Fault-tolerant satellite computing with modern semiconductors

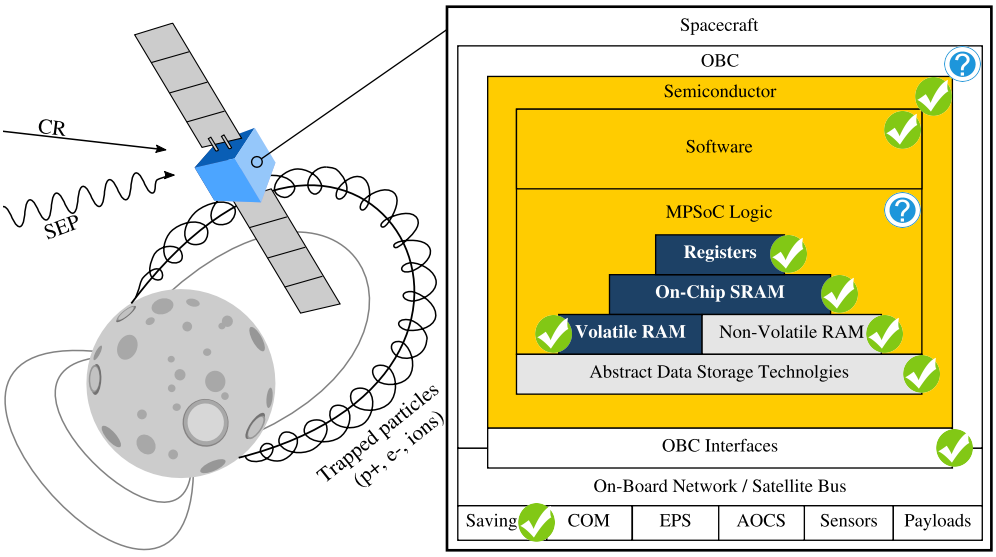
Fuchs, C.M.

Chapter 10

On-Board Computer Integration and MPSoC Implementation

Practical Design Verification on FPGA

In this chapter, we present a practical implementation results for our MPSoC design, as just making up fault tolerance concepts would be insufficient to answer RQ6. We show that this on-board computer architecture in its full functionality can be implemented with a low component cost, and only with standard development tools and IP. We achieve 1.94W total power consumption, which is well within the power budget range achievable aboard 2U CubeSats and larger satellites. This serves as proof-of-concept for our architecture and answers RQ6, paving way to radiation testing and on-orbit demonstration in the future.



10.1 Introduction

Cheap, CTOS electronics designed for the embedded and mobile-markets are the foundation of modern nanosatellite design. They offer an excellent combination of low energy-consumption, minimal cost, and broad availability. However, such components are not designed for reliability, and include only rudimentary fault tolerance capabilities. Due to the elevated risk of loosing a satellite due to failure of these components, CubeSat missions today are kept brief or up-scaled to larger, more expensive satellite form factors.

Low-complexity, low-performance satellite on-board computer (OBC) designs have allowed a variety of successful CubeSat missions, with a few missions even operating successfully for as long as 10 years. This demonstrates that there is no fundamental, hard technological barrier that could prevent the use of modern semiconductors in space missions. However, these designs are sufficient only for missions with very low performance requirements, e.g., for educational missions and brief technology demonstration experiments.

Many sophisticated scientific and commercial applications can today also be fit into a CubeSat form factor, which make a much longer mission duration desirable. To fly these payloads, a CubeSat has to process and store drastically more data, and at all levels requires increased performance. Therefore, all advanced CubeSats today utilize industrial embedded and mobile-market derived systems-on-chip (SoC), which offer an abundance of performance. However, these SoCs in turn are manufactured in modern technology nodes with a fine feature size. They are drastically more susceptible to the effects of the space environment than simple but robust low-performance microcontrollers. Hence, proper fault tolerance capabilities are needed to ensure success for advanced long-term CubeSat missions, as gambling against time and radiation can be risky.

Radiation hardening for big-space applications can not be adopted, as this approach is only effective for very old or very proprietary and costly manufacturing processes. Budget, energy, and size constraints prevent the use of traditional space-grade components used aboard large satellites, while component-level fault tolerance significantly inflate CubeSat system complexity and failure potential. Today, no fault-tolerant computer architectures exist that could be used aboard nanosatellites powered by embedded and mobile-market semiconductors, without breaking the fundamental concept of a cheap, simple, energy-efficient, and light satellite that can be manufactured en-mass and launched at low cost. Hence, we developed a scalable, yet simple OBC architecture that allows high-performance MPSoCs to be used in space, and is suitable for even small 2U CubeSats.

Our proof-of-concept OBC utilizes Microblaze processors on a low-power FPGA, exploits partial reconfiguration and software-implemented fault tolerance to handle system failure. It is assembled only from COTS components available on the open market, standard vendor library IP, and runs standard operating system and software. To protect our system, we utilize a combination of runtime reconfigurable FPGA logic and software-implemented fault tolerance mechanisms, in addition to well understood and widely available EDAC measures. We facilitate fault tolerance in software, which enables our system to guarantee strong fault coverage without introducing the hard design limitations of traditional hardware-TMR based solutions.

Our OBC architectures can efficiently and effectively handle permanent faults in

the FPGA fabric by utilizing alternative FPGA configuration variants. It ages gracefully over time by adapting to an increasing level semiconductor degradation, instead of just failing spontaneously. The performance of the OBC itself is adjustable, allowing spacecraft operator to modify system parameters during the mission. An operator can trade processing-capacity and functionality to achieve increased fault coverage or reduced energy consumption, without interrupting satellite operations. Thereby, we can maintain strong fault coverage for missions with a long duration, while adjusting the OBC to best meet the requirements of complex multi-phased space missions.

To our understanding, this is the first scalable and COTS-based, widely reproducible OBC solution which can offer strong fault coverage even for 2U CubeSats. We provide an in-depth description of our proof-of-concept MPSoC, which requires only 1.94W total power consumption, which is well within the power budget range achievable aboard 2U CubeSats. In the next section, we provide a brief overview over the status-quo in fault-tolerant computer system design for large spacecraft, CubeSats, and ground use. Subsequently in Section 10.3, we describe our OBC's component-level architecture, the MPSoC used, as well as the interplay between the different components of the OBC. Before providing conclusions, we present our implementation results and details about how this MPSoC was tested and validated in Section 10.5. Finally, we discuss advanced applications of our proof-of-concept with multiple FPGAs, Network-on-Chip usage and resistance to full-chip SEFIs in Section 10.4. All components required to re-implement this OBC design are available at low cost to scientists and engineers in an academic environment. The necessary IP and standard design are available free of charge from the relevant vendors, e.g., through Xilinx's university program for academics and scientific users.

10.2 Related Work

In contrast to the initial generation of educational CubeSats, today fewer satellites fail due to practical design problems caused by inexperience [39]. Instead, Langer et al. in [2] showed that a majority of these failures can be attributed to electronics heavy subsystems. Even experienced, traditional space industry actors with years of experience in large satellite design, who develop CubeSats satellites "by the traditional book" with quasi-infinite budgets today struggle to reach just 30% mission success [42].

The main source of failure are environmental effects encountered in the space environment: radiation, thermal stress, and corruption of critical software components that can not be recovered from the ground, and failures caused by power electronics. Considering again Langer et al., [2], with increasing age mission duration, a broad majority of documented failures aboard CubeSats originate from OBCs, transceivers, and the electrical power subsystem. While functionally disjunct, these subsystems all have in common that they are heavily computerized and architecturally rather similar, built around one or multiple microcontrollers and memories.

Fault tolerance concepts targeting generic commercial ground-based computing applications usually cover only a small subset of our fault model: transient faults, material aging, and occasionally gradual wear. Such assumptions are valid for critical applications for ground applications, but not for space applications. Often, the introduction of permanent faults breaks fault tolerance concepts for ground applications, weaken their protective capabilities strongly, or limit their protection to only a brief period of time. Most ground-based and atmospheric aerospace fault tolerance

concepts also aim to guarantee reliable operation from the point in time a fault occurs until maintenance can be performed. This is a problematic assumption for CubeSat use, as servicing missions have only been performed on rare occasions for spacecraft of outstanding scientific, national, and international significance such as the International Space Station or the Hubble Space Telescope. But certainly not for low-cost CubeSats.

These limitations, however, by using a combination of different additional fault tolerance measures across the embedded stack. Fault tolerance concepts for ground and atmospheric aerospace applications can therefor serve as building blocks to design a fault-tolerant architecture for space applications.

