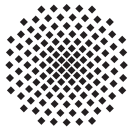

University of Stuttgart
Germany

Institute of
Robust Power
Semiconductor Systems

Student Research Project

Radiation Mitigation Techniques for EIVE Satellite Mission Payload Computer

Tobias Bischof

Supervisor: Laura Manoliu, M. Sc.
Prof. Dr.-Ing. Ingmar Kallfass

Period: 01.11.2021 – 23.03.2022

Stuttgart, 23.03.2022

Postanschrift: Institut für Robuste Leistungshalbleitersysteme
Pfaffenwaldring 47

D-70569 Stuttgart

Tel.: +49 (0) 711 685 68700
Fax.: +49 (0) 711 685 58700
E-Mail: sekretariat@ilh.uni-stuttgart.de
Web: www.ilh.uni-stuttgart.de

Declaration

I hereby declare that this thesis is my own work and effort and follows the regulations related to good scientific practice of the University of Stuttgart in its latest form. All sources cited or quoted are indicated and acknowledged by means of a comprehensive list of references.

Stuttgart, 23.03.2022

Tobias Bischof

Executive Abstract

The 'Exploratory In-orbit Verification of an E/W-band link' (EIVE) satellite mission demonstrates broadband data transmission from the low earth orbit to the earth with data rates of up to 15 Gbit s^{-1}. To ensure correct operation of the EIVE satellite and reduce the radiation impacts on EIVE's circuitry, radiation mitigation techniques are mandatory for the payload computer. Therefore, this thesis investigates the radiation mitigation techniques, mechanisms for the protection of the FPGA configuration memory and implements robust encoding mechanisms of the E/W-band validation files. The investigations and the implemented approaches are in line with the EIVE mission power budget limitations.

Zusammenfassung

Die Satellitenmission 'Exploratory In-orbit Verification of an E/W-band link' (EIVE) demonstriert die breitbandige Datenübertragung von der niedrigen Erdumlaufbahn zur Erde mit Datenraten von bis zu 15 $\mathrm{Gbit\,s^{-1}}$. Um den korrekten Betrieb des EIVE-Satelliten sicherzustellen und die Strahlungseinwirkungen auf die Schaltung von EIVE zu reduzieren, sind Strahlungsminderungstechniken für den Nutzlastcomputer erforderlich. Daher untersucht diese Arbeit die Strahlungsminderungstechniken, Mechanismen für den Schutz des FPGA-Konfigurationsspeichers und implementiert robuste Kodierungsmechanismen der E/W-Band-Validierungsdateien. Die Untersuchungen und die implementierten Ansätze stehen dabei im Einklang mit den Leistungsbeschränkungen der Mission.

Contents

List of Figures

List of Tables

1 Motivation

The project 'Exploratory In-orbit Verification of an E/W-band link' (EIVE) explores broadband satellite communication from the low earth orbit (LEO) to the earth by transmitting data in the frequency band between 71 GHz and 76 GHz with a data rate of up to 15 GBit s^{-1}. During transmission, atmospheric effects will alter the sent data. For this reason, a key task of the EIVE project is to send known pseudo-random bit sequences (PRBS) in order to analyze the transmission quality under different weather conditions and demonstrate the feasibility of the desired radio downlink throughput. Furthermore, it is possible to take and transmit full-HD pictures and 4K-livestreams with the on-board high resolution camera to showcase real-world scenarios with the need of high data rate transmissions [1].

The EIVE satellite represents the joint work of multiple project partners. Figure 1.1 shows a simplified overview of the subsystems, relevant for this work, as well as the responsible division. The Institut für Raumfahrtsysteme (IRS) is responsible for the on-board computer (OBC), which acts as a master circuitry for the multiMIND board and the companion boards. Therefore, the OBC controls the functionality of all subsystems in the satellite, depending on the satellites position and scheduled missions. The OBC receives commands and transmits execution log reports and current state information through a S-band link [2].

In order to execute commands, the on-board computer has to communicate with multiMIND, which is designed by Thales Alenia Space (TAS) and consists of two separate boards. The processing board consist of a Xilinx UltraScale+ MPSoC, where user-defined logic and software can be placed. To ensure the MPSoCs functionality, the processing board also contains a supervision circuitry. This processing board is designed in a way, that users of the multiMIND solution are able to do computational tasks, such as processing data and interacting with different parts on one or more companion boards, for instance an E/W-band link or a 4K-camera in the case of EIVE. Also, the processing board has components which are not accessible by the user. These components are dealing with supervisory tasks like storing and updating boot images, monitoring power lines, booting and shutting down the MPSoC, protecting it from latch-up events and observing the MPSoC with watchdogs. As an interface between processing core, OBC, companion boards and other electronic components, the mission interface board is an essential part of multiMIND. Compared to the processing core, it is a mission specific board, because the companion boards and other used electronic components strongly vary [3], [4].

As a last part of the electronic systems in the EIVE project, there are the already mentioned companion boards. For EIVE, the companion board is an evaluation board for a digital to analogue converter. It receives digital data from the FPGA and puts analogue data directly to the E/W-band

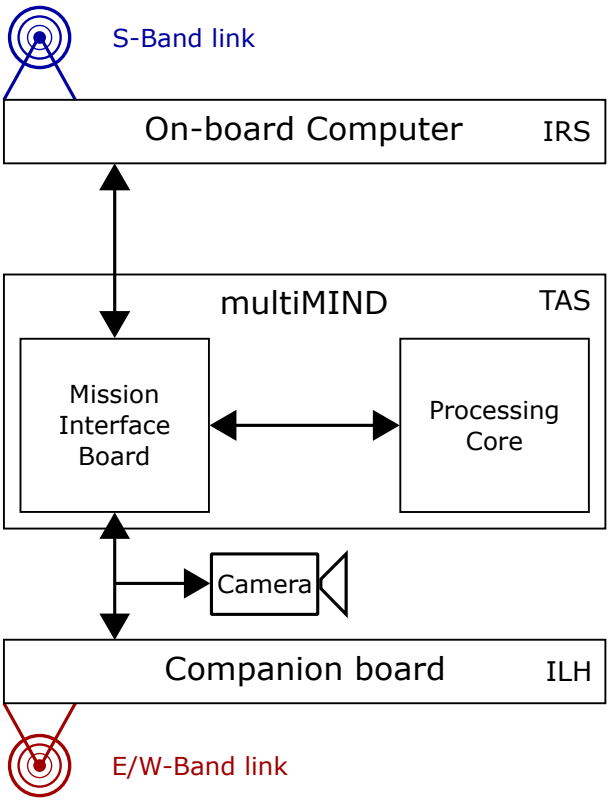

Figure 1.1: Electronic subsystems and their contributors in the EIVE project.

link and is part of the research of the Institute of Robust Power Semiconductor Systems (ILH) [2].

It is well established, that radiation present in space may have destructive effects on electronic components. The effects can differ in severity, since flipped bits may have no effects or can result in data corruption. There is also a chance to result in more severe injuries, for example degradation of microelectronics, biasing of instrument reads and damage of the physical circuits. The consequences of these effects can be wrong calculation results, altered program flows, data loss, unreliable measurements and total, non-recoverable device failures, which implies the loss of the mission. Even if some errors are negligible and have no effect at all, they should be found and fixed because they might have some impact at a later point in time [5]. Since all mentioned incidents can occur in the EIVE project, a stable radiation mitigation concept is needed. Therefore, this thesis investigates radiation impacts and basic mitigation techniques, before a solid radiation mitigation concept is proposed and implemented in order to increase the lifetime and robustness of the EIVE CubeSat.

2 Radiation Environment and Mitigation Techniques

This chapter starts with a short, but general overview about radiation environments, followed by EIVE-specific radiation challenges. Afterwards, radiation impacts on electronic devices, especially CMOS-based digital circuits and memories, are discussed. Based on this knowledge, techniques for radiation effects mitigation are investigated.

2.1 Radiation Environment and Effects

Before proceeding to radiation mitigation techniques, the radiation environment must be explained in a general way, starting from the definition of radiation over an outline of origins until characterization of different radiation types.

2.1.1 Environment

Before discussing the radiation environment, it is worth to point out that radiation is defined as any form of energy, emitted or transmitted by or between energetic sources [6]. For space radiation, which is considered here, the type of energy is limited to highly energetic particles, almost moving with light speed, and their corresponding wavelength. In general, this radiation can be found everywhere on earth. Apart from radiation sources in space, the earth itself is emitting radiation through natural radiation sources like radioactive elements, and human made ones like mobile phone networks. In comparison, radiation fluxes from earth sources are magnitudes of order smaller than from space ones. Additionally, through the existence and shielding functionality of the each magnetic field, radiation with its origin on earth is negligible in space [6], [7].

Radiation is emitted by sources, which are either galactic cosmic rays (GCR), solar particle events (SPE) or trapped particles. For a better understanding, these three categories along with their sphere of influence are schematically depicted in figure 2.1.

Galactic cosmic rays originate from outside the solar system and are typically found in free space [5]. GCRs consist of nucleus, travelling with nearly the speed of light. These nucleus are namely 85% protons, 14% helium nucleons and 1% highly charged ions (HZE particles), for instance highly charged carbon or iron ions [6]. The particle energies are below $LET_{th} < 15\,\mathrm{MeV\,cm^2\,mg^{-1}}$ for protons, where LET_{th} is the minimum linear transferred energy to cause a particle flux of

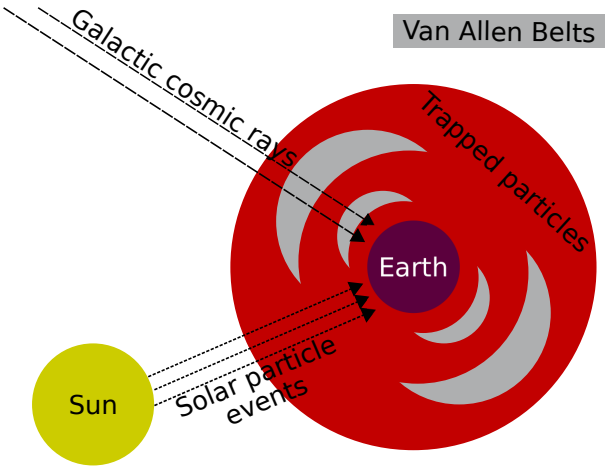

Figure 2.1: Space radiation sources and their locations.

$1 \cdot 10^7$ ions cm^{-2}. For larger particles, the linear energy transfer threshold is within the range of 15 MeV cm^2 mg^{-1} < LET_{th} < 100 MeV cm^2 mg^{-1}, which makes it difficult to shield against such events [5]. Nevertheless, especially HZE particles are able to produce huge damages which can yield to problems in electronic components and diseases for humans [8]. Furthermore, the quantity of galactic cosmic rays varies with solar cycles in an inverted manner: During solar maximum, the period when more solar flares occur, less GCRs can be observed and vice versa [5], [6].

Another radiation source are **solar particle events**, which can be coronal mass ejections, where a billion tons of solar particles are blasted into space, or solar flares. The latter ones eject some protons and lots of heavy ions which have a comparable amount of energy to corresponding particles, emitted by GCRs [5]. Therefore, their impact to humans and electronic devices is similar to those of GCRs. However, since solar flares are temporal events, which are emitting particles, their particles can just be measured during short periods of time, ranging from a few hours to some days [8]. Apart from these short-term fluxes, the overall frequentness of SPEs depends strongly on the current solar cycle. Instead of occurring more frequently during solar minimum like GCRs, SPEs are more frequent during solar maximum and less frequent during solar minimum [5], [6]. Furthermore, their abundance is correlated with the distance to the sun and decreasing with higher removal from the sun [5].

As third member of space radiation sources, **trapped particles** are the least dangerous ones for missions in the lower earth orbit. They consist of protons and electrons at energy levels below LET_{th} < 15 MeV cm^2 mg^{-1}, which are trapped by the earth magnetic field. These low-energy particles can mainly be observed in the inner Van Allen belts at altitudes between several hundred to approximately 6000 kilometres. At altitudes until 60000 kilometres, especially within the outer Van Allen belt, their energies can raise up to LET_{th} < 100 MeV cm^2 mg^{-1}. Apart from the measured height, there is one cluster with high amounts of trapped particles above the southern Atlantic region, which is called the south Atlantic anomaly. In there, even low energetic particles at small altitudes can have impacts to electronic devices due to their quantity [5], [9].

Radiation, emitted by the three sources can be split up into two different groups. Even though they all emit electrons, protons, neutrons and ions, the origin of these particles can be entirely different. On the one hand, they can be sent out directly by a radiation source, which is then referred to as primary radiation. This case applies for galactic cosmic rays and solar particle events. On the other hand, primary radiation can hit some other matter and ionize it. Then, electrons are emitted as secondary radiation in succession. This secondary radiation yields to large radiation doses in electrical components and can be found in LEO [7], [8].

2.1.2 Radiation Effects

Apart from their origins and particle energies, radiation can be characterized in terms of impacts, which can concern living organisms, human beings and electronic components. Effects on humans can be tissue damages and a higher risk to develop cancer [6]. But as the EIVE mission is not crewed, these effects are not further investigated. Nevertheless, impacts on electronic components are very important to consider and therefore specified in the remainder of this section. The investigation is mainly based on [5], [6], [7] and [10]. Wherever additional resources are referenced, they explicitly denoted.

In a first step, effects on electronic devices can be split up into two separate categories. The first one are cumulative effects, where long-term changes are considered. They can be observed as non-reversible degradation and long-term changes of device characteristics. Secondly, there are short-term effects caused by a single particle, called single-event effects (SEE). On the one hand, SEEs can be destructive an result in permanent damages of devices. On the other hand, SEEs can also be non-destructive, yielding from no effects at all, over bit errors in stored data until modification in operational procedures.

2.1.2.1 Cumulative Effects

Both, single-event effects and cumulative effects, can be split into various subcategories. Cumulative effects are subdivided according to the amount of energy per particle. On the one hand, if the energy amount is high enough, it can take out electrons from matters' atoms and molecules. This process is called ionization, coming from ionizing radiation. The resulting effects are measured in terms of a total ionizing dose (TID) with the unit of radiation absorbed dose (1 rad). This is an amount of energy absorbed by a unit mass, defined as $1\,\mathrm{rad} = 0.01\,\mathrm{J\,kg^{-1}}$. Typically, electronic components for radiation environments are designed to operate properly until a specified total ionizing dose. If the actual TID exceeds this specified value, the device is not guaranteed to work properly. Typical observed effects with rising TID are leakage currents in transistors, timing skews and threshold voltage shifts.

On the other hand, there is also non-ionizing radiation. It is not able to remove electrons due to its insufficient amount of energy. Non-ionizing radiation can be emitted from natural and ordinary sources like radio frequencies, visible light, infrared and UV-light. It is measured in terms of

total non-ionizing dose (TNID), also named displacement damage (DD), with the unit 1 rad. Non-ionizing radiation can create electrical defects in semiconductors crystal lattice. This affects bipolar devices, but not CMOS devices like FPGAs or ASICs. Furthermore, it is easy to shield against.

2.1.2.2 Single-Event Effects

For single-event effects, there is a larger space for subdivision. First of all, SEEs can be grouped by their severity and secondly, they can be characterized by their impact. As shown in table 2.1, the severity can be either destructive for long-lasting or non-reversible effects, or non-destructive for errors, which can be repaired. Below, the different types of SEEs are described.

Table 2.1: Subdivision of single-event effects

Destructive	Non-destructive
Single-event latch-up (SEL)	Single-event transient (SET)
Single-event snap back (SESB)	Single-event upset (SEU)
Single-event hard error (SEHE)	Multiple-cell upset (MCU)
Single-event burnout (SEB)	Multiple-bit upset (MBU)
Single-event gate rupture (SEGR)	Single-event functional interrupt (SEFI)
Single-event dielectric rupture (SEDR)	

Destructive Single-Error Effects

Single-event latch-ups are the most likely destructive single event effects for modern CMOS-based electronics. They occur when a single particle triggers a parasitic thyristor, which results in a high current flow and therefore, an increase in temperature. After some time, the device or some of its parts can be destroyed by the thermal effects. The only way to protect from SEL events is a power-reset of the circuit, before thermal damages can occur.

Very similar to SELs, **single-event snap backs** also yield to high current flows. They just differ by the underlying semiconductor structure: While SELs need parasitic thyristors to occur, a parasitic bipolar structure like PNP or NPN is mandatory for SESBs.

Apart from those parasitic structures in devices, radiation is also able to corrupt memories in non-reversible ways. In this case, charged particles affect the memory structure and damage one or more cells. Observed effects are stuck bits, whose values cannot be changed any more. Therefore, this type of radiation impact is called **single-event hard error**.

