ISBN: 978-3-95450-238-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2023-M04A002

HydRA: A System-on-Chip TO RUN SOFTWARE IN RADIATION-EXPOSED AREAS

T. Gingold*, A. Arias Vázquez, G. Daniluk, J. Serrano, T. Włostowski, CERN, Geneva, Switzerland
M. Rizzi, PSI, Villigen, Switzerland

Abstract

In the context of the High-Luminosity LHC project at CERN, a platform has been developed to support groups needing to host electronics in radiation-exposed areas. This platform, called DI/OT, is based on a modular kit consisting of a System Board, Peripheral Boards and a radiation-tolerant power converter, all housed in a standard 3U crate. Groups customise their systems by designing Peripheral Boards and developing custom gateware and software for the System Board, featuring an IGLOO2 flash-based FPGA. It is compulsory for gateware designs to be radiation-tested in dedicated facilities before deployment. This process can be cumbersome and affects iteration time because access to radiation testing facilities is a scarce commodity. To make customisation more agile, we have developed a radiation-tolerant System-on-Chip (SoC), so that a single gateware design, extensively validated, can serve as a basis for different applications by just changing the software running in the processing unit of the SoC. HydRA (Hydra-like Resilient Architecture) features a triplicated RISC-V processor for safely running software in a radiation environment. This paper describes the overall context for the project, and then moves on to provide detailed explanations of all the design decisions for making HydRA radiation-tolerant, including the protection of programme and data memories. Test harnesses are also described, along with a summary of the test results so far. It concludes with ideas for further development and plans for deployment in the LHC.

BACKGROUND

The High-Luminosity Large Hadron Collider (HL-LHC) project [1] will increase the luminosity in the LHC in order to provide more frequent collisions in the experiments and maximise their discovery potential. This project brings a number of challenges to the control system of the accelerators. In particular, in the lowest tier, some areas will be exposed to increased levels of radiation, precluding the possibility of installing off-the-shelf electronics. A dedicated work package covers the development of a modular radiation-tolerant platform [2] which can serve as a basis for diverse systems. This allows equipment groups to capitalise on a set of basic building blocks and focus their efforts on their customisation, avoiding unnecessary duplication of developments and increasing overall quality.

* tristan.gingold@cern.ch

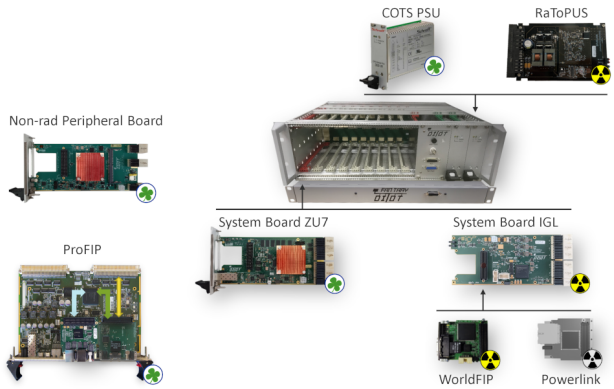

Figure 1: Distributed I/O Tier hardware kit.

THE DI/OT PLATFORM

The Distributed Input/Output Tier (DI/OT) project aims at providing a modular kit for building systems in the lowest tier of the control stack, directly interfacing to accelerator equipment. The basic kit is illustrated in Fig. 1. Modules are housed in a 3U Europa crate featuring a backplane compliant with the CompactPCI Serial (CPCI-S.0) standard. Having a fully passive backplane prevents problems related to radiation in this critical component.

The left-most slot in the crate is reserved for the system board. From that slot, a star topology in the backplane allows communication with the other cards, called peripheral boards. There are two variants of the kit:

- In the radiation-tolerant variant, the power supply is designed in-house and the system board features a flash-based IGLOO2 FPGA.
- In the non-radiation-tolerant variant, the power supply can be purchased off-the-shelf (a bonus of having chosen a standard format for the crate and backplane) and the system board is based on a Xilinx Zynq Ultrascale+ SoC.