10.2.1 Fault Tolerance for Large Spacecraft

Traditional OBCs for large satellites realize fault tolerance using circuit-, RTL- [344], IP-block- [104, 132], and OBC-level TMR [90] through costly, space-proprietary IP. They make heavy use of over-provisioning and tries to include idle spare resources (processor cores, components, memory, ...) where necessary. Naturally, this is done at the cost of performance and storage capacity, increases system complexity, and power consumption. Circuit-, RTL-, and core-level measures are effective for small microcontroller-SoCs [88,345], if they are manufactured in large feature-size technology nodes. More and more error correction and voting circuitry is needed to compensate for the increased severity of radiation effects with modern technology nodes [345]. This in turn again inflates the fault-potential, requiring even more protective circuitry, making this approach ineffective for modern semiconductors.

Processor lockstep implemented in hardware lacks flexibility, limits scalability, and is feasible only for very small MSoCs with few cores [88, 346]. Timing and logic placement becomes increasingly difficult for more sophisticated processor designs, and becomes infeasible for SoCs running at higher clock frequencies. Practical applications run at very low clock frequencies [347] with two or three very simple processor cores, even for ASIC implementations [88, 132]. Common to all these solutions is that they are proprietary to a single vendor, implying a hefty price tag and tight functional constraints. Especially the space-proprietary single-vendor solutions available are often difficult to develop for, have in many cases no publicly available developer documentation, have no open-source software communities which could provide support in development, and usually imply vendor lock-in into a walled garden ecosystem.

To design nanosatellites, we instead utilize the energy efficient, cheap modern electronics [41], for which traditional radiation-hardening concepts become ineffective. Specifically, CubeSats utilize COTS microcontrollers and application processor SoCs, FPGAs, and combinations thereof [40, 41]. Some of these were shown to performing well in space, and others poorly. On-orbit flight experiences varying drastically even between different controller models of the same family and brand [39]. Specifically, components that were discovered to perform well are very simple microcontrollers with a minimal logic footprint and low complexity. These are manufactured in coarse feature-size technology nodes, and were by coincidence designed to be rather tolerant to radiation (radiation-hard by serendipity) [46]. Examples of such parts are the PIC controller family, which are logically extremely simple, and controllers that include inherently radiation-tolerant functionality such as the Ferroelectric RAM (FeRAM) [332] based MSP430FR family [225]. Unfortunately, these “well behaved” components also

offer very limited performance, which is sufficient only for simple educational missions, technology demonstration, and short low-data rate science missions.

Computer designs for nanosatellites utilized about 10 years ago began to heavily utilize redundancy at the component level to achieve failover, to provide at least some protection from failure. However, practical flight results show that such designs are complex and fragile, as compared to entirely unprotected ones [39, 41]. Entirely unprotected OBC designs, in turn, may fail at any given point in time. However, today satellite designers are usually forced to simply accept this risk, leaving the hope that a satellite will by chance not experience critical faults before its mission is concluded. Risk acceptance is viable only for educational, and uncritical, low-priority missions with a very brief duration.

10.2.2 Fault Tolerance Concepts for COTS Technology

FPGAs have become popular for miniaturized satellite applications as they allow a reduction of custom logic and component complexity. FPGA-based SoCs can offer increased FDIR potential in space over ASICs manufactured in the same technology nodes [40] due to the possibility to recover from faults through reconfiguration. Transients in configuration memory (CRAM) can usually be recovered right away through reconfiguration [105], while permanent faults may be mitigated using alternative configuration variants. However, fine-grained, non-invasive fault detection in FPGA fabric is challenging [345], and is a subject of ongoing research [239, 240]. Applications thus rely on error scrubbing, which has scalability limitations and covers only parts of the fabric.

Software implemented fault tolerance concepts for multi-core systems were identified as promising already in the early days of microcomputers [131], but was technically unfeasible and inefficient until few years ago. Modern semiconductor technology allows us to overcome these limitations and recent research [348,349] shows that modern MultiCore-MPSoC architectures can theoretically be exploited to achieve fault tolerance. However, these are incapable of general-purpose computing, and instead cover deeply embedded applications with a very specific software structure [241, 350]. They require custom processor designs [348], or programming models which are suitable for accelerator applications [349]. The fundamental concept of software-implemented coarse-grain lockstep, however, is flexible and can be applied, e.g., to MPSoCs for safety-critical applications [348, 351], networked, distributed, and virtualized systems [201].

10.3 A Reliable CubeSat On-Board Computer

A system designed for robustness must avoid single-points of failure and assist in fault-detection. It should also support non-stop operation. Ideally, it should be capable of tolerating the failure of entire block and individual attached component. The OBC architecture presented in this chapter consists of an FPGA and a microcontroller in tandem, which is used for test and diagnostic purposes. Within the FPGA, we implement an MPSoC architecture, which is then made fault-tolerant using software measures, while its robustness is increased using memory EDAC and FPGA reconfiguration.

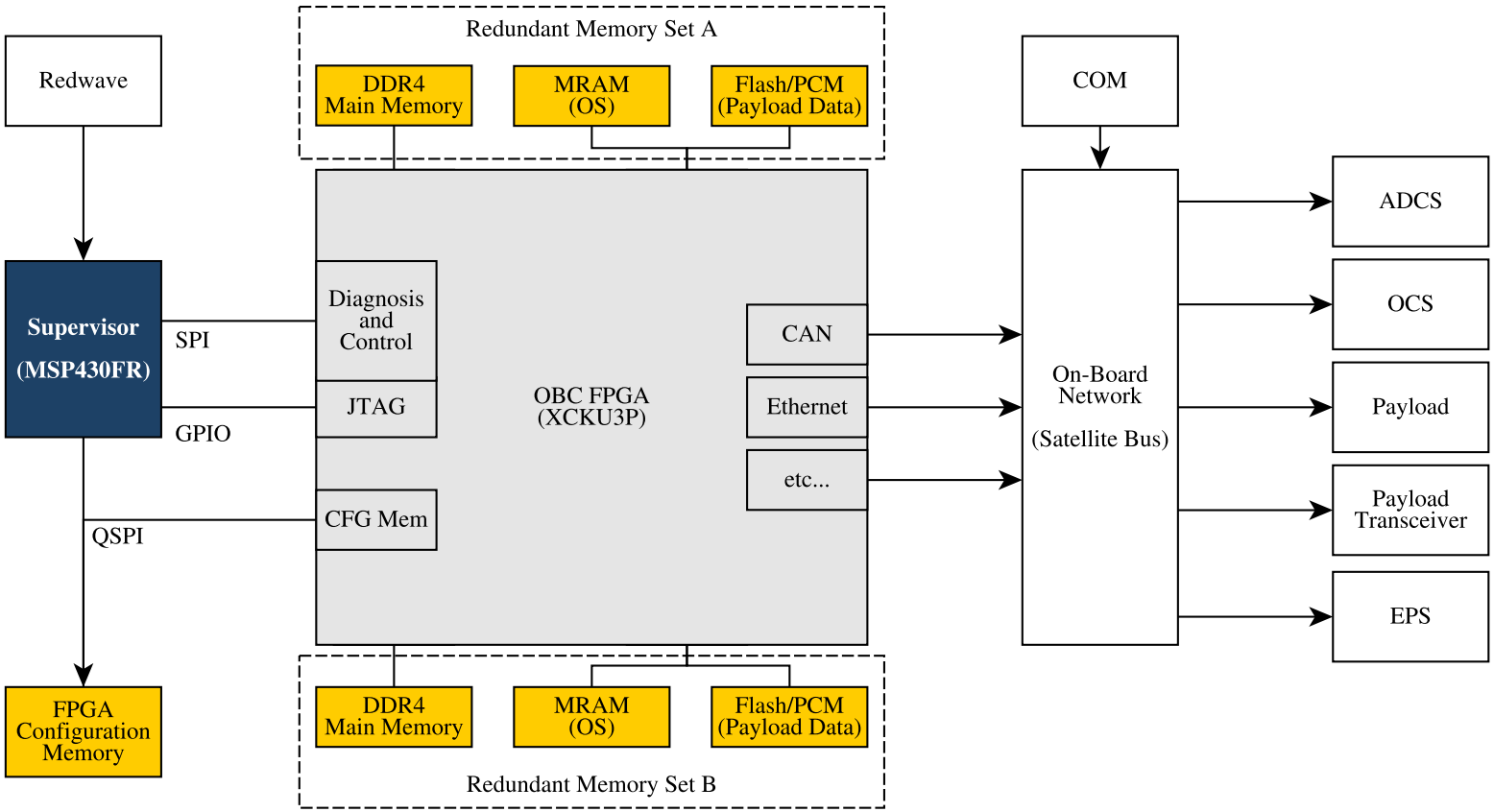


Figure 67: A component-level diagram of our OBC architecture. This architecture is intended as an in-place substitute for a conventional ASIC-based System-on-Chip, and only adds a second set of memory ICs to counter component-level failure.