Another type of high current flow is called **single-event burnout**. It occurs when a power transistor is triggered, which yields to a high current flow and thus a thermal damage of components and devices. Fortunately, this effect is not common to FPGAs and ASICs.

Finally, there are **single-event gate ruptures** and **single-event dielectric ruptures**. For these events, a single particle strike ruptures a gate oxide or a dielectric layer, respectively. Thus, biases and leakage currents can be measured. For digital circuits, possible results are stuck bits. Up to now, there is no possibility to protect against such events, since they occur in a picosecond timescale.

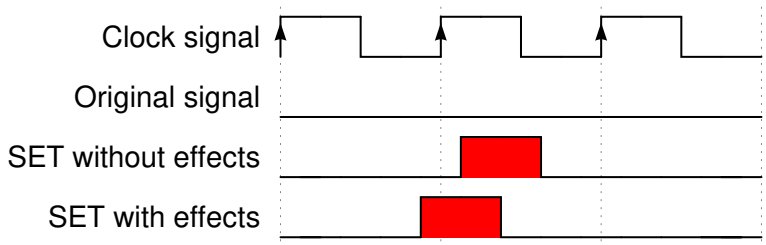

Figure 2.2: Single-event transient and its effects

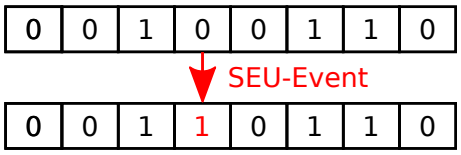

Figure 2.3: Single-event upset

Non-Destructive Single-Error Effects

Unlike destructive SEEs, non-destructive SEEs, also referred to as soft errors, don't have potential to destruct electronic devices, since non-destructive SEEs can just lead to flipped bits. Nevertheless, they are a concern in terms of device functionality and operational safety, depending on the kind of affected bits. Soft errors can have various impacts, which are briefly described below.

The first non-destructive SEEs are **single-event transients**, shown in figure 2.2. They are the most frequent type of SEEs in FPGAs, inferred by particles, hitting a connection wire at a combinatorial path between two registers. As an effect, a voltage spike on the hit line can be observed. Resulting effects in the circuit can be wrong data if the SET is captured by a memory, loosely speaking, then the SET becomes a SEU [11]. Other effects can be wrong signals or SEFIs, if the particle hits the clock distribution tree or a reset line. But it is also possible, that no effects will occur, especially when the voltage spike is not captured by a memory element [10].

SETs are not the only way to store wrong data, because particles can affect memories directly. These events are called **single-event upsets**. As shown in figure 2.3, SEUs induce single bit errors. They can have lots of effects, from wrong results in calculations to operational failures, depending on which memory is affected and the design of the circuit. Fortunately, they can be detected and corrected by error correction codes. Also, it is important to note that SEUs are more likely for small technologies and circuits with lower supply voltage, since they have smaller critical charges [11].

Furthermore, SEUs can appear multiple times in parallel. For most devices, the pattern, how bit errors are aligned, is essential. One possibility is to have multiple SEUs, but at maximum one per physical memory word. This type is called **multiple-cell upset**. As most error correction codes are able to correct one bit error per word, they could be completely recovered.

For multiple upsets in one word, namely **multiple-bit upsets**, it may not be possible to restore a fault-free state and the error may distribute across the device. Both of multiple upset effects are visualized in figure 2.4. With use of smaller technology, multiple upsets are becoming more likely.

As a last type of soft errors, **single-event functional interrupts** are caused by the same effects than

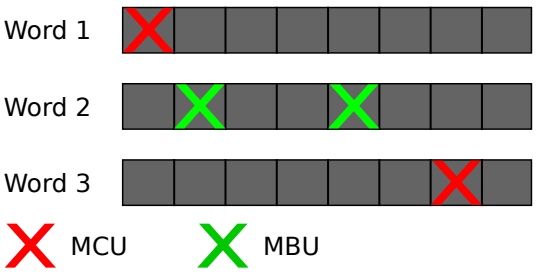

Figure 2.4: Multiple-cell upsets and multiple-bit upsets

SETs. The key difference between them are the hit wires. While SETs allow any wires to be affected, SEFIs require particles to hit either the clock distribution tree or reset lines. Therefore, they can result in complete or partially resets of the circuit or violation of setup and hold times. SEFIs can be corrected by a reset of the whole device, either by software or by power cycling.

Since this thesis is only about mitigation of radiation effects in digital devices, table 2.2 briefly reviles which single-event effects can have influences to different devices. As there is clearly visible, just SELs, SESBs and SEHEs are possible destructive events for digital devices. SEBs, SEGRs and SEDRs can be completely ignored in the radiation mitigation process, as they will not occur. For non-destructive events, any devices with some kind of internal memory can be affected by upsets and most of them also by multiple upsets and functional interrupts. Furthermore, both computational parts, FPGAs and microprocessors, are sensitive to transient events as they have more internal connection lines, where particles may strike.

Table 2.2: Summary of single-event effects and their presence on digital electronics.

Device category	Destructive				Non-destructive			
	SEL, SESB	SEHE	SEB	SEGR, SEDR	SET	SEU	MCU/MBU	SEFI
SDRAM	X	X				X	X	
DRAM	X	X				X	X	X
FPGA	X	X			X	X	X	X
Flash EEPROM	X	X				X		X
Microprocessor	X				X	X	X	X

Finally, some general effects of soft errors in FPGA devices are discussed, as they are the most important for EIVE. As stated above, all soft errors may yield to flipped bits in storage elements, but the effects of them are not only dependent on their count. It is relevant, in which type of storage a bit is flipped. In general, FPGAs consist of five different kinds of memories, shown in table 2.3. If an error exists inside block memories, distributed memory or flip-flops, wrong data can be read by following hardware, which then can produce other wrong calculation results. As a result, this error does not change the device behaviour or circuitry. On the other hand, when

device control registers are affected, there may be impacts on the whole device, like thus, expected by a SEFI. In the remaining case, when an error is in the CRAM, the consequences are completely different. Since all internal routing and combinatorial output tables are stored in there, a bit error can possibly result in rerouted wires, altered functionality or failures in operation [12].

Table 2.3: Memory types in a FPGA and their sizes for the Xilinx UltraScale+ MPSoC XCZU6EG, used in EIVE mission.

Name	Size (from [12], [13])
Configuration memory (CRAM)	17.0 Mb, 48054 frames
Block memory (BRAM)	25.1 Mb, 714 blocks
Distributed memory	6.9 Mb
Flip-Flops	0.429 Mb
Device control registers	Few kb

2.2 Analysis of the EIVE Radiation Environment

In this section, the radiation environment of EIVE is briefly summarized in order to provide facts about the planned mission. At the point this thesis was written, it was clear, that EIVE will be launched in the end of September 2022. Furthermore, the orbit was specified as LEO, with an altitude of 450 to 550 km [14]. For comparison purposes, it is worth mentioning that the International Space Station ISS circles around the earth at 400 km, which is about the same altitude [15]. In general, the low earth orbit is characterized by high-energy trapped protons, mainly created by effects of the South Atlantic anomaly. Because the earth magnetic field is able to shield from space radiation at these altitudes, high energetic particles from galactic cosmic rays or solar particle events play a minor role for EIVE [8].

For planning and implementing radiation mitigation strategies, it is mandatory to know, how often single events occur in the targeted orbit during the mission. This makes it possible to decide, which parts of a design must be protected in a certain way. Such statistics could be calculated with commercial tools like OMERE. In case of EIVE, these calculations were done by experts from TAS the IRS for single-event effects and for cumulative doses. Unfortunately, they are not quite up to date, because the launch date has been postponed.

In a first step, TAS' data for single-event effects, shown in table 2.4, is discussed. It is grouped into columns, depending on the type of events. Negligible soft-error types were removed by TAS. In summary, there will be 11.5 soft-error events per day, whereof 7 ones can be identified as latch-ups and 4.5 as upsets. However, EIVE's uptime is limited to ten minutes per day, due to power consumption limits. Since these soft-errors are able to damage devices, some kind of protection against them is mandatory. In terms of error effects, there is one event per day which may alter the MPSoCs functionality and the remaining 10.5 events can alter stored data. Therefore, protection of important, mission-specific data is essential [14], [16].

Apart from single-event effects, total radiation doses are very important in order to choose electric

Table 2.4: Number of single-event error expected for EIVE in events per day.

Device	Single-event upset	Single-event latch-up
MPSoC CRAM	0.04	N/A
MPSoC BRAM	0.02	N/A
MPSoC CLB	0.0003	N/A
MPSoC total	0.0603	1
NOR NVM	0.5	0.0003
NAND NVM	4	6

components in such a way, that they will outlive the missions duration. The results of this calculation is evinces in table 2.5. Briefly worded, for a satellite, launched in quarter two in 2022 into an orbit between 490 to 520 km altitude and a mission duration of one year, electric devices must be radiation tolerant up to a total radiation exposure of 12.6 krad or 5.2 krad, when shielded by 1 mm or 2 mm aluminium, respectively [14].

Table 2.5: Total ionizing doses expected for EIVE with start in Q2 2022 and sun-synchronous orbit at 490 to 520 km altitude, calculated with the OMERE tool.

Shielding	Mission duration	Radiation exposure
1 mm aluminium	1 year	12.6 krad
2 mm aluminium	1 year	5.2 krad

2.3 Mitigation Approaches

One more general example to underline the need for radiation mitigation is based on metrics, called failure in time (FIT) and mean time between failures (MTBF). FIT is defined by the counted number of failures occurring in a time interval of $1 \cdot 10^9$ hours and MTBF is the average time, in which one failure is expected. As an example, a part may have a FIT rate of 600 at sea level and 370000 at an altitude of 12 km. Thus, the MTBF can be calculated by $MTBF = \frac{1e9 years}{number of failures}$ as 190 years and 0.31 years, respectively [17]. Since EIVE is orbiting the earth at significant higher altitudes, the FIT can be expected even higher and therefore, the MTBF will decrease. In order to complete the EIVE mission successfully, failures during the mission must be reduced by implementing appropriate radiation mitigation techniques at a suitable level, described in this section.

2.3.1 Available Techniques

In order to ease the discussion about the integration of radiation mitigation techniques, the most popular radiation mitigation techniques are described in this section. Therefore, mitigation methods are summarized in table 2.6 in advance, before they are described in the remainder of this section. All investigated techniques are limited to the scope of digital electronics, which means that mitigation techniques for analogue or mixed-signal circuits are skipped.

Table 2.6: Summary of available radiation mitigation techniques.

Name	Protected elements	Radiation effects				
		SET	SEU	MBU	MCU	SEL
Information redundancy						
Error correction codes	Memory, SoC		X			
Error detection codes	SoC		X			
Spatial redundancy						
Duplex architectures	DCs, FPGA	X				
Triple modular redundancy (TMR)	DCs, SoC, FPGA	X	X			
Lockstep	System	X	X			
Reliability-oriented place and route	FPGA	X	X			
Temporal redundancy						
Triple temporal redundancy	Digital circuits	X	X			
Minimal level sensitive latch	Digital circuits	X	X			
Redundancy in software	Software	X	X	X	X	
No redundancy						
Use of radiation hardened parts	Memory		X			
Bit interleaving	Memory			X		
Memory Scrubbing	Memory, FPGA		X	X		
Hardened finite state machines	Digital circuits		X			
Selective use of resources	Digital circuits	X	X			
Watchdog timers	SoC	X	X			
SET filtering in data path	SoC	X				
Partial reconfiguration	FPGA		X			
Shielding	System	X	X			X
Power cycling	System		X			X
Current monitoring	System	X				X

2.3.1.1 Embedded Memories

As a basic element, almost any digital circuit contains embedded memories to store data. Hence, a protection mechanism for memories can improve the radiation tolerance for a wide range of electronic systems. In general, each memory device, as shown in figure 2.5, consists of memory blocks, which are built out of base units, named memory cells. Each cell is made up of basic electric components and has the ability to store one bit of information. When a cell is irradiated, the stored bit could flip and change its value from 0 into 1 or vice-versa. Therefore, the most effective way in terms of radiation resistance is to harden each single cell. This could be done by adding resistors or capacitances in a way to increase the critical charge. The drawbacks of this approach are penalties in used area and speed [7].

Another way to harden cells is to add extra transistors, such that each cell can restore itself if the data is corrupted. Due to the extra transistors, the drawbacks of this approach are an power- and area overhead, which varies with the used cell topology. However, in most cells, the overhead in power consumption is negligible compared to standard COTS-cells [7].

Beside protecting each cell, there are advantages of protecting memories at a block basis. In general,

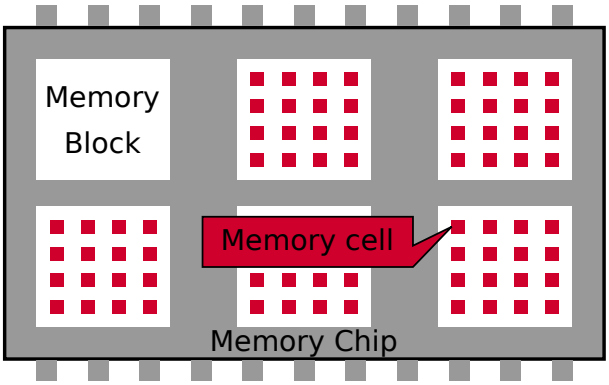

Figure 2.5: Schematic structure of a memory device.

Figure 2.6: SEEs in memory with and without bit interleaving.

block protection is cheaper in production and has the ability to operate on more than one bit of data. Thus, the concepts of error correction codes (ECC) and bit interleaving can be applied either in hardware or software. Error correction codes add redundant data to original information in a way, that correction and detection of a specified amount of bit errors is possible. ECC have the disadvantage of overhead in used memory and computation time, because the redundant bits must be calculated and checked at each time, when a write or read operation is performed. For bit interleaving, shown in figure 2.6, the property of locality of radiation strikes is used: Because cells are small enough, a single radiation strike will likely affect more than one of them. Since adjacent data mostly belongs to a single data word, the injected particles will have large impacts to small numbers of words. Especially when error correction codes are used, many bit errors in a few words are worse than single bit errors in many words, because they can yield to MBUs. Therefore, bit interleaving can be used to distribute originally adjacent data across the device to make neighbouring cells containing data of different words. The only downside of bit interleaving is an increase in access time, because different memory regions must be activated in order to read previously cohesive data [7], [18].

The last approach to protect memory cells is called data scrubbing, which periodically rewrites cor-

rupted memory bits with bits from a golden data location. Scrubbing prevents from accumulation of erroneous bits and ensures that error correction codes are working properly. As disadvantage, a secondary memory is needed to store the golden data. The pitfall is, that the golden data must be stored on a much more radiation hardened memory device. Otherwise, the golden data can also contain errors and wrong data is scrubbed into the used storage element [7], [18].

2.3.1.2 Digital circuits

In this section, radiation mitigation techniques, related to digital circuits, are investigated. Because the underlying semiconductor technology is already chosen, all mentioned techniques cannot mitigate permanent errors, caused by the total ionizing dose. However, almost all of the mitigation techniques are capable to weaken the effects, caused by transients and upsets. To keep it tidy, the mitigation methods are grouped in terms of redundancy. Firstly, duplex architectures and triple modular redundancy (TMR) are belonging to spatial redundancy, i. e. processing the same task multiple times in parallel and comparing the results by a voting circuitry. Secondly, the group of temporal redundancy contains triple temporal redundancy and minimal level sensitive latch approaches. Temporal redundancy is characterized by multiple processing operations, which are executed at different points in time, but not in parallel. As known from spatial approaches, temporal redundancy also requires voting logic in order to detect and correct wrong results. The third group is different, because no redundancy is necessary. Therefore, this last group contains general designing guidelines to design radiation hardened digital circuits.

Duplex architectures architectures consist of two identical logic blocks and one comparator. As depicted in figure 2.7, the input data is split up into two separate paths, such that the bottom and the top module will receive the same data. When both redundant modules are done with their processing tasks, both results are forwarded to a comparing logic. In the comparator, one of its input lines is directly connected to the data output signal and the error signal will be set to an error state, if a mismatch between both input ports is detected. When a mismatch is detected, no assertion about the correctness of the output signal can be made, since either input one or input two can be erroneous. Therefore, a reprocessing with the same input data is required to get a valid result. Apart from weak error correction capabilities, duplex architectures are characterized by doubling the used area and thus the power consumption. Furthermore, meeting timing requirements can be difficult, because the total number of recomputation is not determined.

