SHARP: A Space HARdened Processor for Next Generation CubeSats

G. Alonzo Vera, Jeff Love, Jorge Piovesan, Brendan Burke
IDEAS Engineering and Technology, LLC
2350 Alamo Ave. SE Suite 220
avera@ideas-tek.com

Jesse Mee
Space Electronics Technologies Air Force Research
Laboratory Kirtland AFB, NM 87117
jesse.mee@us.af.mil

ABSTRACT

As small satellites continue to proof capable of producing useful scientific data and supporting commercial applications their typical safe orbit/short life span missions are replaced by longer duration missions at harsher orbits. Radiation effects concerns become more serious and steps need to be taken toward improving the reliability of the spacecraft's electronics. Thus, there is a need for high reliability computing engines capable of surviving this new type of mission. The use of qualified radiation hardened parts is generally too expensive for this new type of missions. Less expensive alternatives have focused on designing computing engines using radiation effects mitigation approaches at the architectural level (e.g redundancy, testing, failsafe mechanisms, etc). IDEAS-TEK has taken the approach of designing a custom radiation hardened ASIC using the same techniques that high-end radiation hardened microelectronics is designed with, with the exception of high end costly tools and processes. This paper presents SHARP- a 32-bit RISC processor being developed under sponsorship of AFRL as a radiation hardened ASIC at 180nm technology node. SHARP prototypes (ASICs and equivalent soft-cores FPGA implementations) are expected to be available early 2018.

BACKGROUND

As small satellites are deployed to harsher orbits and longer missions are considered, concerns over radiation effects on the spacecraft electronics become more serious. However, radiation hardened qualified devices are still generally too expensive for these missions. To improve the spacecraft reliability, designers usually take the approach of implementing mitigation techniques at the architectural level at the cost of increasing complexity and Size, Weight and Power (SWaP).

Of particular concern are the computing engines in the spacecraft. These are commonly at the center of onboard critical systems, and are by nature less tolerant to occasional data corruption due to Single Event Upsets (SEUs) or reboots generated by system failures or latchup mitigation approaches. The current CubeSat and small satellite ecosystem comprises a wide variety of processors, ranging from right out of the box Commercial Off-The Shelf (COTS) devices [1] to carefully screened parts in fault tolerant schemes that include redundancy and a myriad of failsafe mechanisms [2]. Recently, radiation hardened devices have been made available at prices that are accessible to CubeSat missions [3]. The inexpensiveness of these devices is due to the lack of qualification (QML-V) as well as the microprocessor architecture (low cost or free licenses). IDEAS-TEK has taken this approach to develop a Radiation Hardened 32-bit RISC processor based on the OR1K architecture that will enable high reliability onboard computing at a fraction of the cost of traditional radiation hardened microelectronics. The reduced cost is accomplished by 1) using a radiation hardened standard cell library available to IDEAS-TEK at no cost through a strategic partnership for this design and others in the future, 2) using open source tools for the ASIC front-end, 3) using a Multi-Project Wafer (MPW) at the180nm node and 4) using an open source microprocessor architecture. The result of this effort is a 32-bit RISC processor ASIC labeled Space HARdened Procesor or SHARP. At the core of SHARP is an OpenRISC1000 (OR1K) open source architecture CPU originally designed as a soft-core in that it is malleable and can be configured in many different ways to fit a particular application or implementation target. This feature was pivotal in the decision of using this architecture for SHARP and its ASIC implementation.

Comparison with other available solutions

There are other soft-core solutions that have been used for high reliability, fault tolerant and space applications such as [4-6]. Except for [5], none of these have yet being ported into an ASIC to the best of our knowledge – let along a radiation hardened ASIC. Additionally, source code availability is either restricted or obscured by using proprietary methods and there are costs involved in licensing them. The OR1K architecture has been ported before to ASICs with success at different technology nodes and in different configurations and it is freely distributed under a GNU Lesser General Public License (LGPL) license.

In terms of performance most of the soft-cores available have comparable metrics. Although difficult to compare in equal terms given the different – highly configurable – features each architecture has, they are considered roughly equivalent. In order to benchmark soft-cores, it is important to choose near equivalent configurations for each. The choice of configuration should also be specific to the application for which the processor will be used. For instance, if resources are limited in the application, the benchmarks should be run on resource optimized configurations of each processor.

