral's registers. Since the coprocessor registers were loaded into data memory on the same clock as the CPU, contention access was avoided and the data transfer from register to data memory was minimized. This approach treated the coprocessor as an external asynchronous peripheral for the RadPC computer system. Once this approach passed timing, access, and PR testing, it was implemented into the RadPC architecture.

Coprocessor Design and Integration

Integration directly into the RadPC architecture was attempted with the developed coprocessor, floating point unit (FPU). However, it was found that direct integration with RadPC required PBLOCKS where all four cores had to be individually configured with the nine different FPU configurations. This proved impractical as it required $4 \times N_{functions} + 1$ partial bitstreams for N coprocessor operations and maintaining RadPC's PR recovery mechanisms. The bitstreams needed to hold both a single RadPC softcore and a coprocessor which greatly increased the PR region size.

Changing the design to a coprocessor operating in parallel with RadPC allowed the number of bitstreams to be reduced to $(4 + (N_{functions} + 1))$ for N coprocessor operations and RadPC recovery operations via PR. Furthermore, this design verified that a coprocessor could be build in parallel with RadPC on resource limited hardware and perform complex mathematical operations.

Three designs for a coprocessor in parallel to the CPU were investigated. Since the coprocessor is designed as part of radiation tolerant system, the first designed apply QMR to the coprocessor and used PR as both a swapping and recovery option. Four copies of a single function are implemented in parallel, each with an individual PR region. This approach had the highest radiation tolerance out of the coprocessor designs tested. An FPU was developed and preliminary testing of a QMR version was done. After this testing, this approach was abandon due to the large number of FPGA resources, PR bitstreams $(4 \times N_{functions} + 1)$, and PR time required. Routing into the four PR regions introduced significant timing errors and increased the FPGA resource usage. Swapping the functionality of the coprocessor required a minimum of four PR to be performed sequentially and massively increased swap latency. Furthermore, the number of bitstreams needed an enormous amount of external memory for storage between configurations.

The second coprocessor design implemented QMR for the coprocessor within a single PR region for all four coprocessors. Since all four coprocessors were implemented only a single floating point operation concurrently, the timing issues for routing around multiple PR regions were avoided. This approach still required similar number of FPGA resources even when dealing with a single floating point operation per coprocessor. The size of the PR region greatly increased the PR time until real-time operation was unfeasible.

The finalized design, shown in Figure 6, was a coprocessor without QMR inside a single PR region. Testing with the floating point operations using QMR showed it defeated the purpose of maximizing FPGA hardware usage, Having 4 copies of the operations running in parallel increased resource and timing requirements. The most efficient coprocessor framework was implementing the FPU within a PR region

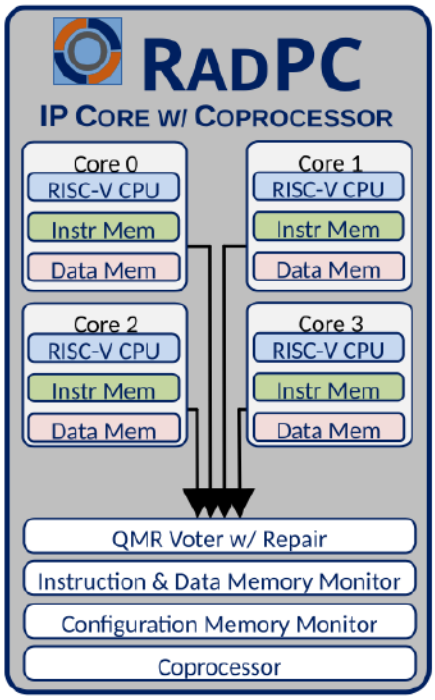


Figure 6. RadPC+Coprocessor block diagram.

as a peripheral of RadPC. This approach, shown in Figure 4, balanced the processor's fault tolerance, optimal hardware resource usage, coprocessor timing requirements, and usability.

Measurements & Results

Testing was performed using the Nexys A7 FPGA Development Board featuring the Artix-7 100T FPGA. The RadPC+Coprocessor system will load onto the FPGA using the Vivado Design Suite. Both full configuration and PR were performed by the user via Vivado's Hardware Server as shown in Figure 5. Measurements were taken with the Analog Discovery 2 on the FPGA's JTAG DONE pin for PR timing. Compute timing was measure with a similar process using a signal routed out from the coprocessor to Nexys' I/O. All nine floating point operations were successfully performed by the coprocessor. The dynamic reconfiguration process worked. Coprocessor's outputs for the ADD function are shown in Figure 7.