Triple modular redundancy can be seen as an extension of duplex architectures. Instead of two identical logic blocks, TMR uses three copies of logic. Therefore, the comparator needs to change into a majority voter. This means, if one of the three voter input signals is changed, the other two ones will likely remain the same and are therefore valid. The only remaining context to perform a recomputation is denoted by three disjoint voter inputs. Nevertheless, the voter is a single point of failure and the output data will be wrong, if the voting logic is affected by radiation. With the

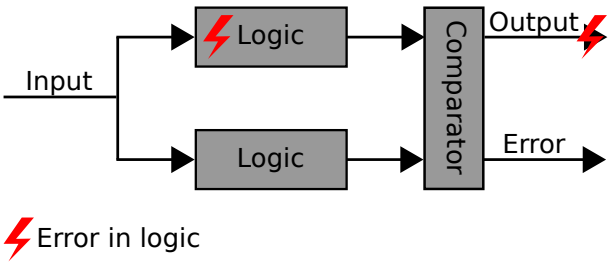

Figure 2.7: Schematic structure of a duplex architecture with comparator.

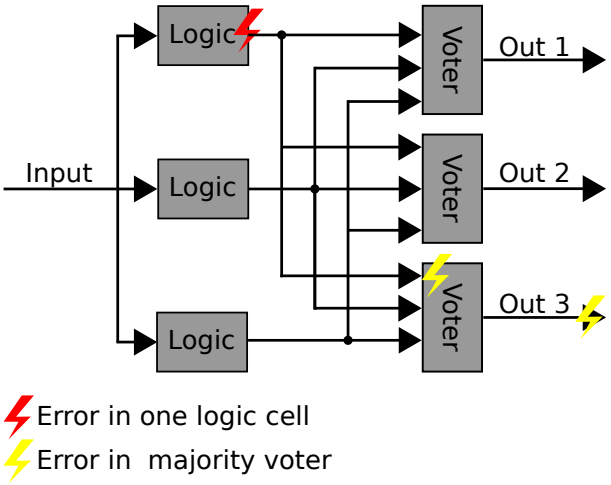

Figure 2.8: Schematic structure of a full TMR architecture with voter and different errors.

additional triplication of the voter, a basic TMR is transformed into a full TMR with three voters and the single point of failure is removed. When using full TMR, single event effects can now occur on computational logic, connection signals and voters, as depicted in figure 2.8. Apart from strong error detection and correction capabilities, both kinds of TMR come with an area overhead of factor three and a comparable overhead in power consumption.

Triple temporal redundancy (TTR) is efficient to filter out transient effects from signal lines. Since transients are short voltage spikes on connection lines, this is done by sampling the processing blocks output signals at different time instances and storing them into flip-flops. As a final step, the flip-flops' values are compared by a majority voter to determine the correct values. A schematic TTR is given in figure 2.9. On the one hand, TTR needs significantly less space and therefore, the power consumption is reduced, when compared to duplex architectures or TMR. On the other hand, flipped bits during data processing cannot be identified.

Minimal level sensitive latch (MLSL) is another technique of temporal redundancy. It consists of the same ports as TTR-blocks, but MLSL uses latches to store sampled signal data instead of flip-flops. Unlikely, an instantiation of latches is harder than an instantiation of flip-flops. Therefore, MLSL is less often used. In summary, MLSLs have the same space usage and power consumption

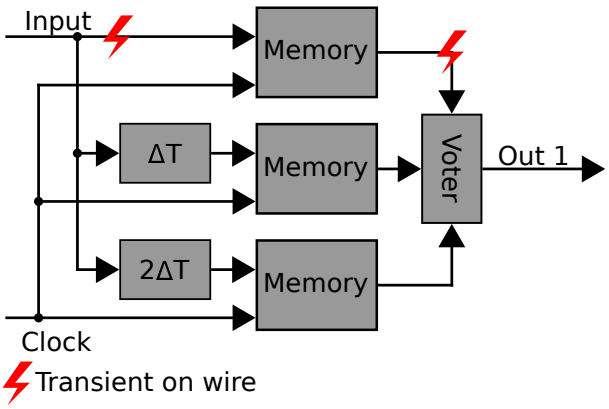

Figure 2.9: Schematic structure of a TTR architecture with delay elements, memories and voter.

than triple temporal redundancy.

Radiation hardened finite state machines are ordinary finite state machines (FSM) with more carefully designed properties. As a first property, state variables are designed in a way, which prevents accidentally state changes as result of a single flipped bit. The second property makes sure, that no illegal states can be reached from a valid state, or at least, illegal states will transition to a valid safe state immediately. Often, synthesis tools can fulfil both of this properties automatically. When lots of upset events are expected, for example when operating the circuit in interplanetary space, state coding with additional error correction codes can be considered to maintain the FSMs functionality. The downsides of radiation hardened FSMs are slightly more consumption of power and a small overhead in area.

Selective use of resources deals with different types of hardware, used for different tasks. For instance, some cells or wires in a digital circuit can be more hardened and should therefore be used for tasks with high reliabilities. More hardened resources could be cells with higher drive strength, radiation hardened flip-flops or memories, cells synchronous reset and set signals or cells with larger transistors. The drawbacks of this technique are dependent on the used cells, but it tends to have an overhead in area and power consumption.

2.3.1.3 System on a Chip

A System on a Chip (SoC) consists of many parts, placed on a single silicon chip. Thus, they can contain small elements like logic gates, medium sized elements like transceivers and large elements like processors. Since all parts together form a complex system, error mitigation techniques can also be applied in SoCs. Two possible techniques are already known from preceding sections. The first one is error correction coding, briefly described in section 2.3.1.1 and more precisely investigated in chapter 5. In contrast to error correction codes in memory, ECCs can now be applied to interconnection wires between parts. The second one is called triple modular redundancy, known

from section 2.3.1.2. From a SoCs' perspective, TMR is applied in larger scales, namely at part level. Because SoCs are at a higher abstraction levels than digital circuits, two new concepts are made possible. These are watchdog timers and filtering of single-event transients.

Watchdog timers are external or internal timers, which are used to indicate liveness of whole system or single parts. Once the timer is started, the task for the observed system is to reset the timer before it expires. As long as the timer is running, the system or part is assumed to be alive. On the other side, if the timer is expired, an error inside the system or part is assumed. In the latter case, the watchdog timer will either perform a hard reset on the affected circuitry, or it will raise error signals in a way, that other external or internal resets can be performed. The only disadvantage is a small area overhead for the watchdog circuitry.

SET filtering is a set of three techniques to break long transient propagation paths. While all three techniques can be found in any digital design, they are worth investigating further. The first technique is called logic masking. There, transient effects are filtered out by logical gates. For instance, an and-gate's output will always be low if one input is low. In this case, the second input can change between low and high arbitrarily or transients can occur there, but the output signal will not change. The other way around, for or-gates, the output will always stay high if one input is high. The second technique is called electrical masking. It is based on the transients attenuation, as it passes through electrical gates. If the electrical path is long enough, SETs will decrease in voltage amplitude until the voltage is too low to change the digital electric level. The third technique is temporal masking, which deals with sample intervals of flip-flops: Whenever a transient is at a flip-flops' input during a sensitive edge, it will be captured. The other way around, when a SET affects the input data wire of a flip-flop outside of the capturing interval, the inputs value will not be stored. Therefore, the transients has no effect.

2.3.1.4 Field Programmable Gate Arrays

Field programmable gate arrays (FPGAs) are chips, containing configurable logic. They are built from configurable logic blocks (CLBs), which consist of lookup tables and flip-flops, external pins, internal wires, routing blocks for internal connections and hard-wired components like processors or cells for digital signal processing (DSP). FPGAs are available in three different variants: Anti-fuse, flash-based and SRAM-based. In anti-fuse-FPGAs, the programmed logic and routing information is stored within anti-fuse cells. These are cells with initially high resistance, where electrically conductive paths are created during configuration. Whilst anti-fuse-FPGAs are highly radiation tolerant, since their configuration is not stored in a memory, they can only be configured once. On the other side, flash-based FPGAs store their configuration inside a non-volatile flash memory. Therefore, the FPGA starts its tasks whenever it is connected to a power supply. Although flash-based FPGAs are reprogrammable, they have less configuration cycles than SRAM-based FPGAs, which are the third variant. As the name suggests, SRAM-based FPGAs store

their configuration in a volatile SRAM-memory. While SRAM-FPGAs can be configured almost an infinite number of times, they need to be reconfigured anytime they receive power. Thus, a secondary memory is mandatory in order to provide the configuration bitstream. Because nowadays, almost all FPGAs are SRAM-based, the term FPGA will refer to those SRAM-based FPGAs in the remainder of this thesis [10].

The main concerns of radiation strikes hitting an FPGA are injuries of the configuration memory (CRAM). Because the CRAM contains all information about cell functionalities and routing resources, a flipped bit in there is potentially able to affect the functionality. On the one hand, malfunctions can occur when wires are rerouted or cut and thus, logic blocks are not connected any more or electric shorts are created. On the other hand, the functionality can be affected if bits of lookup tables are flipped. The latter case results in wrong calculation results. Therefore, reliable mitigation techniques are needed to protect FPGAs [10].

CRAM protection can be achieved in three different ways. Since CRAM is equivalent to any other memory devices in terms of radiation protection, memory scrubbing can be applied. This protects the configuration memory from accumulation of errors. A slightly different approach of CRAM scrubbing is called partial reconfiguration. The main purpose of partial reconfiguration is to exchange parts of the devices functionality during operation. Therefore, some tasks can be done faster or with less power consumption. However, partial configuration can also be seen as an approach of partial CRAM scrubbing. Compared to complete scrubbing approaches, where the whole memory is scrubbed, partial reconfiguration has the same effects within a defined CRAM region. The third approach to protect the configuration memory are ECC codes. Error correction codes can be used to correct single or multiple bit errors. In order to improve their efficiency, bit interleaving becomes reasonable.

Spatial redundancy, especially triple modular redundancy, is also a popular radiation mitigation technique when protecting FPGAs. TMR can be applied at different granularities. On small scales, flip-flops can be triplicated and their output signals can be voted locally. This prevents wrong data at the input of combinatorial paths and thus wrong calculation results. However, local TMR can just mitigate upset events in flip-flops, no transient errors in combinatorics. At a global scale, TMR can triplicate the functionality, including pins, reset lines and clock distribution trees. Since pins are also triplicated, an external voter is needed. The disadvantage of global TMR is, that replicas of internal logic are not placed into distinct areas of the FPGA.

The reliability oriented place and route algorithm (RoRA) can be used to fill the gap of global TMR. RoRA routes the replicas of configured logic into distinct areas of the chip with no interconnections, except for the input signals. Thus, radiation strikes are only able to impact one out of three logic structures. As a result, the external voter is always able to find a majority. Another advantage of RoRA is, that the algorithm prefers short connection wires, because they are less likely to be affected by transient effects. As the only disadvantage, the constraint of distinct area

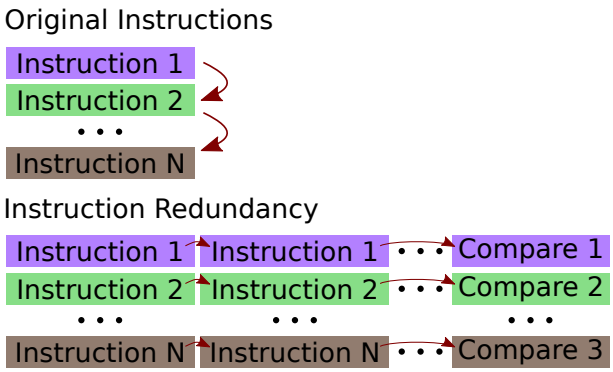

Figure 2.10: Schematic structure of instruction-level redundancy.

prevents the routing algorithm from optimum routing. Thus, the overall performance is reduced [19].

2.3.1.5 Software

Apart from hardware-based radiation mitigation approaches, software can also be designed in a radiation tolerant way. In all cases, software-based techniques are using temporal redundancy as a basis. In general, three approaches are possible, based on the abstraction level.

Instruction redundancy is at the lowest abstraction level. As shown in figure 2.10, any instruction is executed multiple times in sequence, but each instruction execution has its own distinct set of variables. Thereafter, all results are compared by a majority voter. Apart from a massive overhead in computation time and RAM usage, instruction redundancy is not transparent for software developers. During software development, the programmer is in charge of caring about data copies and multiplication of instructions. Furthermore, this approach is not applicable when interrupts are used, because an interrupt service routine is just called once. On the other side, instruction redundancy has nearly 100 % coverage for data and execution flow faults.

Task-level redundancy works on a higher abstraction level. Instead of re-executing each instruction separately, tasks are split up into three execution stages and consistency checks, during software design. As seen in figure 2.11, each task starts with data acquisition, where input data is collected (red incoming arrows). Afterwards, computations are separated into a data processing stage. Finally, during data presentation, computation results are emitted to other tasks, depicted by red outgoing arrows. In order to check for errors, data consistency checks are applied after acquisition and processing of data. The error detection and correction rate of task-level redundancy is, as for instruction redundancy, nearly 100 %. However, drawbacks are impairments in system architecture design, compatibility issues with available libraries and loss of performance.

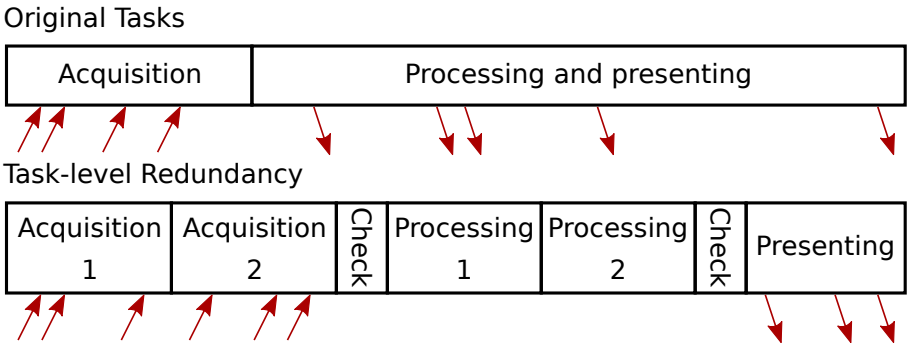

Figure 2.11: Task-level redundancy by separating tasks into three execution stages and performing consistency checks.

Redundancy at application level uses the concept of hypervisors, which are also used for virtual machines on personal computers. By using hypervisors, a whole application can be executed multiple times in parallel. The functionality of application is ensured by watchdogs and monitoring of memory interfaces. Application-level redundancy is therefore completely transparent for software engineers, but the error detection and correction capabilities are below 100 %. As for all software-based approaches, hypervisor-based techniques have high overheads in computation time.

2.3.1.6 System Protection

In contrast to software- or hardware-based radiation mitigation techniques, there are also strategies on system architectural level. System-based mitigation techniques can contain redundancy approaches, internal or external supervisions, or physical protection from radiation. For redundancy approaches, duplex architectures, TMRs, temporal- and software-bases redundancies and mixtures of them are possible. Supervisions can be realized as watchdogs at system or component level or by placing current monitoring circuits.

Shielding is the most simple technique. It can not only protect electronic components from latch-ups, transients and upsets, but also improve their lifetime by increasing the total ionizing dose. Suitable materials are aluminium or hydrogenous materials for light shielding and tungsten for heavy shielding. Their use and thickness are highly dependent on the expected radiation environment. If more radiation is expected, mainly materials for heavy shielding will be used with an increase in thickness. On the other hand, for less irradiated environments, a thin aluminium layer can be used. Apart from putting the whole system into a shielded box, more vulnerable components can have extra shields. In any case, an increase in weight can be denoted as the only disadvantage [14].

Watchdog timers, as known from 2.3.1.3, can also be used on system level. Additionally, the entire system or system parts can be power cycled, when a corresponding watchdog timer exceeds.

Power cycling has the ability to remove malfunctions, induced by single-event effects by withdrawing the power sources. At the moment, when power lines are cut, all volatile memories will be erased.When turning the power back on, the system is restarted in a defined initial state. In case of FPGAs, power cycling yields to erasure of the CRAM, and therefore, the logic must be reconfigured.

Current monitoring is used to protect from latch-up events. When single-event latch-ups are induced into a circuit, high current flows occur. High currents need to be mitigated, since they yield to thermal effects, which can destroy the device. As high current flows can be detected by current monitoring circuits, mitigation can be done by immediately removing all power supplies [3].