Certain benchmark tests are known to be good indicators of how a core will perform. The benchmarks of interest for our application are separated into two categories: Integer and Floating-Point. Integer tests are a base requirement for benchmarks and will typically include compression algorithms (Zip, JPEG, Huffman), sorting algorithms (bubble, tree, quick), hash functions (SHA256), and basic arithmetic operations (multiplication, division). Some processors do not include hardware floating-point support, so in these cases floating-point tests will actually reflect integer performance. Typical floating point tests include Fast Fourier Transforms (FFT), linear algebra, and floating point arithmetic operations (multiplication, division, square roots)[7-8].

The Dhrystone, Stanford, and CoreMark benchmark suites are examples of standard benchmarking programs that each has their benefits (See Table 1).

Previous efforts [9-10] have been made to benchmark the soft-cores in [4-6] (namely, the RISC-V, Leon4 and OpenRISC), however, these were performed on different FPGAs and using very specific compiler optimizations. This benchmark showed the RISC-V Rocket performing at 2.32 CoreMark/MHz/Core on an unknown FPGA, whereas OpenRISC 1200 performed at 1.34 CoreMark/MHz/Core on a Xilinx ML501. Additionally, Leon 4 and Leon 3 performance are listed at 1.7 DMIPS/MHz, or 2.1 CoreMark/MHz and 1.4 DMIPS/MHz, 1.8 CoreMark/MHz respectively [11, 5]. These results will be used to compare and evaluate SHARP when the undergoing benchmarking efforts in its soft-core form or FPGA implementation is completed.

Dhrystone
• Does not stress memory of newer systems. • No floating-point. • Much of the execution time is spent in basic library functions, making it a measure of compiler performance.
Stanford
• Simple and modifiable. • Includes neither compression nor hash tests
CoreMark
• Has become the new standard because of flaws with Dhrystone. • Tests a large variety of different integer and floating-point operations. • Complicated

Table 1: Classic benchmarks pros and cons [7]

SHARP ARCHITECTURE

SHARP includes serial communication channels (UART, SPI, I2C) as well as GPIOs, a Floating-Point Arithmetic Unit, and 128Kbits of onchip SRAM. A simplified block diagram of the system is shown in Figure 1.

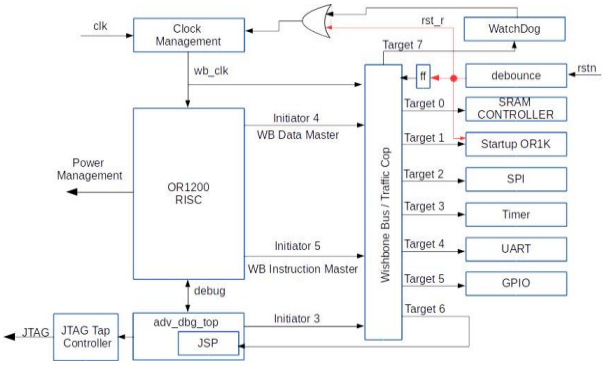

Figure 1. SHARP's high level block diagram

SHARP architecture was designed to be ported to an ASIC, or an FPGA platform. Soft versions (RTL for FPGA implementations) of SHARP have been developed for Xilinx© and Microsemi© devices. The soft-cores can be used for development and testing, or as a starting point for custom designs targeting FPGAs. A prototype version of one of these soft-cores (Xilinx© on and IDEAS-TEK's ARC board for CubeSats) is manifested in the New Mexico Institute of Mining and Technology CubeSat scheduled for launch in 2018. The main architecture features of each of these instantiations of the SHARP core are listed in Table 2.