However, conventional MPSoCs follow a centralist architecture with processor cores sharing functionality where possible to minimize footprint, optimize access delays, improve routing [238]. There, processor cores share memory in full, and have full access to all controllers operating within this address space, to maximize system functionality and code portability. In consequence, conventional high-performance computer designs offer only weak isolation for application running on different processor cores for the sake of performance. Faults in one core may therefore compromise the functionality of other cores and the MPSoC as a whole. This increases the overall failure-potential sharply as compared to very small microcontroller SoCs, as an MPSoC's logic does not have only a larger footprint, but also more components that can independently cause such a system to fail.

From a fault tolerance perspective this is undesirable, and in our OBC we follow a different approach. Designers of fault-tolerant processors for traditional space applications handle this issue by utilizing custom fault-tolerant processor cores, to assure that faults occurring within a core are mitigated and covered before they could propagate. For miniaturized satellite use, this is not feasible, and instead we must achieve fault-isolation and non-propagation through system-, software-, and design-level measures. In the remainder of this section, we show how this can be done with only commodity COTS components and tools that are available to academic CubeSat designers.

10.3.1 System- and Component-Level Architecture

We designed out architecture as in-place replacement for a conventional MPSoC-driven OBC design and utilize a commodity FPGA. The component-level topology of our OBC design is depicted in Figure 67.

We utilize an FPGA to realize an MPSoC that offers strong isolation between the individual processor cores, and to enable recovery from permanent faults. This FPGA serves as main processing platform for our OBC, and capable of running a full general-purpose OS such as Linux. We implemented a proof-of-concept of our OBC architecture using Xilinx Kintex and Virtex Ultrascale+ FPGAs, as well as the earlier generation Kintex Ultrascale FPGAs. For CubeSat use, only Kintex Ultrascale+ FPGAs are relevant at this point due to drastically reduced power consumption as compared to older generation and Virtex FPGAs. We provide further details on this MPSoC in the second to next subsection.

To store the FPGA's configuration memory is attached to the FPGA via SPI. The FPGA by default acts as SPI-master for this memory and automatically loads its configuration from there. In our proof-of-concept implementation, we utilize conventional NOR-flash [153] for this purpose, which also is included on most commercial FPGA development platforms. However, NOR-flash is inherently prone to radiation [153], and phase-change memory (PCM [284]) is much better suited for this task as its memory cells are inherently radiation-immune. Thus, in future applications and in our prototype, we will utilize a PCM IC instead of serial-NOR-flash.

Like most CubeSat OBCs, our OBC includes an additional microcontroller which acts as watchdog, and performs debug and diagnostic tasks. However, as we are utilizing an FPGA as the main processing platform, it only controls the FPGA and the MPSoC implemented within it. Hence, it acts as a saving subsystem (redwave/hard-command-unit), and can resolve failures within the MPSoC its peripheral ICs for diagnostics purposes in case the MPSoC became dysfunctional. To reflect this role,

we refer to it as "supervisor".

As depicted in Figure 71, the supervisor is connected to the FPGA through GPIO and SPI. The SPI interface allows low level diagnostic access to different parts of the MPSoC, as well as facilitate low-level test access to FPGA-attached components. Through the GPIO interface, the supervisor controls the FPGA's JTAG interface and can reset the FPGA as well as different parts of the MPSoC. The FPGA also has access to the FPGA's configuration memory, and shares this SPI bus with the FPGA in a multi-master, so that in case of failure, it can independently reconfigure the FPGA.

The supervisor itself is not connected to other satellite subsystems, and can not control other parts of the satellite beyond the OBC itself. During regular operation, it takes no part in the normal data processing operations of the OBC and only receives correctness information from the MPSoC, which is further described in Chapter 4. However, for failure diagnostics the supervisor can be used to reprogram the OBC FPGA to access the rest of the satellite through its interfaces for debug purposes. Therefore, the supervisor requires very little processing power, and we utilize a robust low-performance MSP430FR5969 microcontroller. The MSP430FR controller family is manufactured with inherently radiation-tolerant FeRAM instead of flash, and has become popular in low-performance COTS CubeSat products due to its good performance under radiation and in space [225]. A space-grade substitute is available in the form of the MSP430FR5969-SP.

10.3.2 Memory Components

Besides the FPGA, configuration memory, the supervisor, and the usual power electronics, our OBC architecture includes two redundant sets of memory ICs for use by the MPSoC implemented on the FPGA. Each memory set includes DDR memory used as main working memory by the MPSOC, magnetoresistive-RAM [150] (MRAM) used to store the operating system and flight software, as well as PCM for holding payload data. In our development-board based proof-of-concept, we are constrained to substituting MRAM and PCM with NAND-flash due to hardware constraints.

DDR-SDRAM is prone to radiation-induced faults [250], though with modern high-density components manufactured in fine technology nodes, the likelihood to experience bit-upsets is low [255, 352]. Hence, for most nanosatellite missions single-bit correcting error correction coding (ECC) [254] is sufficient to protect the integrity of data stored [251] as long as error scrubbing is implemented [353]. In LEO, scrubbing intervals can be kept very low, e.g., once per orbit, as the particle flux and likelihood to receive bit-flips with modern DDR memory is minimal. This can be realized using software-measures as we showed in Chapter 7. ECC can be implemented using standard Xilinx Library IP [331], as well as free open-source cores from OpenCores, and the GPL version of GRLIB. Specifically, standard Xilinx design software out-of-the-box includes the necessary library IP for Hsiao and Hamming coding.

For CubeSats venturing to areas in the solar system with more intensive radiation bombardment, continuous memory scrubbing can be implemented in logic within the MPSoC. Then, stronger EDAC with longer code-words and larger code-symbols should be used, instead of the weaker coding that can be assembled using Xilinx library IP. Symbol-based ECC can compensate better for the effects of radiation in modern DDR-SDRAM: despite occurring less frequently overall, highly charged particles have an increased likelihood to cause multi-bit upsets instead of changing the state of just

a single DRAM cell. EDAC using Reed-Solomon ECC as well as interconnect error scrubber IP cores are available commercially, e.g., via Xilinx or from the commercial GRLIB library. Alternatively, they can be assembled from open-source IP, available from OpenCores, and a broad variety of other open-source code repositories. However, the quality of such cores is often uncertain, and even a good part of the IP available through the curated OpenCores catalog is known to be defunct. Memory scrubbing can be assembled on the FPGA from standard library IP, while ready-made scrubbers are available commercially (e.g., the "memscrub" IP core from commercial GRLIB).

To store the OBC's OS and its data, COTS MRAM ICs are available at low cost on the open market today and flight experience with the parts inside earlier CubeSats has been overwhelmingly positive. However, only the memory cells of these memories are radiation immune. Without further measures, they are still susceptible to misdirected read- or write access, and SEFIs. We showed in Chapter 7 that these issues can be mitigated in software, through ECC, and redundancy. We also showed that this can be achieved with minimal overhead through the use of a bootable file-system with Reed-Solomon erasure coding. FeRAM would be more power efficient than MRAM, and is also inherently radiation tolerant, but its low storage density makes it insufficient for our use-case.

For storing applications and payload data, memory technologies with a much higher storage density than MRAM are necessary. In practice, this limits us to use NAND-flash and PCM, of which only the latter is radiation-immune. The storage cells of both have a limited lifetime, and therefore are subject to wear. However, high-density PCM has not become widely available on the open market, and so we currently have to resort to using NAND-flash. Fault tolerance for these memories can again be realized in software. As both these memories suffer from use-induced wear, the necessary functionality to handle wear is needed to efficiently safeguard their long-term use. Therefore in Chapter 7, we presented MTD-mirror, which combines LDPC and Reed-Solomon erasure coding into a composite erasure coding system.

One of the main causes for failures in commercial memory ICs of all memory technologies are faults in control logic and other infrastructure elements, causing SEFIs [255]. These may cause temporary or permanent failure of memory ICs, regardless of the memory technology used, which can not efficiently be mitigated through erasure coding. Instead, redundancy for these devices is needed, which we can realize by placing two memory sets. However, we do not implement failover in hardware, but merely connect the two memory sets to the FPGA. All failover functionality is realized through the topology of our MPSoC and in software.

10.3.3 The OBC Multiprocessor System-on-Chip

To realize fault tolerance for our OBC architecture, we isolate software run within our OBC as much as possible and without constraining software design. To do so, we co-designed an MPSoC as platform for the software functionality described in Chapter 4. Its logic placement is depicted in Figure 68, and we will describe its composition here.