Lockstep architectures are a special kind of software redundancy, which is applied at system level. As advantages, all software-based faults are detected without constraints on software design. Thus, software needs no modification and usage of third-party libraries and interrupts is possible. Instead, the error detection is based on a second processor, called backup processor, and a consistency checker. Each processor has read access to the memory and runs the same software, but just the main CPU is allowed to write data. Thus, the consistency check will compare memory addresses and data at both memory buses. The disadvantages of lockstep architectures are area and power overhead of one CPU and one checker and a small overhead in processing time.

2.3.2 Possible Levels for Radiation Mitigation

A newly created system goes through several stages of development and production, before it can fulfil its desired purpose. During this development and production process, radiation mitigation techniques can be applied at almost all stages. In figure 2.12, these stages are denoted together with more specific resources for these stages and an overall radiation hardening manner. The latter one can either be radiation hardening by process (RHBP), where special techniques are applied at manufacturing process level, or radiation hardening by design (RHBD), where other techniques are used during the design process [7], [18].

During manufacturing process, radiation can be mitigated by selection of process parameters. This includes modifications of doping profiles, usage of specific materials and optimization of deposition processes. Changes of the manufacturing process refereed to the RHBP group.

At the lowest level of electronics design, the physical layout level, radiation hardening can be achieved in two ways. On the one hand, the physical layout of transistors can be changed in order to mitigate selected effects. On the other hand, the transistor's placement can be changed to reduce radiation sensitivity. For instance, less transient events will be observed, as the total wire length becomes smaller. Since physical layout is optimized during circuit designing, layout optimization belongs to the group of RHBD [7].

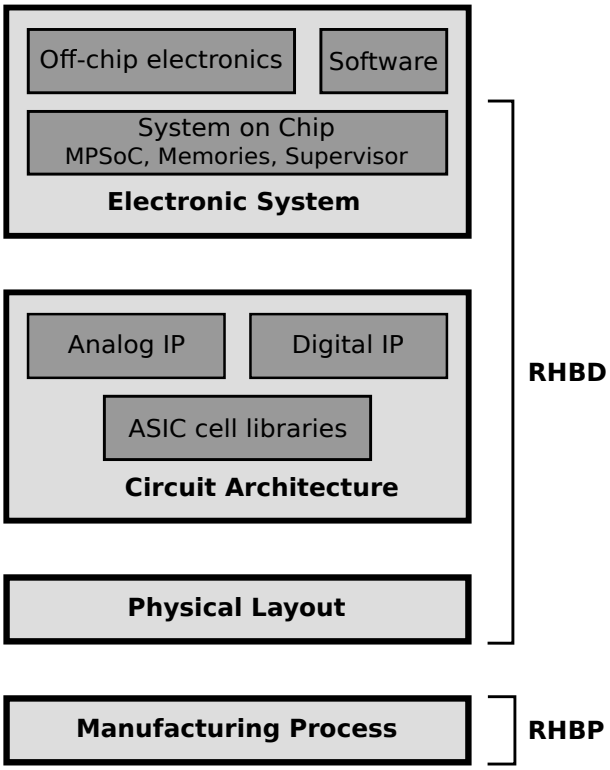

Figure 2.12: Overview of stages, where radiation mitigation is possible.

One level higher, at the circuit architecture level, radiation mitigation techniques are specific to the circuits nature or family. The nature can be either digital, analogue or mixed signal, while ASIC, FPGA and memory are values for the family property. At the architecture level, intellectual property (IP) is used to ease the design process. The majority of mitigation approaches for the architecture level make use of error detection and correction (EDAC) and redundancy [7].

The electronic system level is the highest level, where radiation mitigation can take place. Mitigation techniques for electronic systems are mostly applied at component level, unit level or software level. Even across multiple chips, radiation mitigation techniques can be applied. Most approaches at the electronic system level are based on spatial redundancy. It is also possible to apply current monitors, power cycling and watchdog timers.

3 Selected Radiation Mitigation Techniques for the EIVE Project

The EIVE mission has specific requirements towards radiation tolerance, leading to mission-specific constraints. These requirements and constraints are investigated in this chapter. Thereafter, known radiation mitigation techniques from section 2.3 are assessed due to those requirements and constraints.

3.1 Mission Requirements

The operational duration for satellites is mainly governed by battery recharging cycles and radiation effects. Especially the total ionizing dose limits the total lifetime, because unrecoverable effects are induced by the TID. Furthermore, since batteries will slowly use their capacity and in some cases their functionality, they also have an impact on EIVE's lifetime. By regarding both limitations, EIVE is designed to be operative for one year in LEO and will hopefully operate for a longer time. In any case, the EIVE PLOC shall remain operative at least during the planned satellite lifetime of one year.

Because received data will be compared to known sent data, the correctness of downlink sample data and accurate functionality of the payload computer must be guaranteed. For the case, that incorrect data is sent through E/W-band link, a comparison to original data on ground will not be meaningful. Therefore, error correction and indication of data corruption is required for EIVE. Due to this argument and additional possible destruction of electronic devices, the case of inaccurate functionality must be completely avoided. Therefore, an error-free operation during uptime is desired.

Another functional requirement deals with single-event error observation. All errors in the MPSoC, induced by SEEs, should be logged into the housekeeping report and written to a file in the eMMC-memory. This requirement is proposed by TAS, because the multiMIND-platform in the satellite was recently developed and therefore, no in-space radiation data is available. Thus, error logging will enable product improvements and can additionally be evaluated to regard transmission errors.

3.2 Mission Constraints

Beside mission requirements, constraints must be fulfilled, when contributing to the EIVE payload computer. These constraints are divided up into three groups. The first group contains global constraints, which are important for any contributor. Secondly, there are constraints for radiation mitigation techniques, which are important for developers of the radiation subsystem. The third group contains MPSoC-specific constraints.

3.2.1 Global Constraints

The first global constraint deals with lifetime. As EIVE is designed to operate for one year, no missions tasks at later points in time can be reliably scheduled. Loosely spoken, any component may have been damaged after this duration. Of course, if EIVE is still able to operate, it can perform tasks after the expected lifetime.

Secondly, mission and system planning need to take care of the overall power budget. Since one battery charge is needed to perform one E/W-band transmission, other tasks during the same time are very limited in terms of power consumption. Therefore, it must be carefully considered, which tasks are necessary during an E/W-band downlink. Power characteristics are briefly summarized in table 3.1.

Table 3.1: Available and consumed power in the EIVE project.

Name	Available power	Source
Power needed by MPSoC	6.4 W (0.8 W static, 5.6 W dynamic)	Xilinx power report
Power needed by DAC	approximately 5 W	Measurements
Power needed by downlink	approximately 10 W	Measurements
Total power needed	19.1 W at −20 °C, 22.5 W at 50 °C	Measurements
Total available power	25 W	Fixed by IRS

Finally, operational time slots for the payload computer are limited by two constraints. On the one side, transits over the E/W-band receiving station, located in Stuttgart, are limited by EIVE's trajectory. In numbers, EIVE will pass Stuttgart two times per day, at about 10 o'clock AM and PM. Since the transmission duration is limited by the elevation angle to a 10 minutes time interval [20] and other mission tasks are less likely, the per-day uptime could only be as high as 20 minutes. On the other side, one transmission interval will drain the whole battery and a following charging process will take one day. Thus, the per-day uptime is upper-bounded to 10 minutes in most cases.

3.2.2 Constraints for Radiation Mitigation Techniques

The necessity and selection of radiation mitigation strategy is highly dependent on the radiation environment and thus, the selected orbit. EIVE's orbit is constrained between 450 km and 550 km altitude. Therefore, EIVE needs to face radiation challenges as expected in the lower earth orbit. Typically, these radiation events are mainly caused by high-energy trapped protons, created by

effects of the South Atlantic anomaly. Because the orbit is already set, no radiation mitigation tasks for other radiation environments need to be considered.

Next, radiation impacts can be effectively reduced by shielding either the whole satellite completely or important components separately. Since EIVE is a six space-unit large nano-satellite, its space is already tightly packed by essential parts. Thus, room for shielding elements is limited and radiation effects will be more likely. As a consequence, radiation effects must be mitigated within the electronic system.

As known from section 3.2.1, power is a limited resource in EIVE. Therefore, radiation mitigation techniques must be selected and implemented in a way, that planned operations are not affected. Concretely, they should not be affected in terms of available power and operation duration.

3.2.3 Constraints for the MPSoC

FPGA-specific constraints are applying for the EIVE mission, because the payload computer is implemented and synthesized into a Xilinx UltraScale+ MPSoC.

The first such constraint applies to any FPGA-based project and has direct impacts on the functionality. Any flip-flop requires setup- and hold-times for its input signals. If those times are violated, wrong data will be captured at sensitive edges. In order to avoid storage of incorrect data, setup- and hold-times must not be violated.

Secondly, resources on the MPSoC are limited. These are lookup-tables, distributed RAM, block memories (BRAM), gigabit-transceivers, integrated processor cores and other hard-wired components. Therefore, a synthesized design cannot exceed the limit of available resources. Otherwise, the design will not fit into the FPGA and cannot be implemented. Table 3.2 is showing the available and the occupied resources of the MPSoC for the EIVE project. This logic constraint also applies to any FPGA-based project.

Table 3.2: Available and used resources in the Xilinx UltraScale+ MPSoC for the EIVE mission from Xilinx utilization report.

Resource	Used	Available	Used %
Lookup-tables (LUT)	86474	214604	40.29
LUTRAM	5179	144000	3.60
Flip-flops	89970	429208	20.96
BRAM	186	714	26.05
DSP	1	1973	0.05
Gigabit transceivers	9	16	56.25
Global clock buffers	28	404	6.93
Mixed-mode clock managers	2	4	50.00

The third constraint is EIVE-mission specific. The configuration of all hard-wired components, especially the processor system, is stored within a board support package (BSP). In case of EIVE, the BSP is provided by TAS. Hence, there are less degrees of freedom available to configure the MPSoC.

3.3 Consideration of Radiation Mitigation Techniques

In this section, radiation mitigation techniques form section 2.3 will be assessed in terms of constraints, known from section 3.2.

3.3.1 Techniques with Information Redundancy

Information redundancy is available in two different flavours. On the one hand, redundancy can be added at the granularity of files, done by saving the same file multiple times. For EIVE, duplicated or triplicated storage of files is possible, because 32 GB are available and the majority of files is not larger than approximately 99 kB. To improve the reliability, file copies can be distributed across two distinct memory chips. These redundant memory chips are also possible, but due to time constraints, both techniques will not be realized in this thesis. However, redundant memories will be implemented before launch.

On the other hand, redundancy can be added at bit- and byte-granularity. Therefore, a stored file will contain the original content and additional bits, needed by error correction and detection codes (EDAC). Due to the fine granularity, it is possible to use different EDAC codes for different groups of files. For EIVE, data files are split up into three groups. The most important data group contains E/W-band validation data. This group is important, because their content is known on ground and will be compared to received data. Therefore, all samples are protected by a heavy EDAC technique, as explained in chapter 5. The second-most important group contains log files. They contain performance and soft-error statistics, recorded during operation. If log files are corrupted, past metrics can be lost. Consequently, they should be protected by EDAC codes, but there is no need for ultimate protection. However, log-file protection is not implemented, because of its additional latency and increased power consumption during log-writing operations. The least protected category contains pictures, taken by the on-board camera. When pictures are lost, new ones can be taken with almost no effort. Also, as they have the largest size, EDAC codes will further strongly increase the picture size and power consumption for decoding. Therefore, taken images are not protected by any error codes. Another advantage of EDAC codes is the ability to correct a specified amount of bit errors. Thus, errors during reading of encoded files can be detected and corrected data can be written back to avoid accumulation of incorrect bits.

Apart from files and stored data, the concept of error correction and detection codes can also be applied to signal wires and data buses, either local on one chip or between chips. For EIVE, error detection codes without correction capabilities are used for communication between OBC and supervisor, and between OBC and MPSoC. These codes are implemented in terms of a one-byte-sized cyclic redundancy checks (CRC). Because CRC codes cannot correct errors, the retransmission of data is indicated by the receiver, if an error is detected. However, this usage of error detection code was determined by implementing the space-packet standard for communication between chips [21], which was implemented, before this thesis started.

To close the assessment of information redundancy techniques, the above investigated techniques

are summarized in table 3.3.

Table 3.3: Summary of information-redundancy based radiation mitigation techniques.

Granularity	Memory overhead	Time overhead	Expected mitigation capabilities
Memory chips	2×	almost none, only comparison logic	high, but error source remains unknown without additional concepts
File level	$n\times$ for n copies	n times if on one chip, $\frac{n}{2}$ times if on two chips for reading, $O(n^2)$ for comparing	high, if minority of copies is corrupted, the original can be reconstructed
Bit level	$\frac{codeword\_bits}{information\_bits}$	high, dependent on the used code	high, if code is designed for expected errors
Signals	None	Small if no error was detected, additional transmission latency on error	Low, but increased by retransmission in case of errors

3.3.2 Spatial Redundancy

Spatial redundancy is characterized by multiple instances of the same component. Thus, more chip area is needed and computations are done multiple times in parallel, before the results are compared. Hence, for spatial redundancy, the power consumption increases linearly with the number of redundant instances. Since the power budget for EIVE is limited and power consumption is close to the limit, spatial redundancy in only possible for small parts of the complete system. This constraint holds for duplex architectures, triple modular redundancy and lockstep architectures.

For this thesis, there are two reasonable places, where spacial redundancy can be implemented. On the one hand, duplex architectures, TMR or lockstep architectures can be implemented in the MPSoC. However, global approaches are not possible, as the utilization will be increased to more than 100 % for gigabit transceivers or clock managers. Local spatial redundancy approaches might be possible in some cases, for example in data filtering tasks. Nevertheless, local approaches come along with partial software redundancy, since the processor system (PS) controls the programmable logic (PL) and almost all PL parts are connected to the PS by an AXI-bus system. Because of the added complexity to already existing and tested software and hardware, redundancy approaches in the MPSoC are not used.

TMR can be extended by the reliability place and route algorithm. There, all logic is triplicated and routed into different, distinct parts of a FPGA. Additionally, the minimum wire distance is used as routing constraint, because transient events are less likely on short wires. Although the concept is reliable, RoRA is not used in the EIVE project, because of the massive overhead in area and power consumption. Furthermore, the pricing and complexity of the needed routing tool Precision Hi-Rel are to high.

On the other hand, a duplex architecture can be implemented around the two eMMC-memory chips,

which are accessible by the MPSoC. In this case, the voting logic can be implemented in software, since both memory controllers are pinned to the PS. As stated in 3.3.1, redundant memories will not be part of this thesis, due to time constraints. Anyhow, they will be implemented before launch.

At the end of this section, all three spatial redundancy approaches will be compared in table 3.4. In there, overhead values are compared to a system without spatial redundancy.

Table 3.4: Summary of spatial-redundancy-based radiation mitigation techniques.

Name	Space and power overhead	Used for
Duplex architecture	2× + comparison logic	eMMC-memory redundancy
Triple modular redundancy	3× + comparison logic	Not used
Lockstep architecture	1 processor + comparison logic	Not used
RoRA	3× + comparison logic	Not used

3.3.3 Temporal Redundancy

Temporal redundancy can be implemented either by hardware or by software. Since the electronic system is based on TAS' multiMIND, the only place for custom circuitry is the integrated MPSoC. Therefore, the minimal level sensitive latch (MLSL) approach is not applicable. This is because MLSL is based on a latch structure and latches have to be avoided in FPGA designs [22]. The other hardware-based technique, triple temporal redundancy (TTR), can be theoretically applied in the MPSoC. Practically, critical paths length will be increased and therefore, the desired clock frequency cannot be reached. Apart from these timing issues, TTR faces the same issues as TMR regarding power consumption. Therefore, TTR is not used in the EIVE project.

Beside MLSL and TTR, the concept of temporal redundancy can also be applied to software on different levels of abstraction. The approach at the highest abstraction layer, application-level redundancy, is based on the concept of virtual machines. Therefore, a hypervisor software is necessary. For EIVE, a hypervisor-based temporal redundancy approach is not possible, because hypervisors are scheduling applications according to their metrics. By implication, EIVEs software is interrupted in undefined places, which makes it impossible to perform tasks like filling the transmission buffer.

One abstraction level lower, task-level based temporal redundancy differentiates between data acquisition, data processing and data presenting. Hence, software must be designed in a way to make this differentiation possible throughout the whole planning and implementation process. Because task-level redundancy was never introduced or considered at the beginning of EIVE, it is hard to integrate task-level temporal redundancy afterwards, as it would change the whole implemented software. Therefore, task-level redundancy is not used by EIVE.