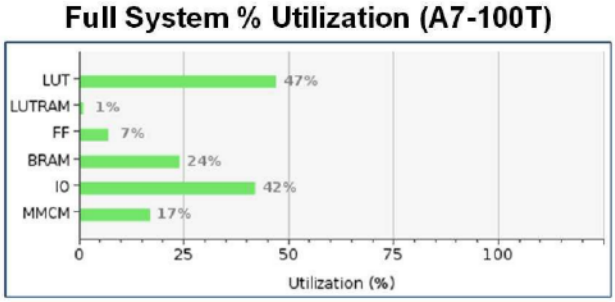
```
[11:17:09:040] RadPC FPU Demo Program
[11:17:09:040] PR External FPU Addition Test
[11:17:09:056] a + b = d
[11:17:09:056] 10.75 + 4.25 = 15.00
[11:17:09:056] Correct Answer = : 0xF0000
[11:17:09:056] Input A    (Hex): 0xAC000
[11:17:09:056] Input B    (Hex): 0x44000
[11:17:09:056] Input C    (Hex): 0x0
[11:17:09:056] Output D   (Hex): 0xF0000
```

Figure 7. Serial report on Coprocessor ADD operation.

A test program for RadPC was created to exercise each of the nine FPU operations using the following steps.

1) RadPC loads inputs into the FPU coprocessor via storing floating point numbers in the memory map location corresponding to inputs A, B, and/or C.
2) RadPC requests the desired FPU operation from the user using control registers for the coprocessor peripheral and repeats the request over a serial terminal.
3) RadPC waits for the coprocessor to be configured with the desired FPU operation
4) AD2 measures the PR time by using an external logic analyzer to monitor the DONE pin on the JTAG configuration interface to the FPGA.
5) Coprocessor begins floating point computation using the start flag in the control register
6) AD2 measures compute time using an external logic analyzer to monitor a coprocessor signal flag which is routed to the GPIO of the FPGA development board and is toggled at the start and end of the FPU coprocessor operation.
7) RadPC reads the results from the coprocessor's memory map location and writes the results to a serial terminal for the user to compare FPU computed results vs correct results for the selected operation.
8) RadPC writes the result to the HEX displays on the Nexys a7 FPGA board for additional verification.
9) Steps 1-8 are repeated for all nine floating point operations.

Look-Up Table (LUT) and I/O usage was 40% of the available percentage of the Artix-7 100T FPGA. Between RadPC's need for LUTs and the coprocessor's need for LUTs, scaling the coprocessor to more complex or multiple simultaneous operations will be difficult. Total resource usage of the RadPC+Coprocessor design is shown in Figure 8.



Full System Resource Utilization (A7-100T)

Resource	Utilization	Available	Utilization %
LUT	29772	63400	46.96
LUTRAM	118	19000	0.62
FF	8953	126800	7.06
BRAM	33	135	24.44
IO	88	210	41.90
MMCM	1	6	16.67

Figure 8. Nexys A7 100T FPGA resource usage.

PR timing and compute timing are listed in Table 2. PR times are the total time for the reconfiguration to be preformed measured from the starting transmission of the bitstream into the FPGA until the end of the transmission when the coprocessor reports ready for operation. The time taken by the user to manually order PR to start was not included because dynamically swapping coprocessor in real-time would be an automated process. Unfortunately, the development hardware for this proof-of-concept does not support automatic coprocessor swapping at this time. The closest representation possible is to measure PR time only and note that an automated version would be faster due to supporting higher transmission clock speeds.

Compute time is the time required for the coprocessor to

Table 2. Coprocessor PR and Compute Times

Function	PR Time	Compute Time
Addition	250 ms	3.11 us
Subtraction	250 ms	3.11 us
Multiplication	250 ms	3.11 us
Multiply/Accumulate	250 ms	3.61 us
Division	250 ms	3.11 us
Division-by-2	250 ms	3.11 us
Modulus	250 ms	3.11 us
Remainder	250 ms	3.11 us
Reciprocal	250 ms	2.62 us

receive an input, perform the floating point operation, and return the result. This time was measured via a flag raised by the coprocessor upon receiving input data and lowered once result data is output. The floating point operations were implemented as combinatational logic which by default was much faster than the CPU. To insure the flag accurately matched the start and end of the compute time, the floating point operation was implemented in a process where the flag must be raised and lower immediately before and after operation. As shown in Table 2, the PR times are a factor of 10^5 greater than compute times.