SHARP's main differentiating features	Implementation Targets		
	ASIC @ TJ 180nm	Xilinx's Virtex 5	Microsemi RTG4
Freq. of operation	100MHz	150MHz	150MHz
Cache size	4KB (D/I)	8KB (D/I)	8KB (D/I)
On Chip Memory	128Kb	32KB	32KB
Spacewire Support	-	√	√
CAN Bus Support	-	√	√
Power Consumption	low	high	medium
Footprint	Small	Large	Large
Cost	Low	High	High
Radiation effects mitigation	By design / Process	Architectural enhancement	By design / Process
Thrusted	Potentially yes	No	No
Radiation hardiness level	>300K	No	>100K

Table 2: SHARP's main features on each of the different implementation versions

It is desirable for the different instantiations of the design to be swappable. This feature would allow a team to develop a SHARP-based system using – for instance – a version of SHARP instantiated in a Xilinx® device for ease of testing and development to later drop in the ASIC implementation for a more resilient, low power implementation without having to change the system's schematic significantly. To this end, SHARP is described using Verilog, making sure that it is synthesizable on all the expected implementation targets. In some cases, defines and parameters are used to direct the synthesizer to use primitives available on specific targets (e.g. memory using BRAMs in Xilinx devices, vs. LSRAMs on Microsemi's RTG4 and standard cell library memory components for the ASIC).

Additionally, the design requirement of each target implementation being a drop in replacement for the others drove some architectural features in the system. In particular, the memory controller for the system had to be compatible with commercially available suitable memory chips to be used externally to the system, as well as the memory structures available in each of the different platforms the core can be ported on. Obviously, a *highly platform-dependent* component such as memory is instantiated differently on each of the possible implementation targets, but the overall system implementation behaves similarly on all targets. Additionally, memory has to be protected against Single Event Upsets (SEU), and this protection takes different forms depending on the target platform. Details are described in this paper's subsections specific for each target.

Another aspect of the design that was highly customized to fulfill this requirement was the bootloader (or startup block in Figure 1). This block had to account for the different options it could extract the boot code from, which varies depending on the target implementation of the system. Both, memory and bootloader considerations are described in the following subsections.

Memory controller

In general terms, Static Random Access Memories (SRAMs) are usually faster and more rugged to radiation than Dynamic Random Access Memories (DRAMs) although they are usually more expensive to produce. Our literature search yielded several manufacturers and memories of different densities that could be used for our purposes. Table 3 shows a list of the main options we found and took into consideration while designing SHARP.

Part	Mfr	Size (Mb)	Async/ Sync	TID – KRad(Si)
AT68166H	ATMEL	16	Async	> 300
AT60142H	ATMEL	4	Async	>300
SMV512K32-SP	TI	16	Async	>300
RC7C1024RHS	RedCat	1	Async	>300
RadStop QDR-II SRAM	Cypress	-	Sync	> 300
RadStop Fast Async SRAM	Cypress	-	Async	>300
3D2D512M16VS1605	3D Plus	512	Sync	> 50
3DSR4M08CS1647	3D Plus	4	Async	> 1000
HLXSR01632	Honeywell	16	Async	> 3, >1000
16Mb Monolitic SRAM	BAE	16	Async	>1000

Table 3: SRAM options for space use in conjunction with the RH-OpenRISC

Synchronous SRAMs are generally more dense and faster than their Asynchronous counterparts. On the other hand, asynchronous memory controllers are simpler than synchronous ones and more flexible in terms of adapting to different memories. Since speed and density are not critical factors for our application (SHARP runs at 100MHz which is relatively slow for fast synchronous memories and RAM of 4Mb is more than adequate for the applications we are targeting) asynchronous memories are preferred for SHARP. For these reasons, SHARP includes a memory controller for

asynchronous SRAM memory with a WishBone interface to SHARP's CPU. The internal SRAM on the FPGA implementations are modeled to match the controller's interface, so that internal and external SRAM can be treated as one. The core includes EDAC encoding and decoding circuitry compatible with byte addressable scheme implemented at the SHARP main data bus.

Bootloader

SHARP's bootloader is built around a custom interface to access an external (or internal) non-volatile memory that stores the microprocessor's program. In the case of external non-volatile memories, the interface used implements the SPI protocol. In the case of internal non-volatile memory (for the RTG4 implementation), a custom interface was developed.

After a reset, the bootloader module freezes the microprocessor program counter and assumes control of the Wishbone bus. It then accesses the non-volatile memory to read its contents. Once data is received the bootloader stores it into the microprocessor's RAM. Once all data is transferred, the bootloader releases the Wishbone bus and the microprocessor assumes control as the program counter starts to increment. For this approach to work correctly, the bootloader has to be aware of the processor's RAM configuration (e.g. external vs. internal RAM, reset vector address, etc.). To this end, the bootloader includes a small set of instructions used to drive the non-volatile memory interface and the storing of information into particular positions of the RAM. This code varies depending on the target implementation. However, the overall functionality remains the same, thus fulfilling the design requirement of swappable implementation targets.