At the lowest abstraction level, instruction redundancy executes each instruction a defined number of times in sequence, before a consistency check is applied. Beside the added latency, power consumption and memory usage, instruction redundancy is not compatible with interrupt-based programming paradigms. As interrupts are used by the UART transceiver and the SEM-IP in the

EIVE-project, it is impossible to apply instruction-based redundancy.

To summarize this section, EIVE is not using any temporal-redundancy based radiation mitigation approaches. This is reasoned by the use of FPGAs, which disables the usage of MLSL and TTR techniques, and the software design, which makes implementations of software-based approaches impossible.

3.3.4 No redundancy

Apart from already investigated approaches, there are also techniques without need of redundancy. The most basic technique is physical shielding against radiation. EIVE is shielded by a aluminium shell. Because the available space inside is limited to six space units, i. e. six units of $10 \times 10 \times 10$ cm^3, there is no space left for additional shielding of individual components. The concept for physical shielding was done during physical design of the EIVE satellite and is thus not in the scope of this work.

During electrical design, TAS selected all components for multiMIND. Therefore, components were not only selected by their electrical characteristics and high efficiency, but also with respect to their reliability and radiation tolerance. The only digital hardware component, selected by the ILH is the DAC-board. Because all components were chosen before the start of this thesis, the part selection will not be investigated further. The same applies to selective use of resources.

The MPSoC in multiMIND is listed as non space-grade part. Therefore, extra protection is needed to guard the MPSoC from radiation effects, especially from SELs. Since latch-up events yield to current spikes, they could be detected by a current monitoring circuit. In the EIVE-project, a current monitoring circuit for the MPSoC is implemented in multiMIND by TAS. After current spikes are detected, power is withdrawn from the MPSoC within 100 μs [3]. The power cycling is completed by reconnecting the power lines to the MPSoC. Thereafter, the MPSoC will start booting and can operate in a safe and defined state.

When other single-event effects occur in the MPSoC, they might not be detected by multiMIND. This is the case, when upsets affect the MPSoCs expected internal behaviour, more precisely, if data in flip-flops or BRAMs is damaged. However, such errors can yield to altered functionality of software in the PS or hardware in the PL. In order to detect these SEE-induced errors, multiMIND observes the PS and PL by connecting them to distinct watchdogs. In case one watchdog is expired, the MPSoC will also be power-cycled and continue from the well-defined initial state [3]. As a slightly modified version of watchdogs, the used SEM-IP core in the MPSoC emits a heartbeat signal, which is observed by additional logic. Whenever the heartbeat goes out of specification, the MPSoC will request a power-cycle for itself. More information about the heartbeat observation can be found in section 4.2.2 [12].

As stated by the requirements in section 3.1, the MPSoC must operate highly reliable, especially during E/W-band transmission. Since malfunctions of FPGAs are mainly caused by single-event upsets, which induce flipped bits into the configuration memory, it is mandatory to find and

correct those errors. The applied concepts can be error correction codes, golden images and memory scrubbing. While error correction codes are already investigated in section 3.3.1, the remaining two concepts are not assessed up to now. Any memory scrubbing technique starts at the beginning of a memory and reads it word by word. If an error is detected in one word, either by EDAC codes or by comparison to a golden image, the damaged data is corrected. Afterwards, the corrected data will be written back into the memory. Once a read and correction cycle has been finished, the next cycle will start. For EIVE, memory scrubbing techniques are used for the FPGAs configuration memory are and implemented by the SEM-IP core, described in chapter 4. However, the board support package is delivered by TAS and the possibility to correct the CRAM from a golden image is not provided there. Nevertheless, since bitstreams for the MPSoC are stored in redundant NOR-memories, one of them can be considered as golden image, when the FPGA is completely reprogrammed.

Additionally, the whole configuration memory can be reloaded by power-cycling the FPGA. In EIVE, power-cycling is used when watchdog timers exceed, high currents are detected in the MPSoC or uncorrectable errors are detected in the configuration memory. Furthermore, only chunks of the configuration memory can be refreshed, by using the technique of partial reconfiguration. Because partial reconfiguration implies massive changes in the PL design and have impacts to PL performance, partial reconfiguration is not used for EIVE.

The concept of bit interleaving spreads multi-bit errors into distinct and distributed data words. Therefore, EDAC-codes are able to mitigate more errors without an increasing complexity. Because the concept of interleaving and EDAC-codes is very powerful, Xilinx uses bit interleaving and error correction codes in the CRAM of MPSoCs from the UltraScale+ series [12]. Furthermore, bit interleaving is used to protect E/W-band sample files, because they must not be corrupted during flight.

To ensure accurate internal state changes in finite state machines, FSM states can be coded in robust ways. In theory, this technique is easy to implement. On the other side, in practice, realization of safely coded states is not possible, because EIVE's FPGA design is IP-based. Since IP-based design mainly deals with closed-source IP blocks, finite state machines inside IPs are not visible to designers and therefore, coding of FSM states is impossible.

As described in section 2.3.1.3, SET filtering in data paths can be used to mitigate transient event effects. Beside consciously implemented elements for filtering SET pulses, the majority of them is filtered out, if the design contains and- and or-gates, long data paths and edge-triggered storage elements. Therefore, EIVE uses the SET filtering technique in an implicit way.

3.3.5 Implemented Radiation Mitigation Techniques in EIVE

To close this chapter, all radiation mitigation techniques, implemented in EIVE are summarized in table 3.5. There, the contributing institutions for each technique and references to other sections of this thesis are given as additional information. Table 3.5 reveals, that temporal and spatial redundancy approaches are not possible in EIVE, due to the power and timing constraints. The

only exception is a duplex architecture, where two eMMC memory chips are configured in a redundant way. Almost all other radiation mitigation techniques are possible and thus applied, in order to extend EIVE's lifetime.

Table 3.5: Radiation mitigation techniques for EIVE.

Technique	State	Contributors	References
Redundant files	Done in future	ILH	3.3.1
Redundant memory chips	Done in future	ILH	3.3.1
EDAC in files	Done	This thesis	2.3.1.1, 3.3.1, 5
EDAC on signals	Done	TAS, IRS	2.3.1.1, 3.3.1
Duplex architectures in MPSoC	Not possible		2.3.1.2, 3.3.2
TMR in MPSoC	Not possible		2.3.1.2, 3.3.2
Lockstep architectures in MPSoC	Not possible		2.3.1.6, 3.3.2
Duplex architecture for eMMC	Done in future	ILH	2.3.1.2, 3.3.2
RoRA	Not possible		2.3.1.4, 3.3.2
MLSL	Not possible		2.3.1.2, 3.3.3
TTR	Not possible		2.3.1.2, 3.3.3
Application redundancy	Not possible		2.3.1.5, 3.3.3
Task redundancy	Not possible		2.3.1.5, 3.3.3
Instruction redundancy	Not possible		2.3.1.5, 3.3.3
Shielding	Done	IRS	2.3.1.6, 3.3.4
Use of hardened parts	Done	IRS, TAS	2.3.1.2, 3.3.4
Current monitoring	Done	TAS	2.3.1.6, 3.3.4
Watchdogs	Done	TAS	2.3.1.3, 3.3.4
Heartbeat observation	Done	This thesis	3.3.4
Memory scrubbing in CRAM	Done	This thesis	2.3.1.1, 2.3.1.4, 3.3.4
Golden image	Done	TAS	3.3.4
Power cycling	Done	TAS	2.3.1.6, 3.3.4
Partial reconfiguration	Not possible		3.3.4
Bit interleaving in CRAM	Done	Xilinx	2.3.1.1, 3.3.4
Bit interleaving for samples	Done	This thesis	2.3.1.1, 3.3.4
Hardened FSMs	Not possible		2.3.1.2, 3.3.4
SET filtering	Done implicitly	This thesis	2.3.1.3, 3.3.4

4 Integration of the SEM-IP Core into EIVE

In chapter 4, the process of integrating the SEM-IP core is shown. Therefore, the need of the SEM-IP is motivated in section 4.1, before the the SEM-IP is described in section 4.2. Afterwards, the position in EIVE's programmable hardware system is outlined and two different approaches of integration are depicted. In addition, the selection of one integration approach is justified. The chapter is closed by a showing the processor system's side of SEM-IP integration and the testing results.

4.1 Necessity of the SEM-IP Core

Since EIVE operates in the lower earth orbit, effects caused by radiation, are more likely than on earth. For digital electronics, these effects can be high current flows, damage of devices or components, flipped bits in data and wrong voltages on signal wires. While high-currents and damages must be prevented by external devices, provided by TAS, and wrong voltages on signal wires will almost always have no effects on digital circuits, flipped data bits can have various effects. If user data is affected, wrong calculation results can be observed. When configuration bits of FPGA devices are flipped, either no effects, design malfunctions or changed pin behaviours can be noticed. In the worst case, the FPGA itself or external, connected components can be destroyed, because their device register contents are altered.

Indeed, the soft error mitigation (SEM) IP-core is not able to remove radiation from the FPGA, but it is able to mitigate, and in most cases also to eliminate radiation-caused effects in the configuration memory (CRAM). As result, changed circuit and pin behaviours, as well as corruption on external devices can be avoided in most cases. In order to increase the lifetime of EIVE and ensure 100 % uptime during E/W-band transmission, the use and integration of the SEM-IP core is indispensable.

4.2 The Soft-Error Mitigation IP Core

CRAM in all Xilinx FPGAs, and thus also in the used Xilinx UltraScale+ MPSoC, is organized as an array of frames, as shown in figure 4.1. In the used XCZU6EG MPSoC, the array of configuration frames has 48054 elements. Each configuration frame itself consists of 93 words with a size of 4 bytes or 32 bit. In sum, the configuration memory for EIVE has a size of 17.048 MB, uniformly distributed across the whole chips space [12].

The MPSoC uses three mechanisms to enable the SEM-IP core to detect and correct errors. For

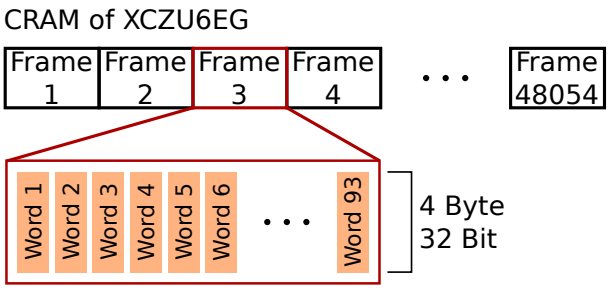

Figure 4.1: Configuration memory layout of the Xilinx UltraScale+ XCZU6EG.

detection, a CRC code is applied to the entire frame array. These CRC code enables robust error detection with low computational effort. To enable error correction, an error correction code (ECC) is applied to each configuration frame. Xilinx does not specify, which code is used, but the used ECC is able to correct four bit errors per frame, more precisely, four bit errors per 2976 bits. In order to effectively correct more errors, bits from one configuration frame are not stored next to each other. They are stored next to bits from other configuration frames. This technique is called bit interleaving. For the case, that a multi-bit error cannot be corrected, the SEM-IP can be configured to fetch whole configuration frames from external memory devices and write them to the corresponding corrupted frame [12].

Anytime, an error is detected by the SEM-IP, two different behaviours can be configured. On the one side, the error can be corrected immediately, regardless of the caused effects. On the other side, if the configuration frame is not used by the implemented design, or the flipped bit is known to have no effects, a correction can be omitted. This second behaviour is desired, when only small parts of the configuration memory are relevant for the design.

Like any VHDL component, the SEM-IP core must be tested as a single component during its development. More important for users, it must pass integration tests. Typically, both tests are done by using a simulation testbench, including a signal generator, device under test and an output signal analyzer. Since simulations are independent of the later used device, they are unaware of configuration memories. Thus, there is no chance to simulate the SEM-IP core during integration tests. Consequently, the only possibility to test the integration is to run the design on existing hardware. Therefore, the SEM-IP core provides the ability to inject errors, read the content of configuration frames during testing periods, read configuration registers and send ASCII-coded reports via an UART interface.

Compared to the whole EIVE design, the SEM-IP's overhead in used hardware is almost negligible. Only 425 LUTs, 490 flip-flops and 4 BRAMs are used.

4.2.1 Latency Times of the SEM-IP Core

Apart from the mentioned correction and detection capabilities, bit errors are existent until they are corrected. During this time, all negative effects, caused by flipped bits can potentially be observed. Therefore, quickly detected errors may have less impacts on the design than errors, which lasts in

the CRAM for longer time. To calculate the latencies, two global parameters are set by the used XCZU6EG device. The first parameter is the maximum possible frequency $ICAP\_F_{Max}$, to access the internal configuration via the internal configuration access port (ICAP), provided in [12], and the second one is the error detection time at this frequency.

$$ICAP\_F_{Max} = 200\,\text{MHz} \tag{4.1}$$

One more parameter is given by the actually used ICAP access frequency, which is set in the Vivado project. This is determined by the designed system and can differ between projects. For EIVE, the actual used frequency was read from the PL configuration in Vivado and is given below.

$$Frequency_{ACTUAL} = 100\,\text{MHz} \tag{4.2}$$

To simplify further calculations, a slowdown factor is introduced. It is defined as the ratio of maximum ICAP access frequency, defined in (4.1), to the actually used frequency from (4.2).

$$FrequencySlowdown = \frac{\text{ICAP}\_F_{Max}}{Frequency_{ACTUAL}} = \frac{200\,\text{MHz}}{100\,\text{MHz}} = 2 \tag{4.3}$$

To calculate the latencies for start-up, detection, correction and classification, Xilinx provides values for UltraScale+ devices at $ICAP\_F_{Max}$, as shown in (4.4), in [12].

$$BootTime\_F_{Max} = 127\,\text{ms} \tag{4.4a}$$
$$InitializationTime\_F_{Max} = 71\,\text{ms} \tag{4.4b}$$
$$DetectionTime\_F_{Max} = 28\,\text{ms} \tag{4.4c}$$
$$CorrectionLatency_{Correctable,F_{Max}} = 44\,\mu\text{s} \tag{4.4d}$$
$$CorrectionLatency_{Uncorrectable,F_{Max}} = 22\,\mu\text{s} \tag{4.4e}$$
$$Classification Latency_{Correctable,F_{Max}} = 154\,\mu\text{s} \tag{4.4f}$$
$$Classification Latency_{Uncorrectable,F_{Max}} = 5\,\mu\text{s} \tag{4.4g}$$

From these values, the startup latency at $ICAP\_F_{Max}$ is determined by the sum of boot- and initialization time from (4.4a) and (4.4b), respectively. However, the actual startup latency considered the desired ICAP frequency and the actual startup latency is determined by multiplying with *FrequencySlowdown*. As result, the SEM-IP core takes 396 ms to start. During this time, no errors

can be mitigated.

$$\begin{aligned} StartupLatency\_F_{Max} &= BootTime\_F_{Max} + InitializationTime\_F_{Max} \\ &= 127\,\text{ms} + 71\,\text{ms} \\ &= 198\,\text{ms} \end{aligned} \tag{4.5a}$$

$$\begin{aligned} StartupLatency_{ACTUAL} &= StartupLatency\_F_{Max} \cdot FrequencySlowdown \\ &= 198\,\text{ms} \cdot 2 \\ &= 396\,\text{ms} \end{aligned} \tag{4.5b}$$

To calculate latencies for detection, correction and classification, only a multiplication with *FrequencySlowdown* is necessary. For correction and classification, a differentiation between correctable and uncorrectable errors is necessary, since they are treated in different ways. The reason is explained in section 4.2.3. From the calculations, all results are given in (4.6).

$$\begin{aligned} DetectionTime_{ACTUAL} &= DetectionTime\_F_{Max} \cdot FrequencySlowdown \\ &= 28\,\text{ms} \cdot 2 = 56\,\text{ms} \end{aligned} \tag{4.6a}$$

$$\begin{aligned} CorrectionLatency_{Correctable,ACTUAL} &= CorrectionLatency_{Correctable,F_{Max}} \cdot FrequencySlowdown \\ &= 44\,\mu\text{s} \cdot 2 = 88\,\mu\text{s} \end{aligned} \tag{4.6b}$$

$$\begin{aligned} CorrectionLatency_{Uncorrectable,ACTUAL} &= CorrectionLatency_{Uncorrectable,F_{Max}} \cdot FrequencySlowdown \\ &= 22\,\mu\text{s} \cdot 2 = 44\,\mu\text{s} \end{aligned} \tag{4.6c}$$

$$\begin{aligned} ClassificationLatency_{Correctable,ACTUAL} &= ClassificationLatency_{Correctable,F_{Max}} \cdot FrequencySlowdown \\ &= 154\,\mu\text{s} \cdot 2 = 308\,\mu\text{s} \end{aligned} \tag{4.6d}$$

$$\begin{aligned} ClassificationLatency_{Uncorrectable,ACTUAL} &= ClassificationLatency_{Uncorrectable,F_{Max}} \cdot FrequencySlowdown \\ &= 5\,\mu\text{s} \cdot 2 = 10\,\mu\text{s} \end{aligned} \tag{4.6e}$$

Any error correction cycle contains the sequence of detection, correction and classification. If classification is disabled, the state changes will take the time, determined by $ClassificationLatency_{Uncorrectable,ACTUAL}$. Therefore, the cycle durations can be calculated as sum of $DetectionTime_{ACTUAL}$, $CorrectionLatency_{ACTUAL}$ and $ClassificationLatency_{ACTUAL}$. The results are given in table 4.1. As visible, the detection time with 56 ms is the major part of the total durations.