3. CONCLUSIONS

The principal results of this work were a proof-of-concept design which demonstrated the possibility of implementing a dynamically reconfigurable coprocessor within the RadPC architecture.

Applications & Advantages

The proof-of-concept design confirmed that dynamically reconfigurable coprocessors for low-power, real-time operation is possible for small satellites. Custom coprocessors implemented with this design would be increase the processing capability of low power systems. Furthermore, the equipment required for an automated dynamically partially reconfigurable system fits within the size, mass, and power constraints for a small satellite. Key future applications is image preprocessing/digital signal processing where dedicated processing hardware is needed for a short period. [19], [20] An image preprocessing coprocessor could be swapped after use and the hardware used to support other operations. The advantage of this design is the capability to change the hardware during real-time operation. The reconfiguration process would also clear out any SEEs while loading the new coprocessor. This adds a limited, but beneficial recovery method from radiation strikes for the coprocessor subsystem.

Limitations

This design illustrated several issues to be addressed. First, the PR latency is the dominating factor for real-time capability. PR time was 250 ms vs 3.61 us for compute time. With the current hardware setup, the CPU is required to pause operation and wait for the user to initiate the PR. Between the PR time and the wait time, the CPU spent the majority of its time during testing paused. Obviously, the CPU being paused the majority of the time would not be acceptable for actual operations. PR latency needs to be reduced, and the PR process automated for this design to work in real-time operation.

Second, the hardware setup was lacking in key areas to support PR. FPGA size limited the size of coprocessors which could be implemented in parallel to the main processor. Major limiters were the available memory and integrated hardware support for PR. FPGA RAM limited storage for coprocessor data transfer. FPGA fabric size limited the system size due to routing conflicts Onboard memory, which is memory external to the FPGA for configuration bitstreams, limited the number of coprocessor configurations. These issues demonstrated the need for hardware capable of store bitstreams, rapidly configuring the FPGA, and a larger FPGA to support more complex coprocessor operation.

Third, the coprocessor design did not adequately address radiation tolerance. Constraints on partial bitstreams numbers limited the redundancy built into the coprocessor component. With the current design, the coprocessor's main radiation protection and recovery method is PR. This method flushes out faults during the PR swapping the coprocessor in then assumes the coprocessor will complete operation and be swapped out before being faulted. Modeling and radiation testing is needed to prove this works in the majority of cases.

Further Work

The current work illustrated multiple limitations to iterate on for this current proof-of-concept design, hardware improvements, radiation modeling, radiation testing, coprocessor timing, and new coprocessor development. This proof-of-concept design showed PR is a viable option for swapping coprocessors. However, the PR latency is the limiting factor for applying this solution to real-time operations. The 250 ms required for PR swapping the coprocessor is 8 million clock cycles for RadPC operating at 32 MHz. Reducing the PR latency is one of the next steps for this research. The proposed solution is the development of a new printed circuit board which includes both the FPGA and a microcontroller for PR purposes. By concentrating devices on a single board, the JTAG clock frequency increased above 6MHz. This new board would permit RadPC to preload coprocessors from the software level similar to PR schedulers. [21], [22], [23] Preloading would be preformed by having RadPC request a specific coprocessor beforehand and continue operation while the coprocessor is swapped in the background.

For the coprocessor development, the proposed research path is expanding the variety of coprocessors. Floating point operations were successful which suggests fixed point operations, image preprocessing operation, or convolution operations would be possible. Given the extensive use of image processing for small satellites, image preprocessing would be an excellent use of a dynamically reconfigurable coprocessor.

ACKNOWLEDGMENTS

The authors thank the National Aeronautics and Space Administration (NASA) for funding this project.

The authors thank Montana State University - Bozeman (MSU) for providing funding, lab space, equipment, and personnel for this project.

The authors thank Resilient Computing for providing personnel, equipment, and advisors for this project through their

partnership with MSU.

REFERENCES

[1] B. Osterloh, H. Michalik, S. A. Habinc and B. Fiethe, "Dynamic Partial Reconfiguration in Space Applications," 2009 NASA/ESA Conference on Adaptive Hardware and Systems, San Francisco, CA, USA, 2009, pp. 336-343, doi: 10.1109/AHS.2009.13.