ASIC IMPLEMENTATION

The ASIC version is expected to operate @100MHz and withstand 300Krad(Si) and be latchup immune. This implementation was originally restricted to a 5mm x 5mm die and roughly 200 I/Os. An initial estimation of the system size, without including memory structures (SRAM, Cache, etc) yielded an area 5.63mm^2.

The ASIC implementation of SHARP is using a radiation hardened by design standard cell library at TowerJazz's 180nm technology node. The library currently only counts with a single block Asynchronous Static RAM memory block that implements 128Kbits (4K addresses and 32-bit words, plus 8-bits for EDAC). Each one of these blocks is roughly 1950µ by 2520µ. Given our 5mm by 5mm die restriction, SHARP can only instantiate 3 of these memory cores besides the CPU and peripherals logic. Figure 2 depicts the concept implementation currently being worked out.

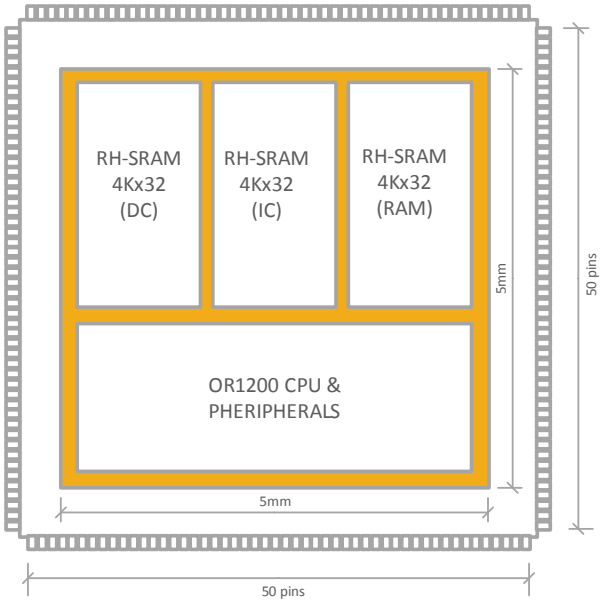

Figure 2. SHARP's ASIC layout concept

SHARP's RTL description was synthetized using open source tools and custom scripts. The resulting output netlist is provided to the back-end ASIC tools for place and route. A SystemVerilog-based verification suite is being developed to verify the correct functionality at each step of the ASIC flow.

Tapeout is expected on Q3-2017, with first prototypes available in Q1-2018.

FPGA IMPLEMENTATIONS

For SHARP's softcore version we picked Xilinx's Virtex 5 and Microsemi's RTG4 devices as target. The Virtex 5 was chosen because there is a radiation hardened part (XQR5VFX130) and a commercial equivalent that would fit the requirement of being interchangeable if necessary. This would allow a relatively inexpensive test and development setup that can be upgraded with a rad-hard part for flight. The Microsemi's RTG4 was selected as the lower power alternative for a radiation hardened FPGA. As the device is relatively new in the market, IDEAS-TEK also saw a commercial opportunity in providing a soft-core solution. Both instantiations of SHARP are expected to be available (including an extensive verification suite) in Q1-2018.

For the soft-cores implementation, SHARP's architecture has been enhanced using well know single event effects mitigation techniques such as Triple Modular Redundancy (TMR), Error Detection and Correction (EDAC), CRC, scrubbing, and watchdog

timer. Decisions on where to implement which approach are based on an analysis of the architecture. Each of the methods used are described in the following subsections.

Watchdog timer

Timers are common peripherals in microprocessor systems; they can be used for scheduling via interruptions, for the generation of pulse width modulated signals, etc. Commonly available timer descriptions can be added on or modified to also serve as a watchdog timer, implementing a mechanism to reset the CPU in case a SEE sends the processor to an unrecoverable state. This mitigation approach does not have an impact on performance, and has minimal impact in terms of logical resource use. The original OR1K architecture does include a timer used for other functions. Although this timer core could be used, it was considered more convenient to implement a separate core, outside of the CPU, whose only function is to serve as a watchdog for the system. Figure 1 shows the implementation of this alternative. The core used for this task is a modified version of a core available at OpenCores.org. When implementing this solution, two options were considered: 1) make the core independent (not connected to the CPU bus and use a generic I/O to reset (pet) the timer, or 2) connect the core to the CPU bus and use a memory mapped address to reset (pet) the timer. The latter option was chosen for simplicity; however we do not discard using the first solution further down the road.