Table 4.1: Error mitigation durations for different error types and SEM-IP configurations.

		Error type	
		Correctable	Uncorrectable
Classification	Enabled	56.396 ms	56.054 ms
	Disabled	56.098 ms	56.054 ms

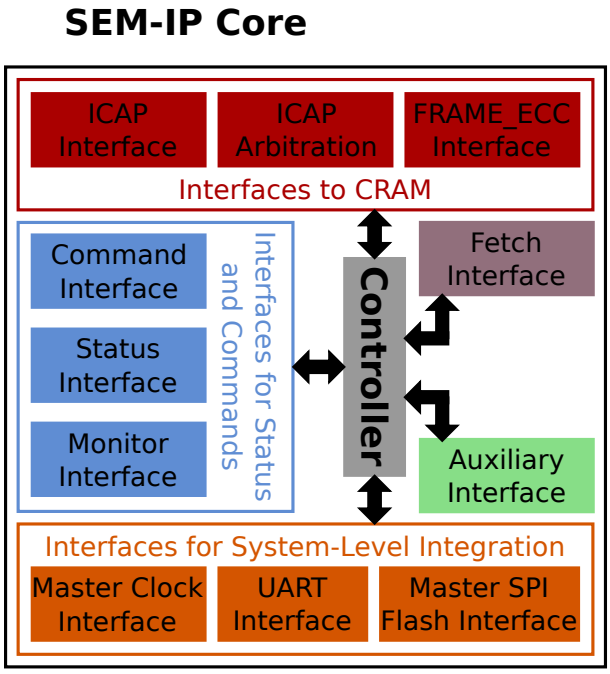

Figure 4.2: Interfaces of the SEM-IP core.

4.2.2 Structure and Submodules

The SEM-IP core consists of one controller, which is the central element, and eleven interfaces, depicted in figure 4.2. There, the interfaces are grouped by their purpose. At the top, the ICAP, ICAP arbitration and FRAME_ECC interfaces are directly connected to the MPSoCs configuration memory or CRAM control registers. Interfaces, used for communication with the SEM-IP are grouped to the left side. Through the command and monitor interfaces, instructions can be sent, while the status and monitor interfaces are used to track the overall operation. At the bottom, three interfaces to simplify the integration process are shown. Each of the remaining interfaces its own group. Through the auxiliary interface, the SEM-IP can be notified of SEEs, which were observed in other parts of the design and not visible to the controller. The last remaining interface, the fetch interface, is used to get data from external sources and forward it to the controller. This data contains information of essential configuration bits. The remainder of this section briefly summarizes the purpose of each component, to make the integration more intelligible [12].

As central component of the SEM-IP core, the **controller** deals not only with error correction and detection. The controller also communicates with the system, where the SEM-IP core is integrated. Therefore, the controller can interact with the command, status and monitor interfaces for low-level communication and with the UART interface to communicate at a higher level. If an external data source is enabled, the controller also initiates the fetch of data, needed for error classification. As with command interfaces, a low-level fetch interface and a high-level master SPI flash interface are provided to request data. Since the SEM-IP core can be used to control the overall system error mitigation strategy, external detected errors can be forwarded to the SEM controller via an auxiliary interface. Therefore, the controller is also in charge of processing external errors. To prevent the

user from wrong wiring, the controller is hidden inside the IP core.

In order to enable CRAM error detection and correction, the controller needs access to the MPSoCs configuration memory. That access is controlled by the **ICAP arbitration interface**. In there, logic for sharing the internal configuration access port (ICAP) is implemented. Thus, also other systems in the design are allowed to access the CRAM. As a side effect, the controller can be paused, when ICAP access is withdrawn. Since the user has to hand-off access, the ICAP arbitration interface is always visible and mandatory to use.

Whenever ICAP access is granted, the **ICAP interface** is used to access the CRAM. More precisely, data can by read from and written to the configuration memory. Because the read and written data should not be used outside of the controller, the ICAP interface is hidden in the SEM-IP core.

UltraScale+ MPSoCs use native mechanisms to correct errors. Their state is exposed to PL users via an MPSoC-internal FRAME_ECC interface. To notify the controller when errors are corrected or uncorrectable errors are found, the SEM-IP-local **FRAME_ECC** interface is present. Since other design parts can be connected to the MPSoC-internal FRAME_ECC interface, there is no need for the SEM-IP to expose its ports.

The **status interface** contains two signal groups. The first group contains a heartbeat signal, which can be used to ensure liveness of the controller. In the second group contains one signal for each state of the internal controller state machine. This group can be used to ensure correct state changes, which are meeting the specified timing requirements. The state machine is more precisely explained in section 4.2.3.

For debugging purposes, the **command interface** can be used to perform various operations on the controller. As the command interface receives the commands by logic values on wires, only an essential subset of all commands is available. These are error injection and reset of the controller. Since there is a more powerful interface to send instructions, the command interface can be completely removed by tying all input wires to low and leaving the output wires open.

For more detailed status information and a larger set of commands, the **monitor interface** can be used. When the monitor interface is used, instructions and status information are coded as ASCII strings. Therefore, it is more convenient and powerful than the status and command interfaces together. Thus, the monitor interface is always existent and exposed to the user. More details on the set of commands and structure of status information is given in section 4.4

To integrate the SEM-IP core on even higher abstraction levels, the monitor interface can be extended by the optional **UART interface**. In the UART interface, output from the monitor interface is serialized and inputs from the UART side are parallelized. Since UART is a widely used low-speed communication standard, which only needs two wires, the UART interface is suitable to observe the SEM-IP from external chips.

If error classification is enabled, the controller needs additional data, in order to detect errors as essential or not essential. Since this information must be stored at an external storage device, the controller needs access to this data via the **fetch interface**. This interface acts mainly as wrapper between the controller and external memory devices. Therefore, the user can implement an own

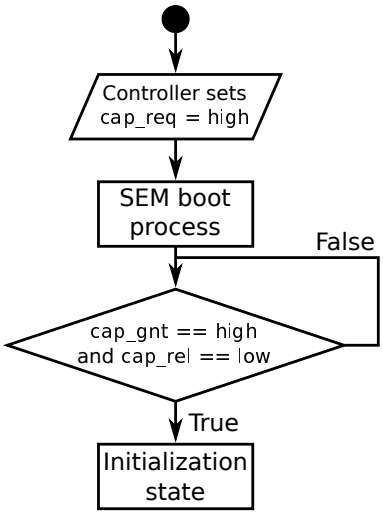

Figure 4.3: Start-up procedure of the controller.

memory access logic. Nevertheless, the fetch interface is only generated, if error classification is enabled.

Comparable to the UART block for the monitor interface, the **master SPI flash interface** is a specific implementation of memory access logic for the fetch interface. Thus, the master SPI flash interface implements a standard SPI bus protocol for the most commonly used type of memory. As this interface requires the existence of the fetch interface, it is only generated, when error classification is enabled.

In the case an error is detected in data or in external devices, the SEM-IP core offers the **auxiliary interface**. It can be used to notify the controller about these external issues. Indeed, the controller cannot correct these errors, but it can perform the same actions as for essential, non-essential or uncorrectable errors. Furthermore, external errors will result in a message on the monitor interface, which makes consistent data logs possible.

Finally, when integrating the SEM-IP core at system level, the **master clock interface** is used to derive all needed clocks. Whenever SEM-IP is integrated at lower levels, the clocks must be assigned manually and the master clock interface is not generated.

4.2.3 Operational Concept

Before the SEM-IP can begin its initialization process, ICAP access must be granted. As seen in figure 4.3, the controller sets the ICAP request signal *cap_req* to high, before the boot process is started. Then, the controller waits until the ICAP release signal *cap_rel* is reset to low and the ICAP access signal *cap_gnt* is high, which means, the controller is now in charge to use the internal configuration access port exclusively. When this condition is satisfied, the controller transitions to the initialization state [12].

The state after initialization as well as the set of valid states are determined by the SEM-IP con-

figuration. All six possible configurations modes are shown in table 4.2 [12]. Both mitigation modes start in observation mode and are compatible with error classification. Mitigation modes should be used, if errors should be detected and corrected automatically. They are just different in terms of error injection. If mitigation is not required, but errors should be detected, for example when characterizing a device according to its radiation environment, the detection modes can be used. For system testing, errors can be injected into the design without capabilities of detection or correction, in other words, test cases can be constructed. This can be done by operation the SEM-IP core in emulation mode. Finally, the monitoring mode can be used, if no errors should be detected, corrected, classified or injected. However, all debug features, namely commanding, frame address and register reads, external memory reads and address translations, are available. In addition, error detection on purpose and diagnostic scans can be performed, as they are grouped under on-demand detection.

Table 4.2: Configurations of the SEM-IP core.

Feature	Configuration modes					
	Mitigation and testing	Mitigation only	Detect and testing	Detect only	Emulation	Monitoring
State after initialization	Observation	Observation	Detection	Detection	Idle	Idle
Classification	Optional	Optional	N/A	N/A	N/A	N/A
Error injection	Yes	N/A	Yes	N/A	Yes	N/A
Debug features	Enabled					
On-demand detection	Enabled					

For EIVE, error mitigation is mandatory. Thus, either mitigation only or mitigation and testing modes are reasonable. As they only differ in their error injection functionality, which is important for testing, the mitigation and testing mode is used during implementation and validation. In space, the mitigation only mode is preferred, because error injection is neither necessary nor suggestive, since errors, induced by radiation impacts should be eliminated.

With the mode selected to mitigation, a state transition diagram during operation can be deduced. As visible in figure 4.4, the SEM controller will transition to observation state after the initialization is completed. It remains there, until an idle command is received or an error is detected. In the latter case, an error mitigation cycle, consisting of a sequence of correction state with unconditional transition to classification state, is performed. Then, the controller moves into the idle state if the error was uncorrectable, or otherwise back to observation. In the former case, when an idle command was received, the SEM controller immediately changes its state to idle. From idle, all other commands can be executed. Therefore, a reset command let the controller move to initialization and a diagnostic scan command moves the controller into diagnostic scan state. During a diagnostic scan, the configuration memory will be scanned exactly once, before the controller transitions back to the idle state. Another command moves the controller to the detect only state, which will return

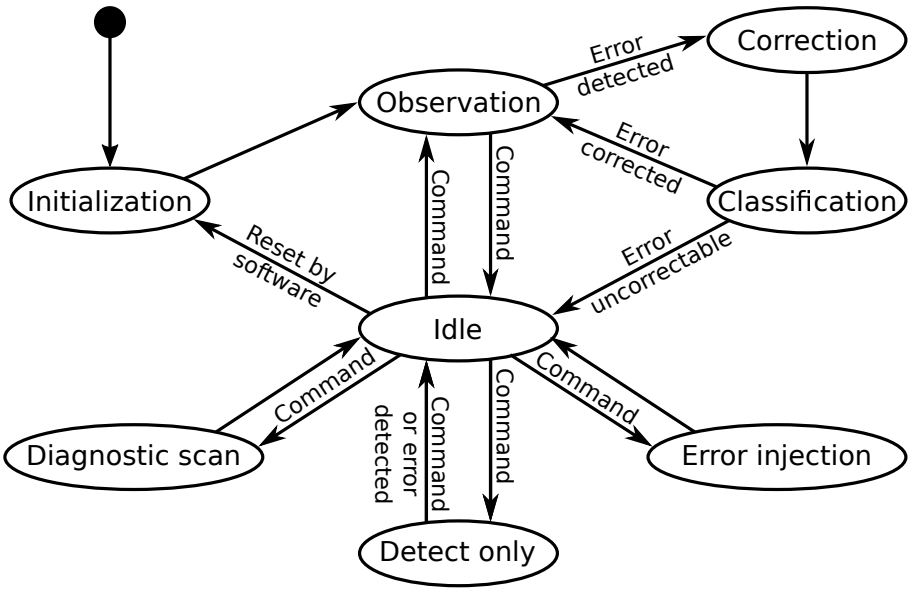

Figure 4.4: State transition diagram of the SEM controller in mitigation mode.

to idle whenever an idle command is received or a CRAM error is detected. The last command, accepted during idle, is error injection. As the name suggests, exactly one bit error will be injected into the MPSoCs configuration memory, before a transition back to idle takes place.

4.3 Hardware Integration

When integrating the SEM-IP core, some general design constraints, specific to EIVE and depicted in figure 4.5, must be fulfilled. As a first constraint, the use of SEM-IP is anticipated by the designers of multiMIND. Therefore, an UART interface for communication between the supervisor from TAS and the SEM-IP core is provided. Consequently, the UART interface is used with the SEM's transmit port connected to the supervisors receive port. Nevertheless, SEM observation is not implemented in the supervisor at the time, when this thesis was written. Thus, the SEM receive port is tied to ground and the supervisors transmit port remains unconnected. In addition, the SEM supervisory software must be implemented in the MPSoCs processor system, described in detail in section 4.4, which is possible, but not ideal in terms of radiation resistance.

Especially for testing purposes, the SEM-IP must be able to receive commands like error injection or directed state changes. In an intuitive approach, these commands can be sent to SEM via the UART interface, because this interface is already available. However, this approach is not possible, because the SEM-to-supervisor UART is routed to pins of the MPSoC and connected to the supervisor on the multiMIND chip. This problem is solved by transmitting specially designed telecommands (TC) and receiving telemetry (TM) through the TM/TC subsystem, which is connected to the on-board computer during flight and the lab computer for testing. The TM/TC subsystem then filters out all SEM-specific commands and forwards their data to the SEM-IP core.

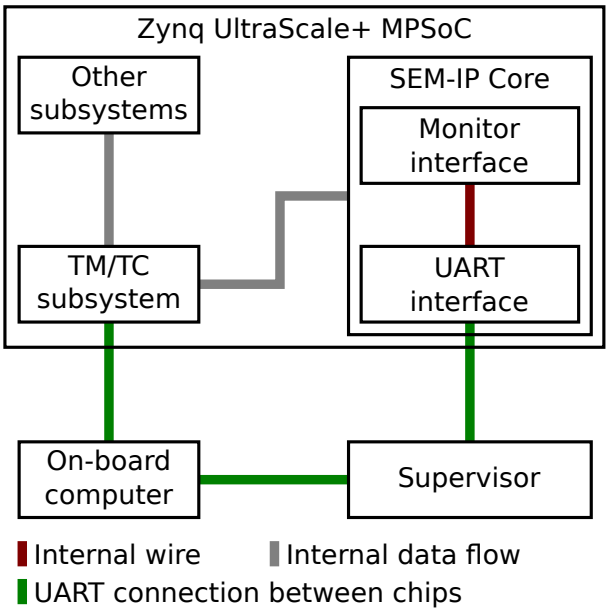

Figure 4.5: Connections of the SEM-IP core together with system overview.

Depending on the used SEM-interfaces and intermediate connection components, different integration approaches are available. At first, the low-level approach will be investigated. In there, only the most basic interfaces are used and wired to the processor system via GPIO pins. Thereafter, a high-level approach is developed. In contrast, the high-level approach makes also use of the more complex monitor interface and uses an AXI-based intermediate component to perform signal preprocessing and allows a register-based control of the SEM-IP core. Both approaches are using the mandatory ICAP and FRAME_ECC interfaces to access the configuration memory. Therefore, these two interfaces are not further mentioned in the remainder of this section.

4.3.1 Low-Level Integration Approach

Figure 4.6 shows the low-level SEM-IP integration concept, which makes use of the status, ICAP arbitration, command and UART interfaces. As already known, the UART interface provides SEM's log and status data to the supervisor without receiving instructions. Loosely speaking, the UART interface is used for simplex communication. Next, the ICAP arbitration interface is used to control SEM's access to the MPSoCs configuration memory. Since the SEM-IP core does not enter its initialization state before ICAP access is granted, the ICAP arbitration interface is used to start and disarm the SEM-IP. Since observation tasks always need information about current states and liveness, the status interface is used to provide those information. In the opposite direction, for sending commands to the SEM-IP, the command interface is used, since the basic command subset is sufficient for the low-level approach [23].