We place each processor core within a separate *compartment*. Applications and the environment in which they are executed are strongly isolated through the topology of the MPSoC. The MPSoC version described in this chapter has 4 Xilinx Microblaze processor cores, and therefore 4 compartments, which are depicted in brown, green,

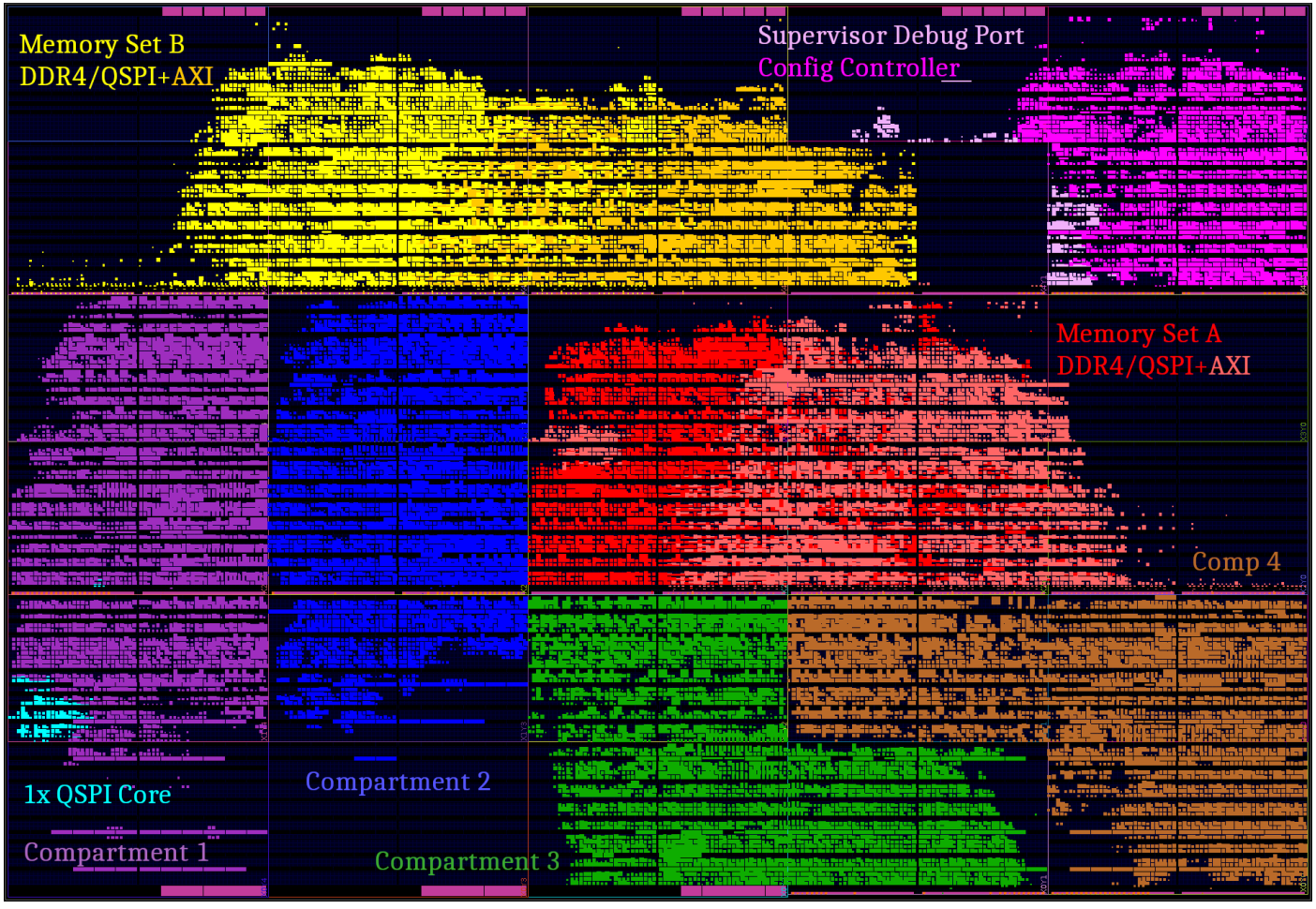


Figure 68: Logic placement of our proof-of-concept quad-core MPSoC for the upcoming XRTC Kintex Ultrascale KU60 device-test board. A QSPI controller is highlighted in teal for size-comparison between an interface core and a compartment's total size.

blue and purple. Compartments have access to two independent memory controller sets through an FPGA-internal high-speed interconnect. The two memory controller sets are depicted in the Figure in red and yellow.

The final, pink-colorized logic segment contains infrastructure IP responsible for FPGA housekeeping, as well as an on-chip configuration controller with access to the FPGA's internal configuration access port (ICAP). As depicted in Figure 69, several MPSoC components related to FPGA housekeeping are placed in static logic:

- the configuration controller makes up only a minor part of the pink-indicated logic,
- the supervisor's debug interface (further described in Section 10.3.4),
- as well as a library IP core facilitating CRAM-frame ECC for the detection and correction errors in the FPGA's running configuration (Xilinx Soft Error Mitigation IP – SEM [354]).

Researchers showed in related work [355, 356] that faults within an FPGA can effectively be resolved through reconfiguration, or mitigated using alternatively routed and placed configuration variants [105]. Usually, full FPGA reconfiguration would interrupt the operation of the MPSoC, and depending on the configuration memory used, can require considerable time. By using partial reconfiguration, we can instead split the MPSoC into separate partitions, which can then be independently reconfigured. The use of an on-chip reconfiguration controller drastically improves the reconfiguration speed, but also allows fine-grained fault analysis and configuration error scrubbing. Multiple alternative partition designs can be provided for each compartment and memory controller set, which can then be reconfigured independently. This not only allows non-stop operation, but also increases the likelihood that a suitable combination of partition variants can be found to mitigate permanent faults present in the FPGA fabric [105].

Compartments and memory controller sets are placed in dedicated partial reconfiguration partitions. Partial reconfiguration allows us to test and repair individual compartments, and to reprogram one memory controller set transparently in the background, without affecting the remaining system. We have implemented this concept in prior research in Chapter 5 for the MOVE-II CubeSat.

Placement in static logic instead of a partition implies that infrastructure logic is not part of any partial reconfiguration partition, which is required both for SEM and logic utilizing ICAP. In practice approximately 90% of the fabric's area is part of the reconfiguration partitions, of which 75% is quadruple-redundant and part of a compartment supporting TMR operation through software. The other 25% of the logic holds the shared memory controllers, which offers simple redundancy and can be recovered transparently using partial reconfiguration. Only 10% of the fabric holds static logic, which can be still be recovered through reconfiguration.

Large clock trees and reset networks are known to be problematic in space applications [357]. The logic in each compartment resides in a separate clock domain, and a memory controller set in 3 – one each for DDR4 backend, memory controller front-ends, and AXI-interconnects. Therefore, clock trees are isolated from each other and are de-coupled on the AXI interconnects of the memory controller sets. This minimizes clock skew and its impact, as well as temperature-related effects, while improving timing and logic routing.