SEE Mitigation on latency intolerant components

The OpenRISC 1200 CPU (an implementation of the OR1K architecture) has the following memory components that are intolerant to latency: QMEM, Instruction Cache (IC), Data Cache (DC) and the CPU's General Purpose Register (GPR). These are depicted in the CPU block diagram in Figure 3.

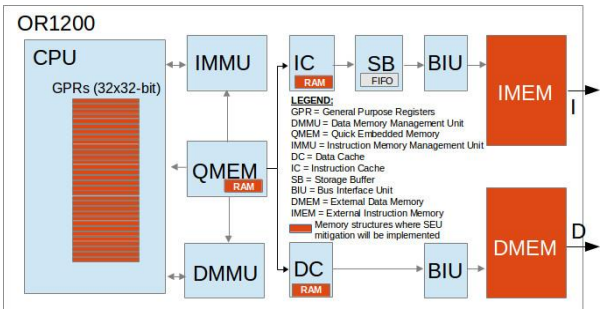

Figure 3. High-level block diagram of the OpenRISC 1200 and its SEE-susceptible memory components.

The components' intolerance to latency classification was decided based on 1) the impact that additional clock cycles of latency would have on the overall system performance and 2) the complexity of adding pipeline stages within the CPU internal organization. The components identified are particularly deep into the CPU structure and do not employ handshake mechanisms to transfer data. Rather they rely solely upon a deterministic data transfer mechanism. Adding extra delays in these components will not only be a complex task, but will have a greater impact on the system performance than other components where data transfer mechanisms account for extra delay. The impact on the system's overall performance was also evaluated by how often such memory components are accessed. Each of the four (4) components identified are continuously accessed (read/write) throughout the system's operation. This continuous and frequent access prohibits us from adding additional latency under penalty of affecting the system's performance. However, continuous and frequent access to these components means its contents are refreshed often. This can be used as a naturally occurring memory scrubbing mechanism that would significantly reduce the chances of error accumulations (multiple bit errors happening in a single word or datapath).

Each of the four (4) components identified have the additional characteristic of being small in size. Both of these factors suggest the use of simpler error correction codes to mitigate the probability of upset bits in these memory components propagating to the rest of the system. Mitigation methods such as parity, cyclic redundancy check (CRC) or error detection and correction codes (EDAC), can be used.

In the case of GPRs, EDAC is preferred in order to always read a correct answer from these registers. No write-back or scrubbing is necessary (which would add latency) as data in these registers is usually short-lived (meaning they are overwritten often).

In the case of the data and instruction caches, the user could pick between EDAC and CRC. EDAC is more costly in terms of logical resources, but will allow the caches to always produce the correct data (by detecting and correcting errors). CRC is less costly in terms of logical resources, but could only signal the processor when an error is detected, without being able to correct it. However, the cache has mechanisms available to refresh its contents on demand. The refreshing mechanisms can be triggered when an error is encountered, at a performance cost (i.e., the time that it takes to refresh its contents). Thus in the case of the caches, the user has a classical "resources vs. performance" trade-off decision to make.

Based on this analysis a solution using extended Hamming code (single-error correcting and double-

error detecting - SECDED) was implemented. Since the core implementing the SECDED is dependent (in size) on the memory it is protecting, three different cores that include encoder and decoder parts were created to protect the GPRs, QMEM, and IC/DC (see Figure 3).

In order to ensure that all memories would function normally with no added latency, combinational logic was used in each module. Also, no scrubbing was implemented with the same goal of avoiding additional latency. In the case of the cache memory blocks, an error detection signal is being used to force the flushing of the cache, effectively scrubbing any error.