The signal preprocessing block, shown in figure 4.7 gets all output wires from the SEM status interface as its input signals and computes additional output signals. It is required to make some effects of SEMs operation visible in software. Since both, the PS and PL are running with a clock frequency

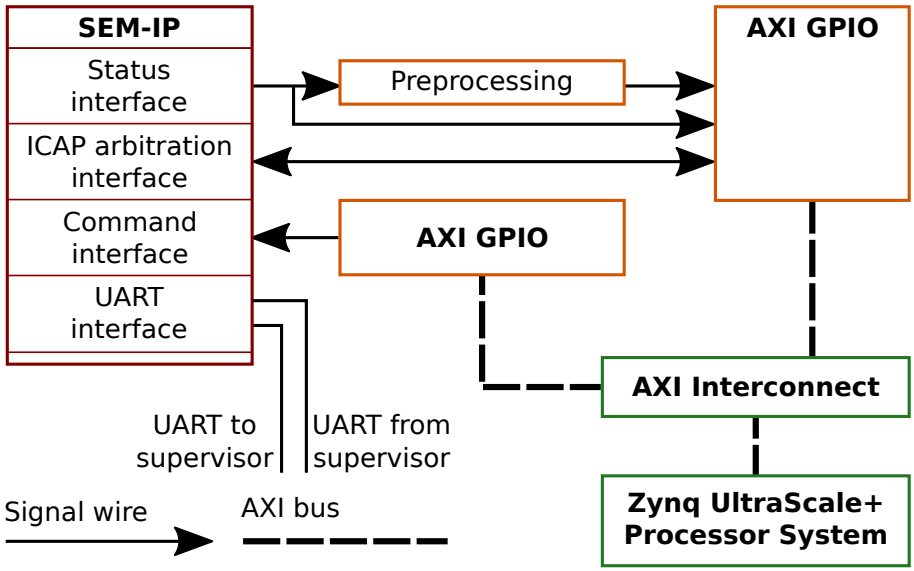

Figure 4.6: Low-level SEM-IP core integration.

of 100 MHz and one software instruction takes multiple clock cycles, short signal pulses cannot be observed in software. However, the *status_heartbeat* signal is specified to be high for one clock cycle and then low for at most 250000 cycles [12]. Therefore, the *heartbeat_timeout* signal is computed in the preprocessing block. The *heartbeat_timeout* signal goes high, whenever the *status_heartbeat* violates its specification and goes low, when the heartbeat is in or is returning back to specification. Furthermore, three additional status signals are computed. The SEM controller is in idle state, if *status_initialization*, *status_observation*, *status_correction*, *status_classification*, *status_injection*, *status_detect_only* and *status_diagnostic_scan* are low. In this case, *idle* is high. When all of these seven signals are high at the same time, the SEM controller is in fatal error state and *fatal_error* is asserted to high. In all other cases, exactly one out of these seven status signals is allowed to be asserted. However, if more than one and less than seven status signals are high, an internal error is detected and therefore, *internal_error* will be set. These three computations are done in hardware, because they result in almost no hardware overhead and the software is not slowed down. Since errors or heartbeat timeouts are sporadic events and the SEM controller may recover to ordinary operation, sticky signals are created to notify whether an error occurred at any time in the past. With these sticky signals, the SEM controller can manually be restarted in an organized way if desired.

To bring signals from the SEM-IP or preprocessing block to the processor system or vice versa, some sort of communication between them is necessary. Since signal wires are used at the programmable logic side and the Advanced eXtensible Interface (AXI) is used as primary data protocol in the Xilinx world, AXI GPIO blocks are used. They work like GPIO pins on a microcontroller. At the one side, AXI GPIO blocks offer GPIO pins, where signal wires can be connected, at the other side, they implement an AXI interface for connection to the processor system. However, if more than one AXI block should be connected, an AXI interconnect is mandatory to coordinate data transmissions.

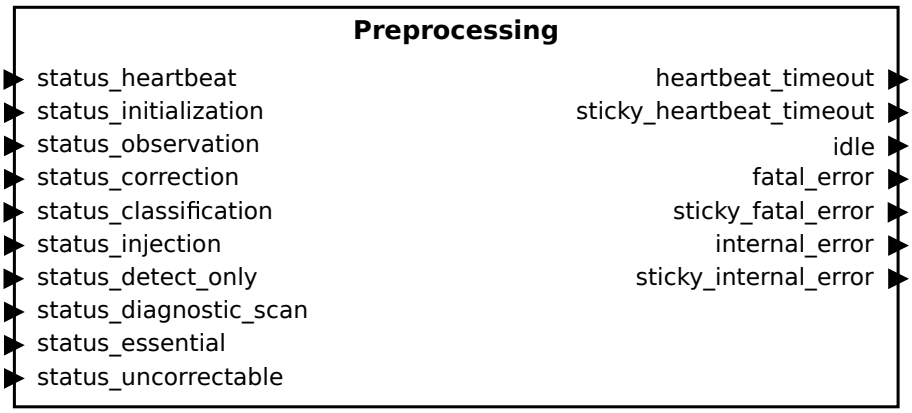

Figure 4.7: Signal preprocessing block, used for low-level SEM integration.

For the low-level approach, two AXI GPIO blocks are necessary, because one block can only offer 64 GPIO pins. In sum, the command interface uses 46 signals, the status interface 10, the ICAP arbitration interface 3 and the preprocessing block 7 signals, which results in a total of 66 needed GPIO pins. To make both AXI GPIO blocks distinguishable, one block is entirely used to send commands to the SEMs command interface. The other block receives information from the status interface and interacts with the ICAP arbitration interface. In summary, the SEM-IP core together with preprocessing logic is directly connected to both AXI GPIO blocks, which are connected to the processor system via an AXI interconnect block.

4.3.2 High-Level Integration Approach

Apart from the low-level integration approach, the SEM-IP core can be integrated at a higher level. As depicted in figure 4.8, the SEM-IP core is connected to the external supervisor via the UART interface, like in section 4.3.1. For receiving status information and accessing arbitration to the configuration memory, the status and ICAP arbitration interfaces are used. The only difference in the SEM-IP configuration is the replacement of the command interface with the monitor interface. Thus, the set of commands is extended and an ASCII-based log is available.

The high-level integration approach is built around the AXI2SEM module. This module takes the input from SEMs status, monitor and ICAP arbitration interface, processes the data and writes computed data to defined memory addresses in the processor system. In the AXI2SEM module, data processing refers to three tasks. First, signals from the status interface are processed and stored to a *Status Register*, which contains values of all status wires, except for the heartbeat signal. The heartbeat is replaced by a *heartbeat_present* field, which is 0, when the heartbeat was not high in the last second and 1 if the heartbeat is in its specification. In a similar way, signal values from the ICAP arbitration interface are mapped into the *ICAP Register* by the second task. As for all registers, a write of a value with a corresponding outgoing wire changes the value of this wire in hardware. For example, the AXI2SEM has an outgoing wire, called *icap_release*. If the corresponding *icap_release* field in the *ICAP Register* is set to 1, the wire will change its value to

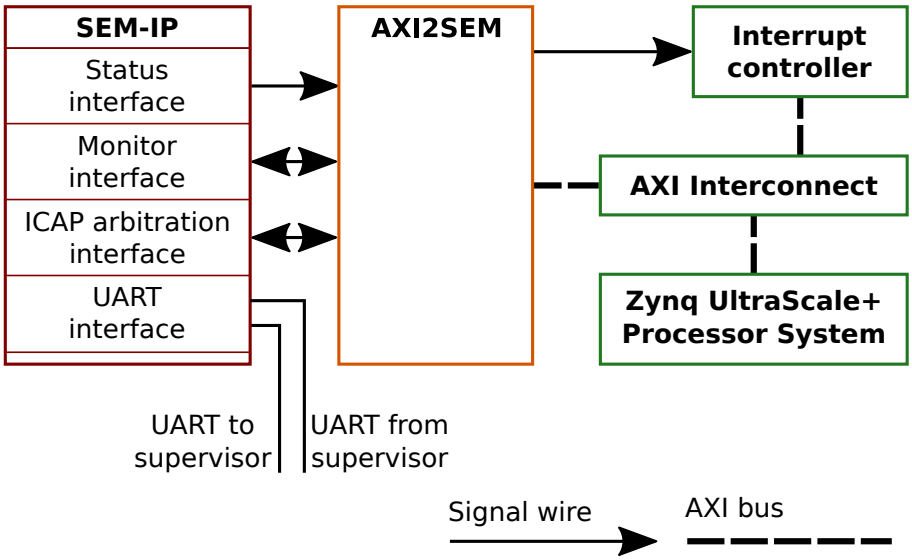

Figure 4.8: High-level SEM-IP core integration.

high. The third task deals with analysis of the log, provided by the monitor interface. After booting the SEM-IP core, the initialization log is analyzed by an ASCII parser. All information from this initialization log will be provided to software in the *SEM Status Register*. Apart from successful initialization or an error during initialization, the *SEM Status Register* allows the user to get more information about the context. After successful initialization, the entered state is shown to draw conclusions about the mode, in which SEM was configured. Afterwards, the log is continuously parsed for error messages and error correction reports, which are stored into the *Error Register*. In addition, the whole log is available to the user in a *Log FIFO Data* register with additional FIFO flags in the *Log FIFO Status* register. The other way around, commands can be sent to the monitor interface. Therefore, their instruction codes and data must be written into *Command Register*, which contains the command, *Command Base Register* for up to 32 bit of additional data and the *Command Extend Register* for 12 more bits of additional data. In order to tell the processor system about the moments for register readouts, interrupts are emitted by the AXI2SEM interface. Thus, whenever an error occurs, the controller's state is changed, the heartbeat times out or log data is available, an interrupt is raised to advise the processor system about new data [24].

In order to recognize interrupts in the processor system, an AXI interrupt controller is necessary. The interrupt controller takes the interrupt signal from the AXI2SEM block and translates this level-based interrupt into a edge-based interrupt, which is required by the processor system. At any interrupt the PS receives, the interrupt service routine (ISR) is executed and desired operations are processed.

4.3.3 Implemented hardware

While the low-level approach was well-suited to understand operational concepts, it yields to many problems during implementation. First, the 66 pins and two AXI GPIO blocks (see 4.3.1),

together with different behaviours between sticky and ordinary signals leads to confusion during hardware and software development. Also, this comes along with huge potential for accidentally errors, for example when a wire is accidentally connected in a wrong way or the wrong number for a GPIO pin is used. Secondly, the range of functions is restricted to the subset of commands, possible by the command interface. Not possible commands are for instance translations between physical and linear CRAM addresses or readouts of configuration frame s. The high-level approach has its main strength in testability. Because values of all wires from the status and ICAP interface are written into registers, ordinary software debugging tools can be used to view SEMs execution sequences. Furthermore, the monitor interface produces ASCII-based log reports, which can be directly written into a file. The same arguments applies to commands, which can now be sent as ASCII strings. These ASCII strings increase the readability and understanding of the SEM-IP operation further. One last advantage is caused by the interrupt based approach. While the low-level integration needs active polling for software monitoring, the high-level approach uses ISRs to execute code, whenever an interrupt is raised. By considering the ISR-based program flow to observe the SEM-IP core, the observation progress can be separated into an independent subsystem without any interactions to remaining PLOC software. In summary, the high-level and interrupt-based approach is much more suitable for EIVE and therefore implemented.

4.4 Software Integration

Since the SEM-IP core needs to be observed to ensure correct functionality, the observation needs to be implemented. For the most radiation tolerant implementation, possible in EIVE, the observation logic is implemented on a hard-wired processor in the MPSoC. As stated in 4.3.3, EIVE uses the high-level and interrupt-based SEM-integration approach. Therefore, the software integration investigates only this selected integration technique.

Most software require an initialization. This also applies to the SEM observing software. The first task, done during initialization, is determining a name for the log file. Therefore, the folder */SEM* is opened and the number of contained files is determined. Afterwards, the filename of the new file is created as concatenation of */SEM* and the number of contained files. For example, if */SEM* contains exactly three files when the PLOC is started, the SEM log file will be named */SEM/3*. At the next reboot, when four files already exist, the new file would be name */SEM/4*, and so on. Since these files are stored on the eMMC memory, ordinary telecommands for deleting and downloading can be used. However, SEM log files are typically smaller than 1 kB and therefore, at most 500 kB of log data are expected within the planned lifetime. Thereafter, the second task enables the AXI2SEM interrupts in order to make the interrupt-based approach working. The third and final step sets MPSoC internal ICAP access registers to grant ICAP access to the SEM controller. After this step, the SEM controller switches to initialization state.

During operation, interrupts are raised by the AXI2SEM module at following conditions:

- The log FIFO contains valid data, is full or in an overflow condition occurs

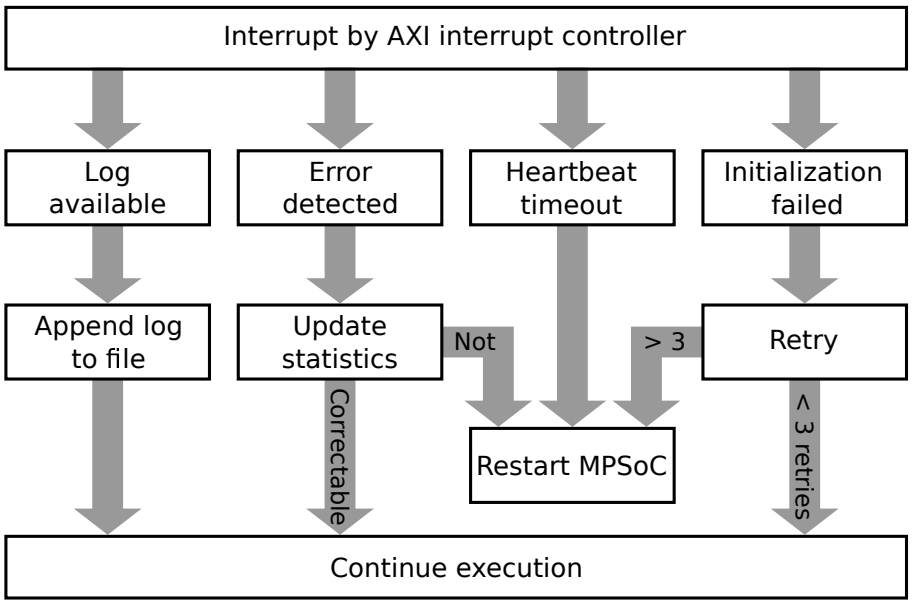

Figure 4.9: Interrupt-based integration of the SEM-IP in PLOC software.

- The heartbeat is out of its specification
- An error occurred during initialization
- The SEM controller changes to correction, initialization or uncorrectable state
- The SEM controller is ready to operate after initialization

All of these interrupts are caught in the same way, but they are threatened different. As seen in figure 4.9, there are four different ways of interrupt-based action. If log data is available, the log FIFO is full or an overflow condition is indicated, the valid data is read from the *Log FIFO Data* register and appended to the log file, determined during initialization. Whenever an error is detected, the error statistic fields in the housekeeping report are updated. These are the number of correctable and the number of uncorrectable errors since the MPSoC was powered on. Afterwards, the program execution is continued, if the error was corrected or the MPSoC is restarted, when the error was uncorrectable. In the third case, when the heartbeat is out-of-specification, the conclusion is an error in the SEM-IP core. Consequently, the MPSoC is rebooted immediately by setting the reboot flag in the housekeeping report. Thereafter, the MPSoC will be power-cycled by the on-board computer. Apart from errors during runtime, the SEM controllers initialization can also fail. In this case, the PLOC software tries to reset the SEM controller three times, before the MPSoC is restarted.

4.5 Testing

Testing of the SEM-IP core and its integration is very difficult, because the SEM controller cannot be simulated in a VHDL testbench. Therefore, hardware test data can be measured by integrated logic analyzers (ILA) or the overall behaviours can be carefully investigated. For EIVE, ILA testing

was only done when absolutely necessary, for example for finding bugs in the PL design. System testing was preferred, because it could be done at software level in the PS. Therefore, a debugger was used. Since the SEM-IP core is not interesting to debug and test for functionality when no CRAM errors occur, errors were injected into the configuration memory. As briefly mentioned in section 4.3, telecommands and telemetry packages are used to view into the soft error mitigation solution. The set of TM/TC consists of two telecommands and one telemetry package, summarized in table 4.3. As result, errors can be injected by using the *TC_SEMSendCommand* command and correct functionality can be observed by reading the log with the *TC_SEMGetUARTLog* command.