[2] W. Lie and W. Feng-yan, "Dynamic Partial Reconfiguration in FPGAs," 2009 Third International Symposium on Intelligent Information Technology Application, Nanchang, China, 2009, pp. 445-448, doi: 10.1109/IITA.2009.334.

[3] Holmes-Siedle, A., Adams, L., "Handbook of radiation Effects", NY, Oxford Univ. Press, 2002.

[4] H. L. Hughes and J. M. Benedetto, "Radiation effects and hardening of mos technology: devices and circuits," IEEE Transactions on Nuclear Science, vol. 50, no. 3, pp. 500–521, 2003.

[5] H. J. Barnaby, "Total-ionizing-dose effects in modern cmos technologies," IEEE Transactions on Nuclear Science, vol. 53, pp. 3103–3121, 12 2006.

[6] R. Liu, A. Evans, L. Chen, Y. Li, M. Glorieux, R. Wong, S.-J. Wen, J. Cunha, L. Summerer, and V. Ferlet-Cavrois, "Single event transient and tid study in 28 nm utbb fdsoi technology,"IEEE Transactions on Nuclear Science, vol. 64, no. 1, pp. 113–118, 2017.

[7] C. Zhang and et al., "Total ionizing dose effects on analog performance of 28 nm bulk mosfets," 2017 47th European Solid-State Device Research Conference (ESSDERC), pp. 30–33, 2017.

[8] C. Zhang and et al., "Gigarad total ionizing dose and post-irradiation effects on 28 nm bulk mosfets," 2016 IEEE Nuclear Science Symposium, Medical Imaging Conference and Room-Temperature Semiconductor Detector Workshop (NSS/MIC/RTSD), pp. 1–4, 2016.

[9] Arora, R., et al., "Impact of Technology Scaling in sub-100 nm nMOSFETs on TID Radiation Response and Hot-Carrier Reliability," IEEE Trans. on Nuc. Science, vol. 61, no. 3, June 2014.

[10] B. J. LaMeres, "Fpga-based radiation tolerant computing," Journal of Aerospace Information Systems, 2012.

[11] C. M. Major, A. Bachman, C. Barney, S. Tamke, and B. J. LaMeres, "Radpc: A novel single-event upset mitigation strategy for field programmable gate array-based space computing,"Journal of Aerospace Information Systems, vol. 18, no. 5, pp. 280-288, 2021.

[12] J. Williams, C. Barney, Z. Becker, J. Davis, C. Major, B. J. LaMeres, and B. Whitaker, "Radpc@scale: A novel approach to radpc single event upset mitigation strategy," 2022 IEEE Aerospace Conference, 2022.

[13] B. J. LaMeres and C. Gauer, "Dynamic reconfigurable computing architecture for aerospace applications," 2009 IEEE Aerospace Conference, 2009

[14] C. Gauer, B. J. LaMeres, and D. Racek, "Spatial avoidance of hardware faults using fpga partial reconfiguration of tile-based soft processors," 2010 IEEE Aerospace Conference, 2010.

[15] J. Hane, B. J. LaMeres, R. W. T. Kaiser, and T. Buerkle, "Increasing the radiation tolerance of fpga-based computers through redundancy and environmental awareness," Journal of Aerospace Information Systems, vol. 11, no. 2, pp. 61-75, 2014

[16] J. W. Abdala Castro and A. Morales-Villanueva, "Exploring Dynamic Partial Reconfiguration in a Tightly-coupled Coprocessor Attached to a RISC-V Soft-processor on a FPGA," 2021 IEEE XXVIII International Conference on Electronics, Electrical Engineering and Computing (INTERCON), Lima, Peru, 2021, pp. 1-4, doi: 10.1109/INTERCON52678.2021.9532810.

[17] N. Charaf, A. Kamaleldin, M. Thümmler, and D. Göhringer, "RV-CAP: Enabling Dynamic Partial Reconfiguration for FPGA-Based RISC-V System-on-Chip," 2021 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), Portland, OR, USA, 2021, pp. 172-179, doi: 10.1109/IPDPSW52791.2021.00033.

[18] J. Tarrillo, F. A. Escobar, F. L. Kastensmidt and C. Valderrama, "Dynamic partial reconfiguration manager," 2014 IEEE 5th Latin American Symposium on Circuits and Systems, Santiago, Chile, 2014, pp. 1-4, doi: 10.1109/LASCAS.2014.6820293.