The DI/OT platform does not aim to replace well-established solutions such as Programmable Logic Controllers (PLCs) and modular electronics platforms based on a bus (VME, uTCA, PXIe…). Instead, it focuses on two use cases not well covered by these platforms:

- Electronics exposed to radiation.
- Systems in which the connectivity among boards is fully custom for a given application. The fully-passive backplane of DI/OT and the configurable nature of the system board allow e.g. using some of the copper lanes in the backplane to directly stream ADC data from a peripheral board, connect interlocks, etc.

In the remainder of this article, we will focus on the

ISBN: 978-3-95450-238-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2023-M04A002

radiation-tolerant system board and the HydRA SoC implemented in its FPGA. The system board features an IGLOO2 flash-based FPGA and an FMC slot to host a communication mezzanine. The nanoFIP FMC (see Fig. 1) is based on a Microchip ProASIC3 FPGA and implements a radiation-tolerant agent for the WorldFIP fieldbus. For applications which need control of a rad-tol DI/OT system from a PLC, the ProFIP card implements seamless translation between Profinet and WorldFIP.

ISSUES WITH RADIATION

We need to protect electronics against two types of radiation-induced effects [3]:

- Dose effects which accumulate through time and eventually lead to device failure. Protection is typically achieved through a judicious choice of components. Our target is being able to withstand more than 250 Gy. This should provide for robust operation in excess of 10 years in most locations.
- Single-Event Effects (SEEs) are random disruptions triggered when a particle goes through the chip. The protections against these effects are the main object of this paper.

The design of the HDL cannot protect against dose effects (total ionising dose and displacement damage). These slowly create defects in material and degrade electronics. However, we carefully select components to choose those which are the most robust and, before producing boards, we also test the components. A dedicated team provides this service at CERN. We also test the behaviour of the system when irradiated by a Co60 source of gamma rays, in order to have a rough idea of the maximum total dose the system can withstand. If the yearly dose rate at the place where the system will be installed is known, we can deduce the lifetime of the system at this location.

The design of the HDL can protect against SEEs. Direct or indirect ionisation due to a high-energy particle can change the value of a gate and thus generate logical errors. The classical protection is Triple Modular Redundancy (TMR), which consists of triplicating a module and using a voter to decide the value. As it is very unlikely that two modules are hit at the same time, at least 2 out of 3 will give the correct result and the voter can detect and correct the error. The voter itself is often triplicated to give 3 correct results as inputs to other triplicated modules.

Triplicating the logic takes a lot of resources: at least three times the number of gates, without counting the logic to implement the voters. Triplication also reduces the maximum operating frequency.

To reduce the overhead, we can limit triplication to the flip-flops, leaving out the combinatorial logic. The reason is that Single Event Transients (SET) in the logic are rare and have an effect only if they are captured by a flip-flop. If they are not captured, the system self-corrects at the next clock cycle. On the other hand, errors in flip-flops (Single Event Upsets, SEU) cannot correct themselves, so they require

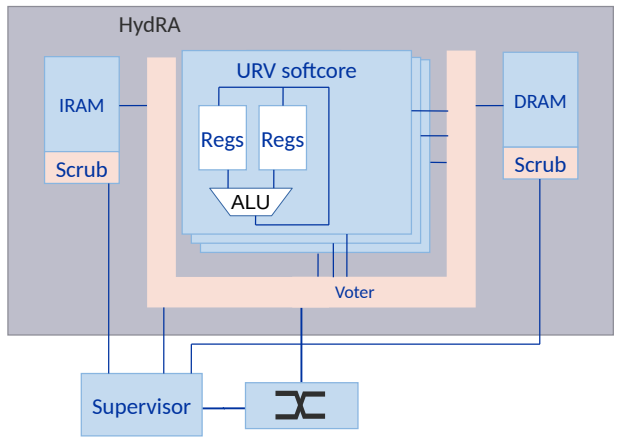

Figure 2: CPU Architecture.