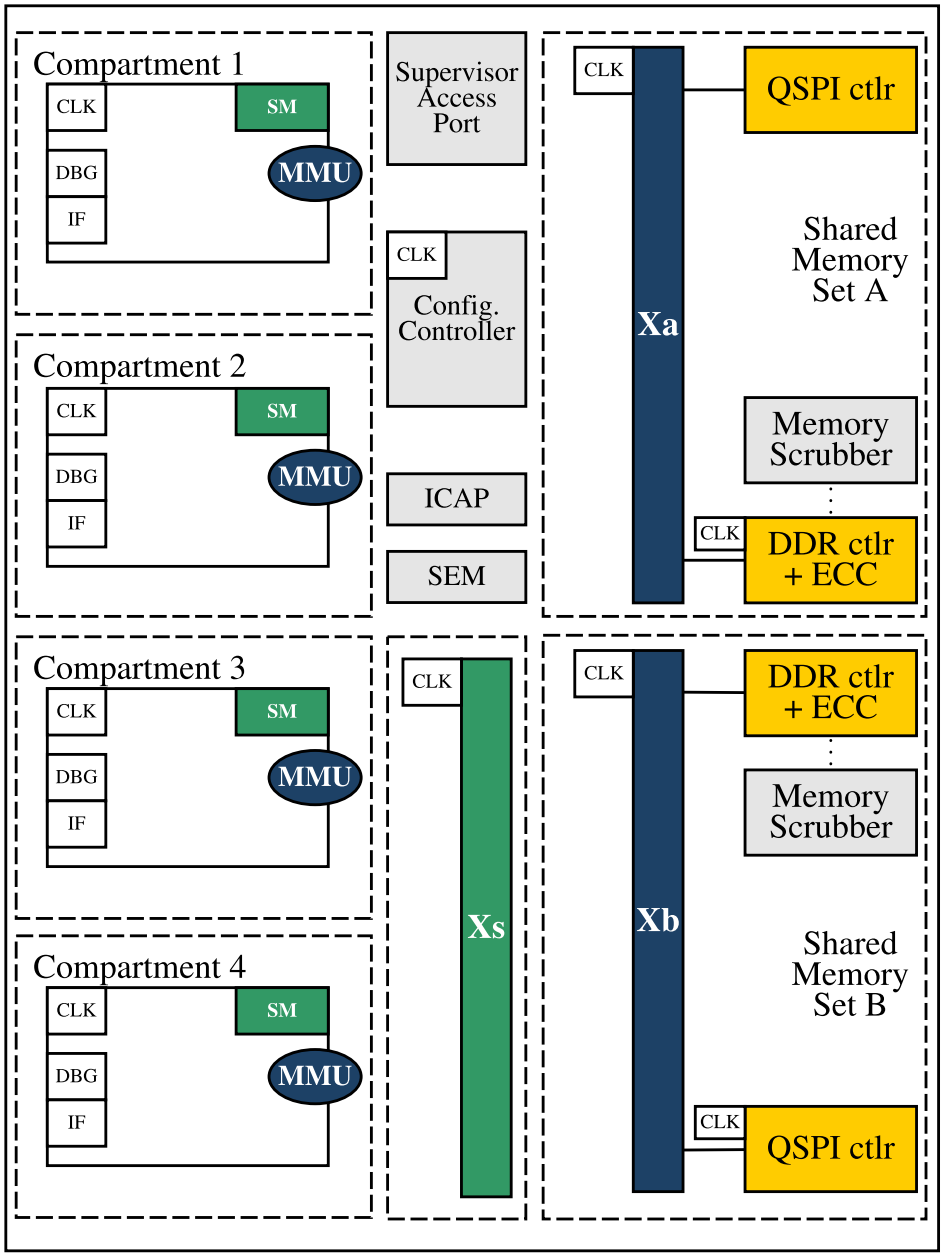


Figure 69: Block-level layout in our MPSoC including clock-placement. Partial reconfiguration partitions are indicated with dashed lines. Compartment and memory controller sets ($X_{a/b}$) can be reconfigured without interruption. The state-exchange interconnect (X_s) resides in a dedicated configuration partition, but during reconfiguration compartments can no access state information. In practice, this results in an interruption of the MPSoC, which can be avoided using a NoC instead of a AXI interconnect.

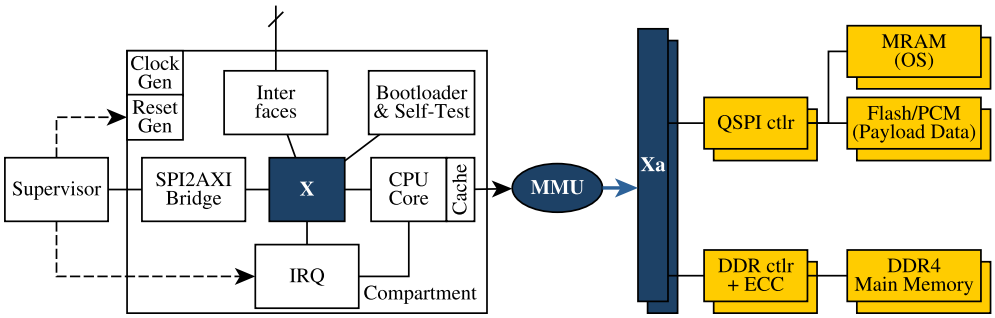


Figure 70: The memory and logical topology of a compartment in a quad-core MPSoC. The compartment local and the global memory controller interconnects are logically isolated. A compartment's processor core has access to the memory controller sets and to compartment-local controllers. Access to compartment-local controllers bypasses the cache.

Compartments are comprised by the minimum set of IP-blocks required for a conventional single-core SoC, including interrupt controller, peripheral controllers, I/O, and bring-up software. A compartment is conceptually similar to a tile in a ManyCore architecture, which are today widely used for compute acceleration and payload data processing [205]. However, their functionality is different, as a ManyCore compute-tile usually is constrained to run simple software, without supporting interrupts, inter-process communication, and I/O. A compartment instead runs a full copy of a general-purpose OS with rich software, has access to hardware timers, interrupts, may preform inter-process communication freely, and can handle I/O autonomously. Besides an on-chip memory holding the bootloader, it is also outfitted with a dedicated dual-port state-memory used to exchange lockstep information. The topology of a compartment is depicted in Figure 70. Each compartment is outfitted with a diagnostic access port, which enables low-level access to a compartment's internal logic through an SPI2AXI bridge. This facility is further described in Section 10.3.4.

In general, for the sake of reliability, the use of SPI or I2C based satellite bus architectures is in general discouraged. However, in Chapter 9, we showed how the interfaces of multiple compartments can be concentrated to emit only a correct result to the satellite bus. Indeally, a network-based satellite-bus should be implemented, which has been shown to be more robust to failures aboard CubeSats of all sizes. If an on-board network is available, no interface-concentration measures are needed, as the network can take care of data de-duplication and can assure that data from a faulty compartment is not propagated. See also [94], for an excellent example of how this can be done while providing real-time guarantees.

On-chip memory controllers used across our MPSoC are implemented in BRAM, which in turn consists of SRAM. Xilinx library IP offers ECC for caches and on-chip memories to detect and correct faults. We utilize Hsiao ECC to protect the data stored in these memories due to its lower logic footprint and otherwise comparable performance as compared to Hamming coding. Due to the brief lifetime of data in caches and buffers, no scrubbing is necessary and the overhead induced through ECC would be detrimental to the overall robustness of the system. Instead, faults in these components are mitigated in software, as described in Chapter 4. To avoid accumulating errors in a compartment's bootloader, we can attach an error scrubber to each

compartment's local interconnect, which is managed by each compartment.

To protect the running configuration of our SRAM-based FPGA, we implement CRAM-frame ECC using the Xilinx Soft Error Mitigation IP (SEM [354]). However, configuration-level erasure coding and scrubbing can still only detect faults in specific components of the FPGA fabric (e.g., not in BlockRAM). We address this limitation at the system level: Our coarse grain lockstep functionality enables us to detect faults in the fabric with compartment granularity within 1-3 lockstep cycles, which is further discussed in Chapters 4 and 5. In practice, this closes the fault-detection gap left by scrubbing and configuration erasure coding.

Each memory controller set consists of a DDR4 memory controller, a QSPI controller, a set of clock and reset generators, as well as an optional memory scrubber core and the top-level AXI crossbar. The optional memory scrubber cores can be controlled by the supervisor to avoid potential interference by malfunctioning compartments.

Each compartment has full write access to a segment DDR memory, while it can access the DDR memory in its entirety read-only. We construct the interconnect used by compartments to access a controller set from an AXI crossbar and four AXI switches, one for each compartment. The top-level crossbar is connected to the area-optimized AXI interconnect attached to each compartment, which makes up the second level of the MPSoC's interconnect. In each interconnect, we realize memory protection for the address space of the relevant compartment to avoid a single point of failure causing misdirected write access. Thereby, we create a topology that strongly isolates compartments from each other, and assures non-interference between compartments.

The address space of all compartments is uniform, enabling memory structures to be migrated between compartments and re-used. Through the MMU component indicated in Figures 70 and 69, we perform the necessary address translation operations.

In case one memory controller set fails, MPSoC compartments that were using this set will switch to failover through a reboot. Compartments that are already utilizing the secondary set can continue executing correctly and provide non-stop operation. Hence, it is desirable to run two of the MPSoC's compartments off the A-controller set, and the rest off the B-set. This allows the software-implemented fault tolerance functionality to guarantee non-stop operation even if an entire memory set would fail. In our proof-of-concept, we realize this functionality by outfitting compartments to be able to use two kernel variants, of which one booting into with main memory in the A set, and the second one into the B set. However, there are more elegant ways to accomplish this, e.g., using position-independent firmware images [358].

To efficiently perform lockstep state comparison and synchronization between compartments, an MPSoC has to provide adequate means of exchanging state-data, as discussed also in Chapters 4 and 9. For small MPSoCs with less than 6 cores, this is realized in DDR/SDRAM memory. For larger designs, a dedicated state-exchange network improves performance and offers stronger isolation. These components are depicted in green in the figures. Access to state memory then takes place entirely on-chip without passing through caches, and the global interconnect.

10.3.4 The Supervisor-FPGA Interface

The supervisor can access the FPGA through the FPGA's JTAG interface. JTAG in principle is powerful which can be used as a universal tool to interact with the FPGA and its MPSoC, and manipulate it in a variety of ways. However, JTAG TAPs can be

very complex, and the protocol does not assure the integrity of transferred data, while binary data transfer via JTAG can be very slow. Hence, we only use it to reconfigure the FPGA in case the on-chip configuration controller fails.

The supervisor can trigger an interrupt or permanently disable a compartment, and can induce a reset in compartments, memory controller sets, for the configuration controller, and for the FPGA itself. This is realized through a set of GPIO pins attached to the supervisor. The supervisor can conduct low-level diagnostics and has access to each compartment's address space, without having to rely upon a compartment's processor core.