SEE Mitigation on latency tolerant components

Data and instruction memories (RAM) are latency tolerant components in the sense that a handshake is used to access them, and extra latency can be tolerated (See Figure 3). The on-chip RAM modules have two particularities that set them apart from the memory blocks that have already been protected by EDAC (error detection and correction) codes. The first particularity refers to their capability to handle additional latency if necessary. These memory cores are connected to the microprocessor systems through a bus with handshake, meaning an additional clock cycle of latency could be handled by the bus protocol. It would obviously have an impact on the overall throughput and performance of the system. The second particularity relates to the on-chip RAM byte addressable feature. In spite of having a 32-bit wide data bus, these memories can be addressed byte by byte. This feature introduces some complexity to the implementation of error correction and detection codes. The difficulty lies in being able to encode and decode the error correction codes that are protecting a 32-bit wide word when only a byte is being read out or written. Several alternatives were researched to quantify their efficiency in terms of performance and resource consumption for the case the different FPGA targets considered. A proprietary method was implemented to overcome the difficulty described while reducing the impact in the microprocessor's performance.

Triple Modular Redundancy

This approach was considered for peripherals attached to SHARP's central WishBone bus. The versions of SHARP currently being tested do not include this approach for any of the basic peripherals included in the distribution. But the hooks and means to instantiate a wishbone interface that supports TMRed peripherals will be included in the final distribution for the user to decide on its implementation in case of using commercial devices which will be vulnerable to Single Event Effects (SEE). Although an effective method to deal with SEE (masking error in the most simple approach), TMR has to be carefully implemented for it to have a positive effect in the overall reliability of the system [12]. The provisions available in the current internal release of the soft-cores implements what is known as localized TMRed. Other approaches should be considered by the user depending on the complexity and criticality of the system being designed.

CONCLUSIONS

This paper describes the architecture of a 32-bit radiation hardened processor for space applications that will have both ASIC and FPGA implementations. Radiation hardening is accomplished by design in the ASIC and by using radiation hardened FPGA devices. Additional architectural modifications were added to improve the system resilience to SEE when implemented in non-radhard platforms. SHARP is expected to be available as early as Q1-2018.

ACKNOWLEDGMENTS

This material is based on research sponsored by the Air Force Research Laboratory under agreement number FA9453-16-C-0413.

REFERENCES

1. CubeSat Kit™ FM430 Flight Module Datasheet, http://www.cubesatkit.com, last retrieved on 06/06/17
2. D. Rudolph, C. Wilson, J. Stewart, et all, "CHREC Space Processor"; 28th Annual AIAA/USU Conference on Small Satellites, August 2014.
3. Vorago's VA10820 Radiation Hardened ARM® Cortex®-M0 MCU Datasheet, http://voragotech.com/products/VA10820, last retrieved on 06/06/2016
4. AAC Microtec's OpenRISC-FT, http://forum.aacmicrotec.com/index.php/Openrisc-ft, last retrieved on 06/06/2017
5. Cobham LEON4 Processor product description, http://www.gaisler.com/index.php/products/processors/leon4, last retrieved 06/06/2017
6. A. Waterman, K. Asanovic, "The RISC-V Instruction Set Manual", Version 2.2, May 2017, https://riscv.org/specifications/, last retrieved on 06/06/2017
7. Daniel Mattsson and Marcus Christensson; "Evaluation of Synthesizable CPU Cores", Master's Thesis Computer Science and Engineering Program, Chalmers University of Technology, Gothenburg 2004.

8. "CoreMark ® - Pro: CoreMark for High-Performance Processors." EEMBC. http://www.eembc.org/coremark/CoreMark-Pro_intro.pdf, last retrieved on 06/06/2017

9. Celio, Christopher, David A. Patterson, and Krste Asanović. *The Berkeley Out-of-Order* Machine (BOOM): An Industry- Competitive, Synthesizable, Parameterized RISC-V *Processor*. Tech. no. UCB/EECS-2015-167.

10. "OR1200 OpenRISC Processor." *OpenCores*. 11 Dec. 2012. http://opencores.org/or1k/OR1200_OpenRISC_Processor , last retrieved on 06/06/2017

11. Cobham LEON3 Processor product description, http://www.gaisler.com/index.php/products/processors/leon3, last retrieved 06/06/2017

12. Melanie Berg, "New Developments in Error Detection and Correction Strategies for Critical Applications". 2016 Single Event Effects (SEE) Symposium and Military and Aerospace Programmable Logic Devices (MAPLD) Workshop; 23-26 May 2016; San Diego, CA; United States