TMR protection.

In HydRA, we do not triplicate RAM (block memory). RAM is a scarce resource and because only the data lines being accessed are activated, TMR may not be effective enough as errors can accumulate in non-accessed lines. The usual technique used to protect RAM is based on error codes where the data bus is extended to store such a code which can be checked when a word is read. A simple error code like parity can be used to detect a single error in a line, but the error cannot be corrected, nor can a double error be detected. A more powerful error code such as Error Correction Code (ECC) can be used to detect a single or double error and to correct a single error. To make it effective and avoid accumulation of errors, it is necessary to periodically read all the memory and correct the errors. This is implemented by a scrubber.

HydRA

The software running on HydRA is bare-metal (no operating system) code, without hardware interrupts, as simple as possible and fully deterministic. After initialisation, it runs an infinite loop, reacting to events by polling.

CPU Architecture

The core of HydRA is a RISC-V compatible CPU. We used our own core, named uRV [4]. It is a classic 5-stage core, supporting only integer operations. In order to simplify and reduce the size of the core, the optional multiplication and division instructions are not enabled. Hardware interrupts are also disabled.

Instruction memory is separated from data memory (see Fig. 2). This simplifies the HDL design and prevents accidental corruption of instructions in case of incorrect writes. There is no pre-initialised data; the whole data memory is cleared by hardware during reset. It is possible to read data from the instruction memory, but not to modify it. Thanks to this architecture, it is always possible to restart the software running on a CPU, provided that software contains initialisation routines at the start to take the hardware to a known

ISBN: 978-3-95450-238-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2023-M04A002

state.

The CPU has access to a supervision unit, which is responsible for managing the health of HydRA: it counts the number of errors, and parametrises the memory scrubbers.

The whole design, including the CPU, is triplicated. There is an option in the synthesis tool to enable TMR but the tool only implements flip-flop triplication. It is important to understand that memory blocks (RAM) are not protected with this TMR option.

Both instruction and data memories have therefore been designed with ECC protection. There are 7 extra bits of error code per 32 bits of data. When a word is read from memory, the error code is checked. In case of a single error (which can be either in the data or in the extra bits), the whole protected word is corrected and rewritten in the memory before being sent to the CPU. In case of double error, this is considered a fatal failure and the CPU is reset. For data RAM, an extra mechanism is required to support partial writes (like writing a single byte in a word). In order to correctly update the error code, the whole word must be read, possibly corrected in case of single error, then modified and written with the new error code.

There is one important detail that needs to be addressed. The CPU contains a register file which contains the data of each register that implemented using a RAM. So it is not protected by the TMR.

We have two different implementations to protect the register file.

The first one uses a manual TMR on the whole CPU where the core is instantiated 3 times and voters are added which compare the outputs of the three instances (their data and instruction buses). If the voters detect that the output of one CPU is different from that of the other 2, that CPU is marked as defective and reset. The two other CPUs are now working in lock-step mode and any difference between them is considered a failure. In order to fall back to a fully protected situation, the software resets the three CPUs so they get completely resynchronised. The monitoring of their outputs resumes and the system goes back to normal operation.

This implementation is not very efficient as the logic of the CPU core is triplicated and the flip-flops are replicated 9 times (3 times by the TMR option and 3 times by the manual triplication). In this implementation, there is no hardware scrubber for the register files. The software implements manual scrubbing, periodically reading the flip-flop-based registers and writing their contents into RAM. If there is any divergence in the contents of the registers among the three CPUs, it will be seen as a difference in their data buses and resynchronised through a reset of the three CPUs as described above.