We realize high-speed interconnect access through SPI, as the CubeSat community is already familiar with this type of interface. As we just required a direct point-to-point between the FPGA and the supervisor without chip select, this interface setup on the PCB-side is very simple. We attach an SPI2AXI bridge to each compartment's local interconnect, and additionally to each memory controller set. This SPI-bridge can be assembled entirely from well tested, free, open-source IP available in the GPL version of GRLIB, using the SPI2AHB and AHB2AXI IP cores. Alternatively, a variety of open-source SPI2AXI cores are available, e.g., on gitlab, but the quality of these cores is uncertain. Xilinx and other vendors offer a selection of commercial IP cores.

The supervisor also communicates with the FPGA-internal configuration controller, which is outfitted with a conventional SPI-slave interface. In contrast to the SPI-diagnostics setup used for accessing the interconnect of compartments and memory controller sets, the configuration controller actively collaborates with the supervisor. The configuration controller communicates with SEM and can be deactivated by the supervisor in case of failure. During normal operation, it will notify the supervisor about faults in the FPGA fabric. It can then perform reconfiguration via ICAP. The satellite developer can therefore deposit multiple differently placed designs for each partition in configuration memory, which the configuration controller can attempt to use to resolve a fault. Finally, the configuration controller will report outcome of the repair attempt to the supervisor.

Architecturally, the configuration controller resembles a stripped-down compartment design, but is constrained to a minimal logic footprint in the following way:

- It can run only baremetal code or an RTOS, not a general-purpose OS, thereby reducing the controller's logic footprint.
- This software is stored directly in on-chip BRAM which is part of the reconfigurable fabric.
- It has no access to the memory controller sets, to prevent interdependence between static logic and partial-reconfiguration partitions.
- Besides its SPI master connected to configuration memory, the configuration controller has no other external interfaces.

In case of failure, the supervisor can substitute the full set of the configuration controller's functionality through JTAG, and can recover it through full-FPGA reconfiguration.

As depicted in Figure 71, the supervisor can utilize it's SPI interface to access the different components of the MPSoC in a controlled and performance-efficient manner.

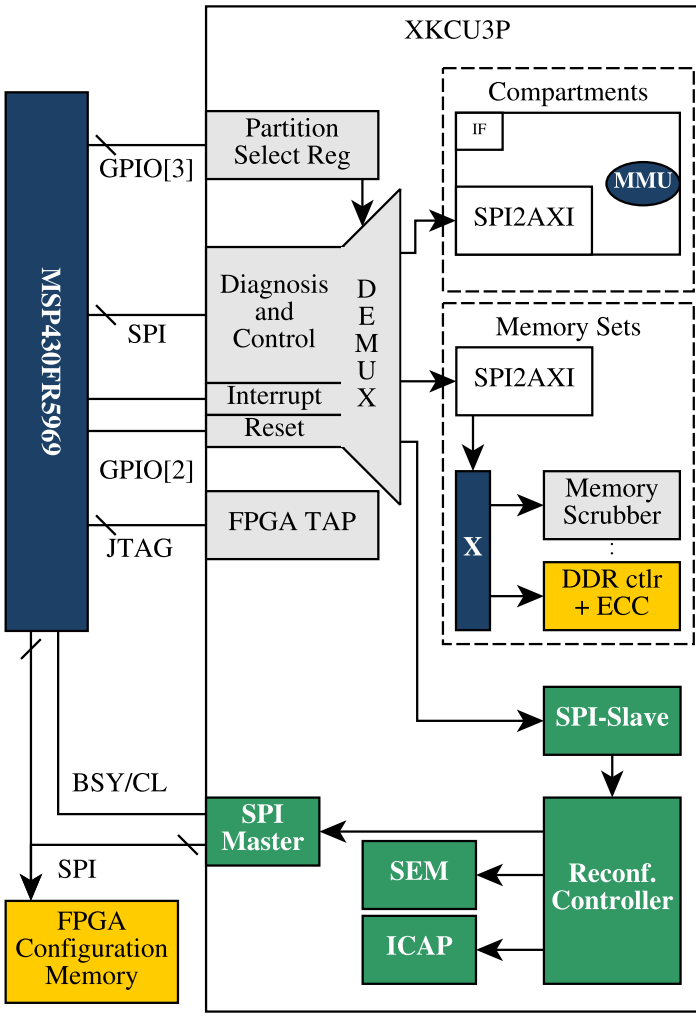


Figure 71: The design of our supervisor-FPGA control and diagnostic interface including the debug-facilities used by the supervisor to access different compartments of the MPSoC.

It can disable individual compartments in case of failure by using existing circuitry required for partial reconfiguration, as indicated in Figure 70. However, instantiating the combination of SPI, reset, and interrupt lines for each compartment, memory set, and the reconfiguration controller would require a large amount of IO-pins. In practice, the supervisor will only communicate one MPSoC component at any given time, and never with multiple concurrently. Hence, we de-multiplex (DEMUX) this interface, thereby reducing the need for I/O resources to just an SPI interface and 5 GPIO lines.

10.4 Handling Chip-Level SEFIs and Failure

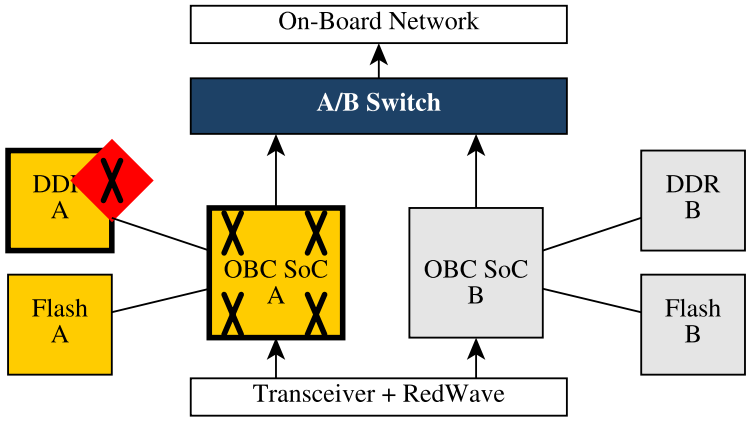
Our proof-of-concept MPSoC design spans only of a single FPGA and is not designed to withstand component-wide SEFIs affecting the entire FPGA. However, it can be implemented to tolerate such faults and even full component failure.

Figure 72a depicts an idealized traditional A/B-failover system with I/O switching. Such a system can tolerate the failure of components in either the A or the B side, but fails if an additional fault occurs elsewhere in the system. The B-side of the system remains inactive until a fault has been detected and isolated, and can be used productively without further design measures in hardware. Due to failover being implemented at the component level in hardware, additional glue logic required for switching between the A and B-system. It is usually not possible to test the failed side without further design measures, and tests can only be conducted if the system is taken offline. These limitations can be worked around with more glue logic and a more complex failover implementation, but even then the relevant logic can usually not just be be turned off and bypassed. Instead, it remains a potential failure source.

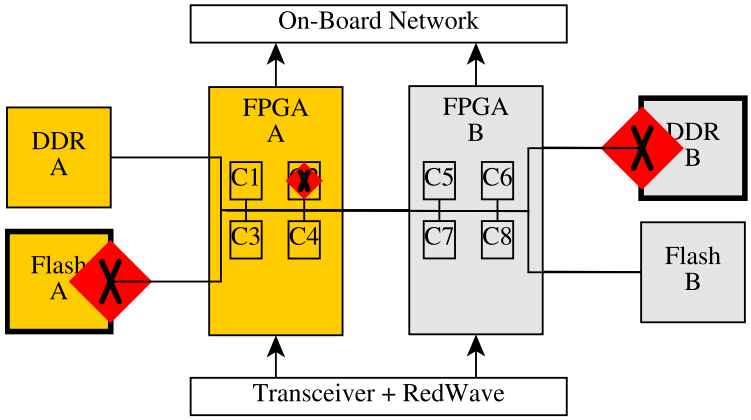
The system depicted in Figure 72b implements our architecture on two FPGAs and does not suffer these limitations: Instead of implementing all compartments and shared memory controller sets on a single FPGA, they can be distributed across multiple FPGAs. The chip-to-chip AXI IP used to connect two or more FPGAs is available in the Vivado IP library. The failure of, e.g., a memory component connected to one FPGA, does not cause the failure of an entire redundant system side. Compartments on one FPGA connected to a failed component can still access components on the B-side. The supervisor and platform controller on the faulty side can then reconfigure the relevant FPGA partitions, and conduct further analysis on the failed components. The system can thus continue thus support non-stop operation in case of severe component failure, if threads-replicas are distributed so that not all replicas of a thread are executed on the same FPGA. In a TMR setup, this enables non-stop operating, e.g., with the A-side running 2 replicas on one FPPGA and the B side running the third replica on the other. In NMR setups, two replicas can be assigned to each side, allowing fault-detection even if one of the FPGAs has failed during the same lockstep cycle. For diagnostic purposes, thread-replication and therefore fault tolerance can also be constrained temporarily or even fully disabled. Even a severely degraded system implementing our architecture that has suffered multiple component failures can thus still operate correctly and support non-stop operation. In contrast to a traditional OBC based on component-redundancy, our architecture thus can delivers stronger fault tolerance capabilities than traditional OBCs. As compartments on different FPGA can share resources, this allows for increased efficiency and performance as compared to traditional systems.