[19] B. Krill, A. Amira, A. Ahmad and H. Rabah, "A new FPGA-based dynamic partial reconfiguration design flow and environment for image processing applications," 2010 2nd European Workshop on Visual Information Processing (EUVIP), Paris, France, 2010, pp. 226-231, doi: 10.1109/EUVIP.2010.5699127.

[20] C. V. Borkute, A. Y. Deshmukh and C. N. Kharkar, "Dynamic Partial Reconfiguration in FPGAs for DSP Applications," 2011 Fourth International Conference on Emerging Trends in Engineering & Technology, Port Louis, Mauritius, 2011, pp. 253-257, doi: 10.1109/ICETET.2011.70.

[21] X. Shi and A. Zou, "Preemptive FPGA Scheduling Based on Dynamic Partial Reconfiguration," 2024 Conference of Science and Technology for Integrated Circuits (CSTIC), Shanghai, China, 2024, pp. 1-3, doi: 10.1109/CSTIC61820.2024.10531952.

[22] A. Dhar et al., "DML: Dynamic Partial Reconfiguration With Scalable Task Scheduling for Multi-Applications on FPGAs," in IEEE Transactions on Computers, vol. 71, no. 10, pp. 2577-2591, 1 Oct. 2022, doi: 10.1109/TC.2021.3137785.

[23] M. Angioli, M. Barbirotta, A. Mastrandrea, S. Jamili and M. Olivieri, "Automatic Hardware Accelerators Reconfiguration through LinearUCB Algorithms on a RISC-V Processor," 2023 18th Conference on Ph.D Research in Microelectronics and Electronics (PRIME), Valencia, Spain, 2023, pp. 169-172, doi: 10.1109/PRIME58259.2023.10161944.

BIOGRAPHY

Hezekiah Austin *received an A.A. degree in General Studies from Regent University in 2014, a B.S. with Honors in Electrical Engineering from Montana State University in 2020, and an M.S. in Electrical Engineering from Montana State University in 2023. He is currently a research assistant and Ph.D. candidate at Montana State University, researching dynamically reconfigurable hardware accelerators for FPGA-based, low power, radiation tolerant computer systems.*

Chris Major *received his Ph.D. in electrical engineering from Montana State University (MSU). He is currently the principal engineer at Resilient Computing and the principal investigator on multiple NASA SBIR projects to advance space computing. His expertise is in FPGA design with specific focus on implementing novel computer architectures that leverage real-time configuration of the programmable FPGA fabric. Major's research is sponsored in part by NASA and the National Science Foundation.*

Zachary Becker *received the B.A degree in design from Virginia Commonwealth University, Richmond in 2009, and the B.S. degree in computer engineering from Montana State University, Bozeman in 2024. He is currently a Research Engineer working with embedded systems (microcontrollers and field-programmable gate arrays) for the application of communication systems and signal processing in the field of optics and photonics at Spectrum Lab, Montana State University. He has a decade of prior experience working as a software engineer.*

Tristan Running Crane *received a B.S. degree in Electrical Engineering and a B.S degree in Computer Engineering from Montana State University in 2021 and an M.S. degree in electrical engineering from Montana State University in 2023. Currently, he is working as a research assistant working towards a Ph.D. degree in Electrical Engineering from Montana State University. The research area he is focused on is advancing the cybersecurity of edge computers used in the nation's power grid.*

Kris Allick *is currently an undergraduate in his last year of a B.S. degree in Computer Engineering at Montana State University. He has spent each year of his degree working in Aerospace labs. First with the MSU Balloon Outreach, Reasrch, Exploration and Landscape Image System (Borealis) and currently as an undergraduate researcher with the Radiation Tolerant Computer Lab at MSU. From these experiences, he has developed a specialty in embedded systems design and testing.*

Brock J. LaMeres *(M'98-SM'09) received the B.S. degree in electrical engineering from Montana State Univ., Bozeman in 1998, and the M.S. degree in electrical engineering from the Univ. of Colorado, Colorado Springs in 2001, and the Ph.D. degree in electrical engineering from the Univ. of Colorado, Boulder in 2005. He is currently a Professor in the Department of Electrical and Computer Engineering at Montana State University (MSU), Bozeman. LaMeres teaches and conducts research in the area of digital systems. LaMeres' research is sponsored in part by NASA, the National Science Foundation, the Idaho National Laboratory, and the Pacific Northwest National Laboratory. LaMeres has published over 100 papers and 5 textbooks in the area of computer engineering and holds 15 US patents in the field of digital systems.*