The second implementation was designed later and is smaller. It is based on ECC, and takes advantage of the fact that our uRV implementation has 2 identical copies of the RAM-based register file because an instruction can read up to 2 registers during execution in a single clock cycle. The register files are extended to store error codes, which are computed on stores and checked on reads. In case of a single error, a software exception handler is executed instead of the faulty instruction to correct the error. We take advantage of the RISC-V architecture which allows implementation-defined instructions. The added instruction takes the contents of the register in the uncorrupted register file and writes them to the corrupted one so the error is corrected.

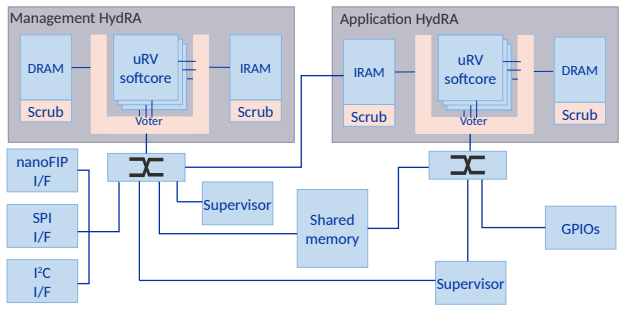

Figure 3: System Architecture.

After power-up, the data in the memories are in a random state, so reading any word will result in an error. A state machine fills the whole instruction memory and clears the whole data memory so that all words are in a coherent state. After that, the CPU initialises all the registers, writing zeroes plus the appropriate ECC. Once this initialisation is complete, the user code can run.

System Architecture

The FPGA contains two HydRA CPUs: one for the management and one for the user. See Fig. 3. The first HydRA CPU is used for the board management. It has three main purposes:

- It periodically scans sensors on I2C buses to monitor the board power state, temperature, the state of the fans and the RaToPUS radiation tolerant power supply. The values are stored in RAM and can be collected by the remote host. As some sensors are not designed to be radiation-tolerant, in case of timeout or incoherent values, the management CPU can power-cycle a sensor. This allows recovery from any SEE.
- It handles communication through the nanoFIP mezzanine. WorldFIP packets can be interpreted as either a command or user data. The latter are forwarded to the application CPU through a shared memory. Commands cover management tasks such as controlling the I2C devices in the crate, performing tests and reading statistics registers.
- It manages the application CPU. In particular, it can download an application either from the local flash or from WorldFIP, and start or stop the application CPU. As a safety measure, the management CPU cannot modify the program run by the application CPU while it is running. The registers in the supervision unit attached to the application CPU can be read by the management CPU and reported to the host.

At power-up, the instruction RAM of the management

ISBN: 978-3-95450-238-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2023-M04A002

CPU is loaded from a flash by a small state machine, and then the reset line of the CPU is released. A remote software update is possible because the management CPU can write to this flash. The program is currently about 1000 lines of C code, occupying 4 kB in memory.

The second HydRA CPU is dedicated to the user. It has exclusive access to the I/O lines on the backplane and cannot be interrupted by the management CPU. Each CPU can be restarted without affecting the other one. Users can add peripherals (e.g. I2C or SPI controllers) to the user CPU by connecting them to its Wishbone bus. The software running in the user CPU must also be an infinite loop, reacting to data coming either from the memory shared with the management CPU, or peripherals on the Wishbone bus.

Verification

The most complex part of HydRA is the home-made RISC-V core. It was tested using the official RISC-V instruction testsuite which exhaustively tests all instructions. Furthermore, we use the smallest instruction subset, and we don't use interrupts. As we mentioned above, we have two HydRA designs under evaluation. The first one (triplicated CPU) doesn't use exceptions. The second one uses exceptions only to correct ECC errors in the register file. In addition to the testsuites, the CPUs have been used and validated in some previous projects: WRTD ([5]) and the White-Rabbit PTP core ([6]) where the software is quite large (about 128 kB of instructions) and uses interrupts. We haven't found any issue in the uRV core for at least 5 years, so given the restricted set of features used we have good confidence in this core.

The I2C and SPI cores are reused from the general-cores library ([7]) in the Open Hardware Repository. Thanks to the experience gathered in many projects, they are considered validated.