To support larger MPSoCs with more than 8 compartments efficiently, a more

scalable interface between compartments and memory controller sets should be used. This can be achieved with a Network-on-Chip (NoC). A NoC allows drastically larger MPSoC designs [329] due to improved scalability, but also enables fault-tolerant routing [349], backwards error correction (re-transmission), and quality-of-service support [359]. When implementing our architecture with a NoC, the shared memory controller sets would be implemented as one NoC layer, while the state-exchange network forms a second layer. In contrast to conventional interconnects typologies, a NoC can also utilize error correction for NoC routers [93].



(a) A traditional redundant system where there A-side failed due to malfunction in one memory components, which will fail once a fault occurs on the B side.



(b) Our architecture, which is still functional and not degraded, even though multiple components have failed on both sides.)

Figure 72: Fault tolerance examples of a traditional OBC and our architecture, which shows that our architecture can tolerate a much increased number of faults than a traditional system.

10.5 Utilization and Power Comparison

The quad-core MPSoC architecture described in this chapter was implemented on a set of Kintex Ultrascale and Ultrascale+ devices using Xilinx Microblaze soft-cores running at 300MHz, and DDR4 controllers. In our proof-of-concept, we utilize a FeRAM-based MSP430FR5969 controller for our proof-of-concept, for which a low-cost space-grade substitute is available. The MPSoC is reproducible in Xilinx Vivado 2017.1 and later. The necessary IP is included in the Vivado IP library, and can be obtained free of charge through Xilinx's university program by academics and non-commercial scientific users. This serves as proof-of-concept for our architecture, with resource utilization indicated in Table 9.

For this Microblaze-based MPSoC implementation, the added logic footprint for instantiating a compartment as compared to just an application-processor without any peripherals is low. For size comparison between an interface IP-core and a compartment, a QSPI controller core is highlighted in Figure 68 in teal. It makes up only 2.5% of a compartment's LUT and 6% BRAM utilization, with other commonly used cores aboard CubeSat such as I2C or UART showing a similar or even lower footprint. The larger size of ARM Cortex-A53 processor cores reduce this ratio even further.

Our initial proof-of-concept was implemented on the Xilinx Virtex Ultrascale+ VCU118 Evaluation Kit with DDR4 controllers running at 1600MHz. This FPGA family was ideal for design space exploration as the kit has two DDR4 memory channels and a large fabric. Within the Xilinx Radiation Test Consortium we are currently working on a Kintex Ultrascale KU60/XQRKU060 test board for radiation testing, to which we ported our design. Logic and partition placement are depicted in Figure 68. FPGA utilization and power consumption tables are indicated in Tables 9 and 10. On KU60, DDR4 memory controllers run at 1000MHz due to generational constraints.

We ported our MPSoC also to smaller Kintex Ultrascale+ devices, the KU60's closest equivalent part KU11P and the smallest FPGA in the family and generation,

	KCU3P		KCU11P		KCU60	**(XRTC)**
Resource	Used	**% Total**	Used	**% Total**	Used	**% Total**
LUT	85505	**52.55%**	87187	**29.20%**	132359	**39.91%**
LUTRAM	9319	**9.33%**	9632	**6.49%**	19536	**13.30%**
FF	93766	**28.81%**	96043	**16.08%**	158617	**23.91%**
BRAM	**303.5**	**84.31%**	303.5	**50.58%**	316	**29.26%**
DSP	30	**2.19%**	30	**1.02%**	30	**1.09%**
IO	224	**73.68%**	224	**43.75%**	378	**60.58%**
BUFG	21	**8.20%**	22	**3.20%**	26	**4.17%**
MMCM	**2**	**50.00%**	2	**25.00%**	2	**16.67%**
PLL	**7**	**87.50%**	**9**	**56.25%**	13	**54.17%**

Table 9: Resource utilization our MPSoC on different Xilinx Kintex FPGAs. The XRTC variant's DDR4 memory controllers has a larger data-width due to package constraints. Design constraining fabric-resources are marked in bold.

FPGA	XKCU3P	XKCU11P	XKCU60
FPGA Generation	Ultrascale+	Ultrascale+	Ultrascale
Technology Node	16nm FinFET	16nm FinFET	20nm Planar
Part Package	SFVB784-I	FFVE1517-I	FFVA1517-I
Clocks	0.23W	0.29W	0.71W
Signals	0.11W	0.15W	0.30W
Logic	0.11W	0.15W	0.42W
BRAM	0.19W	0.19W	0.41W
DSP	<0.01W	<0.01W	<0.01W
PLL	0.37W	0.46W	0.72W
MMCM	0.23W	0.23W	0.21W
I/O	0.27W	0.34W	1.50W
Dynamic Power	**1.51W**	**1.81W**	**4.26W**
Static Power	**0.44W**	**0.70W**	**0.67W**
Total Power	**1.94W**	**2.51W**	**4.93W**

Table 10: Power consumption of the 3 quad-core MPSoC implementations. Data generated by Xilinx Vivado 2018.3's Implementation Power Report.

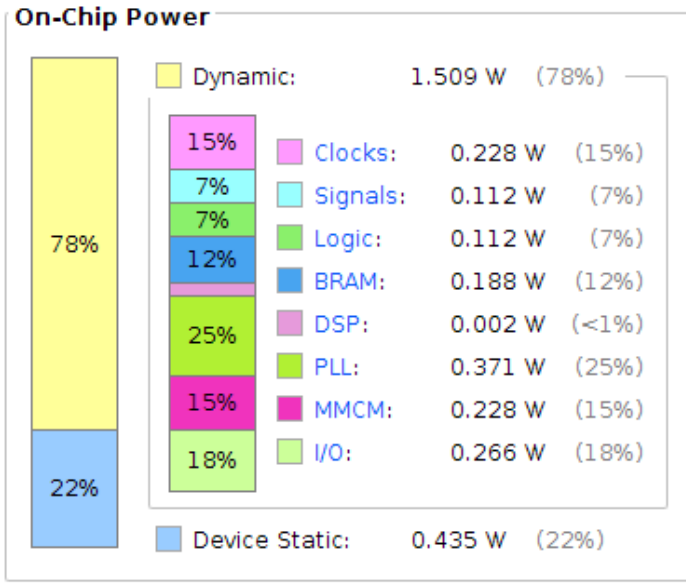


Figure 73: Power consumption of the 4-core MPSoC powering our MPSoC implemented on XCKU3P. Figure generated by Xilinx Vivado 2018.3.

the KU3P. The port required minor adjustments to the utilization of clocking resources, as both KU11P and KU3P have fewer clocking-resource (MMCM and PLL tiles) than the KU60. On KU11P, it was sufficient to switch several clock-generators used in the shared memory controller sets from PLL to MMCM tiles, without changing other parameters. The main constraint of the KU3P, however, required a reduction of clock generators in memory controller sets to 1 clock domain instead of 3 as described in Section 10.3.3. Due to the much smaller fabric of the KU3P, clock-domain sizes and routing distances decrease, resulting better timing of the design.