The register map has been automatically generated [8] and therefore doesn't require a particular verification.

For simulating the design, we have written a small testbench in VHDL. Unfortunately, it is not fully accurate as we use the flash storage block of the FPGA. We have no model for this block, and we had to write a limited model from the documentation. It's difficult to know how accurate our model is, but fortunately flash storage is a non-critical part: it is used only during initialisation to load software into RAM. To avoid any surprise, we don't use any HDL blocks from the vendor: all the gates in the netlist come from our HDL. In particular, the interface with the flash storage doesn't use the standard one provided by the vendor.

The simulation model was not extensively used. The startup is a little bit slow to simulate, as the software has to be loaded in the CPUs. In addition, we had access to prototypes of the system board, so we preferred to do verification on the real platform. However, if we find an issue, it is always possible to reproduce a scenario in simulation in order to have access to all the signals. To be able to simulate the model with open-source HDL simulators, we used an open-source synthesis tool to translate the instantiated uRV into VHDL.

For functional tests, the host program is able to have access to all devices on the board through WorldFIP. Although the host program is written in C, it can provide a command line interface over a socket and we used a Python script to run the test cases.

Error Injection In addition to functional tests, we need to run tests for all the features which fight against the radiation-induced effects. This is different from the functional tests because by default, in a normal environment, these features are never exercised. A traditional method is to add error injection features: extra logic which allow corruption of the main logic. So in each memory, which is in principle protected by ECC, the developer also has the possibility to write an incorrect ECC. By observing the supervisor counters, we can check if the error has been corrected (in case of a single bit error), or has been detected (in case of double bit error). The detection and correction mechanisms can be either the scrubber (which scans the whole memory), or the CPU when it reads or writes the memory. In order to specifically check the CPU mechanism, we need to disable the scrubber.

Writing an incorrect ECC to a data memory can only be done by the CPU attached to that memory. So a specific application needs to be loaded into the application CPU to do the tests. As only the management CPU can write into the instruction memory of the application CPU, it drives the test. The application CPU has to be on reset to allow changes of the instruction memory. Corruption of the instruction memory of the management CPU is done during boot, while the memory is filled from the flash. A specific boot flag allows a few selected instructions to be corrupted.

The error injection scheme described above is used for both variants of the HydRA design. For the first variant, we are also able to make the CPUs diverge: there is one special register in the memory map which returns a different value for each of the triplicated CPUs. Depending on how the value is used, different types of faults can be injected:

- the address sent to the instruction memory can be different (by executing a conditional jump on the value),
- the address sent to the data memory can be different (by reading or writing a word at an address which depends on the value)
- the data written can be different (by writing the value)
- the byte select mask can be different (by addressing a byte within a given word depending on the value)

For the second design variant (the one with ECC in the register file), another instruction has been added in order to be able to write into the ECC bits of the destination register. This allows us to inject errors in these bits.

Test Results

The total specified ionisation dose for the DI/OT radiation-tolerant system is 25 Gy per year for a lifetime of 10 years.

The system was first tested in the CERN Co60 facility. In this environment, the system is continuously working

ISBN: 978-3-95450-238-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2023-M04A002

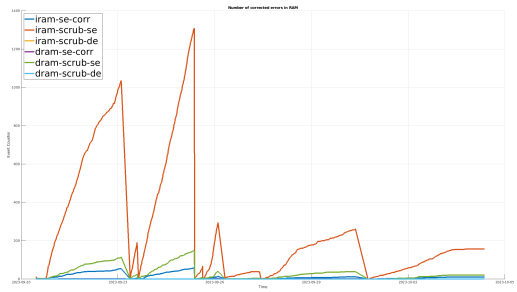

Figure 4: Number of corrected errors in RAM as a function of time.

without any fault but the electronics slowly degrade. During our tests, the system stopped working at 335 Gy. We then tightened the design constraint for the maximum operating frequency to further improve the results.

Then the system was tested in CERN's CHARM [9] facility, where a proton beam hits a target, which generates secondary particles at relatively high energy. The profile of this indirect radiation over our board is comparable to the real situation in LHC, where electronic systems are not directly hit by the beam.

The dose rate in CHARM depends on the position of the system, the nature of the target and the energy of the beam. A radiation sensor (RadMon [10]) is placed very close to the system. The board is remotely powered and connected through WorldFIP to a host. The host runs a test script (written in Python) which continuously communicates with and monitors the system. It reads the values from the sensors (gathered by scanning the I2C buses), the values from the supervisor registers and well as some counters maintained by the software (number of resets, number of packets, cycle number...) and logs them for offline analysis. The application CPU runs a simple program which bounces back the random patterns sent by the host program.

From the logs we can extract graphs like Fig. 4, where we can clearly see the number of (corrected) single errors increasing when the beam was on. "iram" in the figure stands for the instruction RAM, while "dram" is the data RAM. "se-corr" are errors corrected when the CPU accesses the memory and "scrub-se" corresponds to errors fixed by the hardware scrubber. The "scrub-de" counter represents double errors detected by the scrubber and was always zero during our tests, indicating that there was no double error generated. The curves restart from zero when the board is reset. We can see that the number of single errors is proportional to the size of the RAM (the instruction RAM is 4 times the size of the data RAM). The scrubbers are able to scan the whole memory many hundreds of times per second, so that single errors do not accumulate.

The system continued working correctly until it died of dose effects at 500 Gy. This is a sign that all SEEs in the logic part are correctly detected and corrected by the TMR and other methods described.

OUTLOOK

Some CERN teams have already started to use the radiation tolerant DI/OT system for new developments: WPS – the Wire Positioning System – and WIC – the Warm magnet Interlock Controller – validating the basic building blocks of the kit.

Concerning HydRA, there are some additional features that could be implemented. The uRV CPU could support the compressed set of instructions, thus reducing RAM size for the program, or allowing larger programs. Also, in order to help during the development phase, it could be possible to support remote debugging. uRV already supports it, but it is not yet integrated in HydRA.

Until now, execution speed has not been an issue for users. The current clock frequency of 40 MHz after TMR required no special efforts, so it should be possible to increase it. For the 2 CPUs, we use less than 45 % of the FPGA logic, and less than 20 % of the RAM.

Finally, it could be interesting to use open-source tools to implement different TMR schemes, e.g. triplicating also the combinatorial logic (and maybe monitoring it) to see if it improves the reliability.

ACKNOWLEDGEMENTS

The authors gratefully acknowledge the help of the CHARM team at CERN during the preparation and the running of the radiation tolerance tests for HydRA.

REFERENCES

[1] https://hilumilhc.web.cern.ch/

[2] G. Daniluk, C. Gentsos, E. Gousiou, L. Patnaik, and M. Rizzi, "Low-Cost Modular Platform for Custom Electronics in Radiation-Exposed and Radiation-Free Areas at CERN", in *Proc. ICALEPCS'19*, New York, NY, USA, Oct. 2019, pp. 672. doi:10.18429/JACoW-ICALEPCS2019-TUAPP03

[3] Radiation Handbook for Electronics. https://www.ti.com/seclit/eb/sgzy002a/sgzy002a.pdf

[4] https://ohwr.org/project/urv-core/wikis

[5] https://gitlab.cern.ch/be-cem-edl/chronos/wrtd

[6] https://ohwr.org/project/wr-cores

[7] https://ohwr.org/project/general-cores

[8] https://gitlab.cern.ch/be-cem-edl/common/cheby/-/wikis/home

[9] https://kt.cern/technologies/charm-mixed-fields

[10] G. Spiezia and Al. "The LHC radiation monitoring system - RadMon", in *Proceeding of Science*, vol. 143, 2011, p. 024. doi:10.22323/1.143.0024