Despite much lower dynamic power consumption across the board in Ultrascale+, the KU11P variant shows slightly higher static power consumption than the KU60, which is counterintuitive. After discussion within the Xilinx Radiation Testing Consortium, the most plausible explanation for this anomaly is the different IO-bank placement within the fabric between these devices. On KU60, IO-banks are placed in more favorable locations considering MPSoC design than on KU11P. This increases

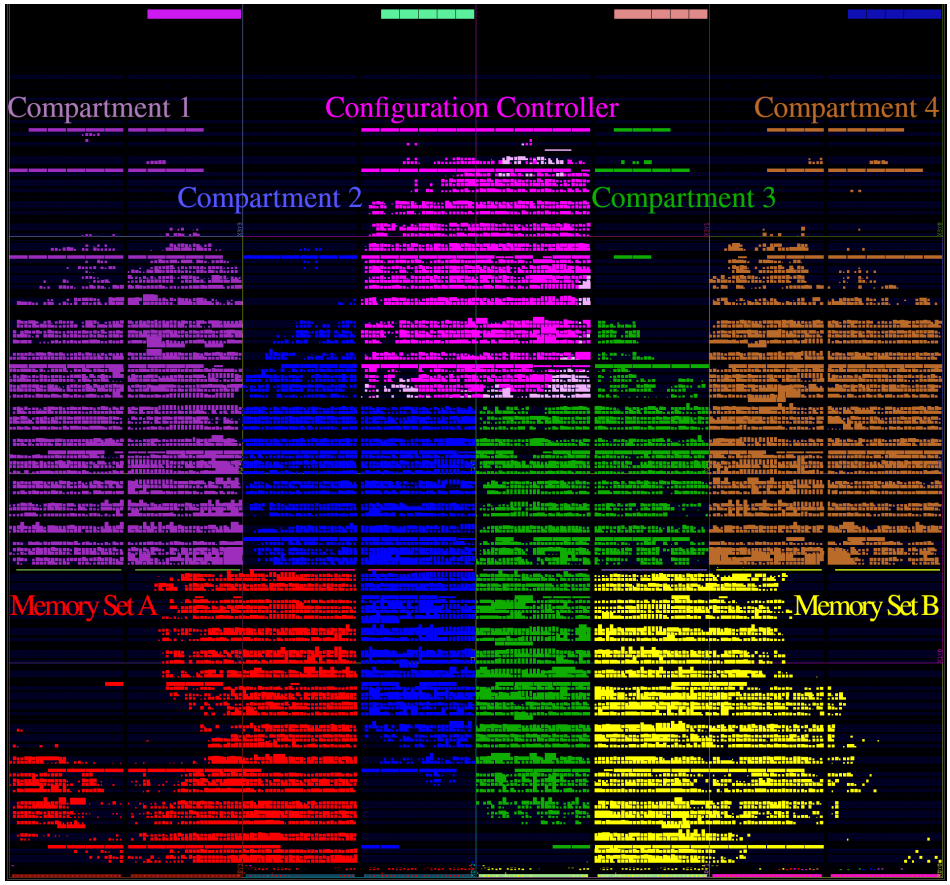


Figure 74: Logic placement of our proof-of-concept MPSoC on a Xilinx Kintex Ultrascale+ KU3P with 4 compartments (purple, blue, green, and brown), two shared memory controller sets (red & yellow) and static logic (pink). In contrast to the KU60 implementation, DDR controllers of this designs have reduced data width.

logic-spread, leaving less fully inactive fabric sections, which could explain an increase in static power consumption due to infrastructure on KU60.

The resulting Ultrascale+ MPSoC implementations, while functionally equivalent, show a 50% lower power consumption than the previous generation. This is due to manufacturing in a 16nm FinFET technology node instead of 20nm planar. Power savings mainly come from a reduced dynamic power consumption of this design, due to an increased degree of logic concentration in a smaller of FPGA-fabric area. For CubeSat-use, the Kintex Ultrascale+ family is therefore more attractive, despite the potential risk of IO-pin latch-up is acceptable [299] which today is mitigated in this field through the system-level measures [39]. On the the smallest Ultrascale+ part and most compact BGA package xcku3p-sfvb784 available at the time of writing, we achieved 1.94W total power consumption. This is well within the power budget range of 2U CubeSats. Vivado's power report for this design is depicted in Figure 73.

Synthesis was run in "Alternative Routability" mode, while implementation was with the "Performance-Explore" strategy with post-route placement & power optimization, as the resulting implementations showed consistently better timing and power utilization.

10.6 Experimental Results and Testing

We have tested our proof-of-concept OBC on Xilinx VCU118 (with 2 DDR memory channels) and KCU116 boards (with 1 channel due to board constraints), and constructed a breadboard setup in conjunction with an MSP430FR development board. Further information on this designs is available in Chapter 9, with an MPSoC implementation paper currently undergoing peer review. The actual platform for our research has been the ARM Cortex-A53 application processor, which is today widely used in a variety of mobile-market devices and certain COTS CubeSat OBCs. The architecture we presented in this chapter is processor and platform independent, with the MPSoC presented here implemented using Xilinx Microblaze processor cores.

To test our implementation, we have conducted fault injection through system emulation into an RTEMS implementation of Stage 1 running on a Cortex-A processor. In 2019, we also constructed a multi-core model of our MPSoC also in ArchC/SystemC on RISC-V to conduct further fault-injection close-to-hardware. The results of this fault-injection campaign are documented in Chapter 8. They show that with near statistical certainty, a fault affecting a compartment can be detected within 1–3 lockstep cycles, demonstrating that Stage 1 is effective and works efficiently.

10.7 Conclusions

In this chapter, we presented a CubeSat compatible on-board computer (OBC) architecture that offers strong fault tolerance to enable the use of such spacecraft in critical and long-term missions. It is the result of a hardware-software co-design process, and utilizes fault tolerance measures across the embedded stack. We described in detail the design of our OBC's breadboard layout, describing its composition from the component-level, to the MPSoC implementation used, all the way down to the software level. We implement fault tolerance not through radiation hardening of the hardware, but realize it in software and exploit partial FPGA-reconfiguration and

mixed criticality. To implement and reproduce this OBC architecture, no custom-written, proprietary, or protected IP is needed. All COTS components required to construct this architecture can be purchased on the open market, and are affordable even for academic and scientific CubeSat developers. The needed designs are available in standard FPGA-vendor library logic (IP), which in most cases is available to academic developers free of charge through university donation programs.

Overall, our OBC architecture is non-proprietary, easily extendable, and scales well to larger satellites where slightly more abundant power budget is available. We successfully implemented a proof-of-concept of our MPSoC for a variety of Xilinx Kintex and Virtex Ultrascale and Ultrascale+ FPGA. This MPSoC was implementable even for the smallest Kintex Ultrascale+ FPGA, KU3P, and we achieved 1.94W total power consumption. This puts it well within the power budget range available aboard current 2U CubeSats, which currently offer no strong fault tolerance.

A comparison to existing traditional space-grade solutions as well as those available to CubeSat developers seems unfair. Today, miniaturized satellite computing can use only low-performance microcontrollers and unreliable MPSoCs in ASIC or FPGA without proper fault tolerance capabilities. Using the same type of commercial technology, our OBC can assure long-term fault coverage through a multi-stage fault tolerance architecture, without requiring fragile and complex component-level replication. Considering the few more robust, low-performance CubeSat compatible microcontrollers, our implementation can offer beyond a factor-of-10 performance improvement even today. Considering traditional space-grade fault-tolerant OBC architectures for larger spacecraft, our current breadboard proof-of-concept implemented on FPGA exceeds the single-core performance of the latest generation of space-grade SoC-ASICS such as an GR740. However, it does so at a fraction of the cost of such components, and without the tight technological constraints of traditional or ITAR protected space-grade solutions.

Traditional fault-tolerant computer architectures intended for space applications struggle against technology, and are ineffective for embedded and mobile-market components. Instead, we designed a software-based fault tolerance architecture and this MPSoC specifically to enable the use of commercial modern semiconductors in space applications. We do not require any space-grade components, fault-tolerant processor designs, other custom, or proprietary logic. It can be replicated with just standard design tools and library IP, which is available free of charge to many designers in academic and research organizations.

Our architecture scales with technology, instead of struggling against it. It benefits from performance and energy efficiency improvements that can be achieved with modern mobile-market hardware, and can be scaled up to include more, and more powerful processor cores. At the time of writing, Xilinx has begun to introduce a new generation of FPGA-equipped devices manufactured in a 7nm FinFET+ technology node, in which the design issue causing latch-up in Ultrascale+ could also have been mitigated [299]. Xilinx's foundry TSMC expects this manufacturing process to offer approximately 65% reduction power consumption as compared to the 16nm FinFET node used for Ultrascale+ FPGAs [360]. Even if only half of this expected power reduction would manifests, in combination with FPGA-fabric optimizations, we can expect to achieve approximately 1W power consumption with our MPSoC implemented on a next-gen Xilinx FPGA. While these expectations based on experiences with the current 20nm Planar and 16nm FinFET manufactured Xilinx FPGAs, future FPGA

generations released within the next decade will, with near certainty [361], allow our architecture to even become usable aboard 1U CubeSats.

At the time of writing, each component of our OBC architecture has been implemented and validated experimentally to TRL3 in a 1-person PhD student project. From each individual component, we have assembled a development-board based breadboard setup. As next step in validating this new OBC architecture, we will construct a prototype for radiation testing. Since 2018, we have therefore contributed to the Xilinx Radiation Testing Consortium to develop a suitable Kintex Ultrascale-equipped device-test board. This will bring our architecture to TRL4, and is an intermediate step before developing a custom-PCB based prototype for on-orbit demonstration. Once this has been achieved, we intend to perform the final step in validation of this technology aboard a CubeSat.