Table 4.3: Telecommands and telemetry packages used for system-level testing of the SEM integration.

Name	Type	Description
TC_SEMGetUARTLog	Telecommand	Request the new log data since the last log was requested.
TC_SEMSendCommand	Telecommand	Sends one or more commands, which should be executed by the SEM controller. If more than one command is sent, commands are separated by a semicolon.
TM_SEMUARTLog	Telemetry	Answer telemetry package for TM_SEMGetUARTLog. This TC returns at maximum 255 characters of the log.

In order to avoid failures during testing procedures, the EIVE test environment was extended by a radiation toolbox. This toolbox enables input of all SEM commands by either pressing buttons in a graphical user interface or entering them as text. Furthermore, the log can be automatically requested after each command and the log will be automatically processed and annotated for better readability. Available commands are listed in table 4.4. To get the whole SEM log of a test, the log file can be downloaded from memory. A screenshot of the radiation toolbox is given in 4.10. In the top row, the basic operations are aligned. These are requesting a log, requesting a status report and sending entered text commands. The second row contains the directed state changes to idle, observation, detect only and diagnostic scan, as well as a button for a software reset of the SEM controller. Error injection, frame reads and address translations can be performed within the third group. In addition, address parameters for these operations can be safely entered there. The last functionality, register reads, can be performed by entering the register number and pressing the read register button in the fourth row. Below, the SEM controller's log and sent operations are grouped by time and processed for better readability. Thus, any row contains exactly one line of log or one instruction, the corresponding length in characters and a brief description of the log line or instruction. During flight, the SEM-log can be downloaded from the eMMC memory like any other file.

Table 4.4: Commands for the SEM controller.

Command	Arguments	Description
O	-	Directed state change to observation
I	-	Directed state change to idle
D	-	Directed state change to detect only
U	-	Directed state change to diagnostic scan
S	-	Request status report
N	LFA	Inject error by using a linear frame address (LFA) with 11 hexadecimal digits
Q	LFA/PFA	Read configuration frame at linear/physical frame address with 11 hexadecimal digits
P	REG	Reads the SEM configuration register with name REG
R	XX	Software reset of the SEM controller. XX are two don't care hexadecimal digits
T	LFA/PFA	Translate LFA to PFA or PFA to LFA

4.5.1 Test Scenario 1: Initialization

In a first test scenario, the SEM initialization should be evaluated. If a valid SEM log arrives in the test environment, the conclusions are successful initialization of the SEM-IP core, a working *S*-command and a well-implemented TM_SEMUARTLog telemetry package. The output of the SEM-IP core after this test is shown in table 4.5. Since the initialization report contains the same data as printed in the IP documentation [12], it can be concluded, that the SEM controller initializes and the *S*-command works as intended.

Table 4.5: SEM initialization report.

Log line	Description
SEM_ULTRA_V3_1	SEM version string
SC 01	State change to initialization
FS 04	Core configuration information (not further documented)
AF 01	Additional core configuration information (not further documented)
ICAP OK	Access to ICAP granted
RDBK OK	Frame read-back is active
INIT OK	Initialization was successful
SC 02	State change to observation
O>	Command prompt for observation state

4.5.2 Test Scenario 2: Injection and Correction of a Correctable Error

In this second test, a correctable error is injected into the CRAM. Therefore, the SEM controller is directed into idle mode by an *I*-command. In idle mode, it is possible to read the configuration frame before injecting an error, by sending a *Q*-command. Thereafter, the error is injected by a *N*-command, before the frame is read again. A comparison of the read frame before and after error injection makes sure, that the error was injected. In order to start a correction cycle, the SEM

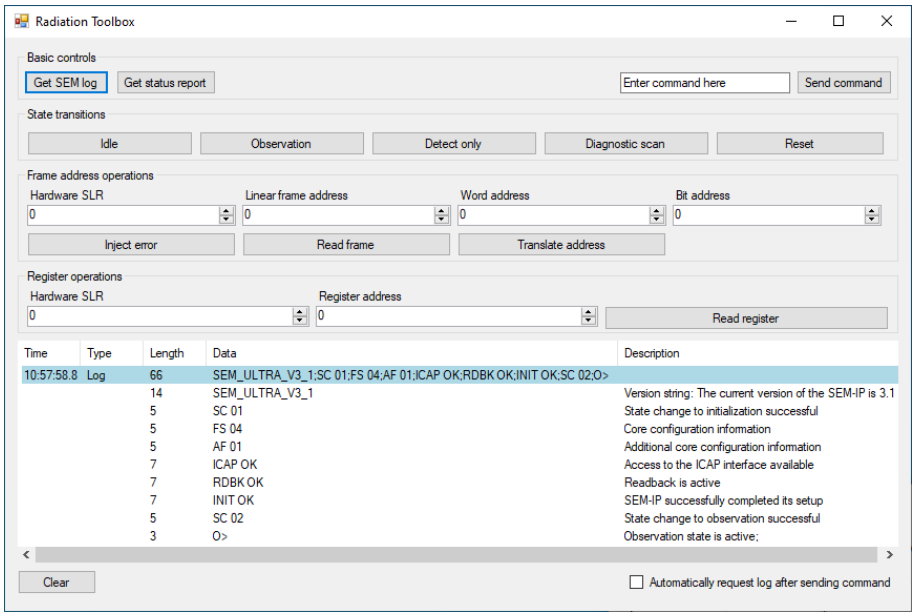

Figure 4.10: Radiation toolbox in the EIVE test environment.

controller is directed back into observation mode by sending the *O*-command. By requesting the SEM log by a *S*-command, the error correction report, visible in 4.6, can be read and analyzed. The correction report in table 4.6 indicates, that exactly one error at word 0, bit 0 in the configuration frame A098 was corrected. This test scenario is repeated at ten different configuration memory frames. Since all the tests result in exactly one corrected error, the conclusions are working *I*, *Q*, *N*, *O* and *S* commands as well as precisely working error correction, and thus a working integration.

Table 4.6: SEM correction report for exactly one correctable error.

Log line	Description
O>	Command prompt for observation state
RI 00	Reserved information
SC 04	State change to correction
ECC	An ECC error was detected
TS 0000171D	Timestamp
PA 00180200	Physical frame address of the detected error
LA 0000A098	Linear frame address of the detected error
COR	Begin of the correction report
WD 00 BT 00	The error was at word 0, bit 0
END	End of the correction report
FC 00	The error was correctable and non-essential (initial value)
SC 08	State change to classification
FC 40	The error was correctable and essential
SC 02	State change to observation
O>	Command prompt for observation state

4.5.3 Test Scenario 3: Injection and Correction of a Uncorrectable Error

The third test deals with injected and uncorrectable errors. In order to make this test, five errors are injected into one configuration frame. As in the second scenario, the controller is commanded into idle state before the configuration frame is read. Thereafter, five one-bit errors are injected by five *N*-commands. Then, the frame is read once again to make sure, that all errors are existent. Finally, the SEM controller is directed back into observation state and the log is requested. Clearly visible in the correction report in table 4.7, there is an uncorrectable CRC error. In addition, the reboot flag in the housekeeping report is now required to be set. Since this is the case, the conclusion is a successful handling of uncorrectable errors.

Table 4.7: SEM correction report for an uncorrectable CRC error.

Log line	Description
O>	Command prompt for observation state
RI 00	Reserved information
SC 04	State change to correction
CRC	A CRC error was detection (error position unknown)
TS 00002143	Timestamp
FC 60	The error was uncorrectable and essential (initial value)
SC 08	State change to classification
FC 60	The error was uncorrectable and essential
SC 00	State change to idle
I>	Command prompt for idle state

4.5.4 Testing Summary

The last two test are repeated for ten different frames, words and bit positions. Since all repeated tests for same test cases yield to equal results, the SEM-IP core works well in conclusion. Furthermore, all other commands are executed at least once. Because there are no problems with any commands, the inference is the proper integration of the SEM-IP core into the EIVE payload computer.

5 Error Detection and Correction Codes for EIVE

Errors can be introduced into data by a variety of effects. For example, during transmission, the channel can induce flipped bits or for single-event upsets, stored data can be corrupted. In order to mitigate these errors, two different approaches are possible. For data transmission purposes, errors can be detected and thus retransmitted by the sender without the need of correction. Such codes are named error detection codes. However, stored data cannot be retransmitted. Therefore, error correction codes (ECC) are required.

5.1 Theory of EDAC

To describe and compare EDAC codes, a mathematical concept, based on Hamming weight and Hamming distance is used. For any given bit vector $\vec{x} = [x_1, x_2, ..., x_N]$ of length N with $x_i \in \{0,1\}$, the Hamming weight can be defined as $W_H(\vec{x}) = |\{i|x_i \neq 0\}|$. In words, the Hamming weight is the number of all non-zero bits in a given bit vector. If an additional and equally defined bit vector $\vec{y}$ is given, the Hamming distance between $\vec{x}$ and $\vec{y}$ is defined as $d_H(\vec{x},\vec{y}) = W_H(\vec{x}+\vec{y}) = |\{i|x_i \neq y_i\}|$. Thus, the Hamming distance is the number of different bits between two bit vectors. In addition, the minimum Hamming distance d_{min} of a set of bit vectors $C = \{\vec{x_1}, \vec{x_2}, ..., \vec{x_M}\}$ can be determined as $d_{min} = \min_{\forall \vec{x},\vec{y} \in C, \vec{x} \neq \vec{y}} d_H(\vec{x},\vec{y})$. With this minimum distance, the number of detectable errors is $s = d_{min} - 1$ and the number of correctable errors is $t = \lfloor \frac{d_{min}-1}{2} \rfloor$ [25].

In figure 5.1, a block diagram of the encoding and decoding process is given. There, source bits are split up into blocks $\vec{U}$ of size k. Then, the encoder generates one codeword $\vec{X}$ of length n from each block $\vec{U}$ by adding exactly $n-k$ redundant bits. Thereafter, the codeword passes a channel, for example a transmission channel or an irradiated storage element. In this channel, errors can affect the codeword. Hence, a modified codeword $\vec{Y}$ is received. As a last step, the received codeword $\vec{Y}$ is decoded to data $\vec{V}$ of length k and passed to the destination [25].

The encoder uses a codebook, which consists of all $M = 2^k$ distinct data blocks and their corresponding codewords. An example codebook for $k = 2$ is given in table 5.1. The code $C(n,k)$ itself is characterized by the codeword length n and source block size k. Hence, the efficiency of the code can be expressed by the coding rate $R = \frac{k}{n} \leq 1$. For coding rates near 1, almost no redundancy is added and therefore, huge amounts of data can be transmitted with low correction capabilities. For coding rates near zero, almost no information is contained, but the original information can be reconstructed in almost any case. This trade-off between transmission and correction capabilities can be solved by selecting a suitable code. However, there is the upper singleton bound $d_{min} < n - k + 1$,

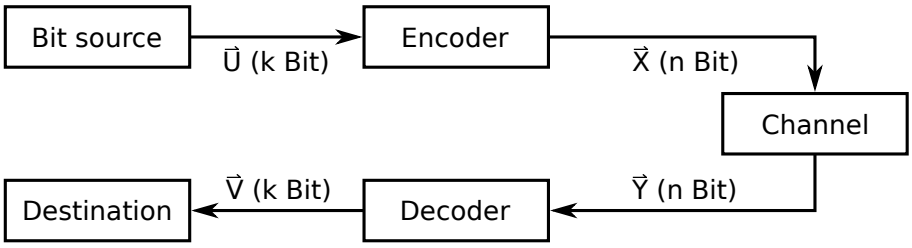

Figure 5.1: Block diagram for encoding and decoding.

which cannot be exceeded by any code [25].

Table 5.1: Example codebook for a code with $k = 2$ and $n = 3$.

Index	Source $\vec{U}$		Codeword $\vec{X}$		
	U_1	U_2	X_1	X_2	X_3
1	0	0	0	0	0
2	0	1	0	1	1
3	1	0	1	0	1
4	1	1	1	1	0

Since not all possible codewords are valid, the decoder has degrees of freedom. These degrees of freedom are used, when the received codeword $\vec{Y}$ is not valid in the used and known codebook. Therefore, the decoder divides the set of all possible, maybe invalid codewords, $\vec{Y} = [Y_1, Y_2, ..., Y_n]$ into exactly $M = 2^k$ decision regions $\{D_1, D_2, ..., D_M\}$. The decision regions must satisfy two properties: First, they must be disjoint and second, the union of all decision regions must fill the complete space, spanned by all possible codewords. Thus, D_i is a subset of the space spanned by $\vec{Y_1}, \vec{Y_2}, ..., \vec{Y_{2^k}}$. For decoding a received codeword, the source bits $\vec{U_i}$ are chosen, if $\vec{Y} \in D_i$. In words, if the received codeword is contained in the i-th decision region, the i-th source bits are chosen. For illustration purposes, an example for $k = 2$ and $n = 3$ is given in figure 5.2. There, all possible $2^n = 2^3 = 8$ possible codewords are plotted in a Karnough map. Since there are more codewords than possible source bit combinations, at least one decision region must contain more than one codeword. If a codeword $\vec{Y} = [Y_1, Y_2, Y_3] = [0, 1, 0]$ is received, it belongs to the decision region D_4. According to the codebook in table 5.1, the decoded bits will be determined to $\vec{V} = [V_1, V_2] = [1, 1]$, because they are at index 4 [25].

5.1.1 Error Detection Codes

Error detection codes can be split up into three groups: Parity check, checksum and cyclic redundancy check (CRC). In addition, error correction codes are also able to detect errors, but they are covered in section 5.1.2.

A parity check is the most simple error detection code. It is applied on a data block $\vec{U}$ with a fixed length k. Then, the encoder calculates the number of ones $a = \sum_i Y_i$, contained in $\vec{Y}$ and adds

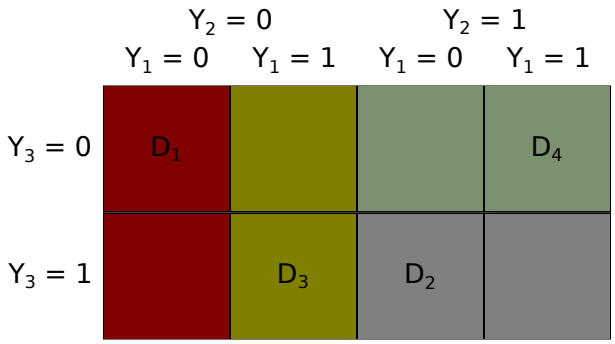

Figure 5.2: Illustration of decoding regions.

one extra bit E. For an even parity check, the extra bit is 0, if a is even and 1, if a is odd. For an odd parity check and odd a, the extra bit is 0, otherwise 1. Thus, the codeword is determined by $\vec{X} = [X_1, X_2, ..., X_k, E]$ with a length $n = k+1$ and therefore, the code rate is determined to $R = \frac{k}{n} = \frac{k}{k+1}$. During decoding, the parity bit F for the sub-vector $\hat{\vec{Y}} = [Y_1, Y_2, ..., Y_k]$ is calculated. When $E = F$ holds, the received data is assumed to be correct. In the other case, wrong data was received. Parity checks are able to detect one bit error per one data block [26].

Checksums are applied across data blocks. Therefore, the incoming set of data $\vec{U}$ is divided into K segments $\vec{S_i}$ with $i \in \{1, K\}$ of size l. Then, a defined operation $\odot$ is applied to determine the checksum $\vec{c} = \vec{S_1} \odot \vec{S_2} \odot ... \odot \vec{S_K}$ of length l. In the last encoding step, the codeword of size $n = (K+1) \cdot l$ is assembled by $\vec{Y} = [Y_1, Y_2, ..., Y_{(K+1)\cdot l}] = [\vec{S_1}, \vec{S_2}, ..., \vec{S_K}, \vec{c}]$. Thus, the code rate is $R = \frac{k}{n} = \frac{K \cdot l}{(K+1)\cdot l} = \frac{K}{K+1}$. For decoding, the same concept, used by parity checks is applied. First, the checksum of $\hat{\vec{Y}} = [Y_1, Y_2, ..., Y_{K*l}]$ is calculated and compared to $\vec{c}$ in a second step. For equality, the received data is assumed to be correct, for inequality, an error was induced [26].

Cyclic redundancy checks, (CRC) are extending parity checks by using more than one check bit. CRC is based on polynomial division. Therefore, the input vector $\vec{U} = [U_1, U_2, ..., U_k]$ is represented as polynomial $U(x) = U_k \cdot x^{k-1} + U_{k-1} \cdot x^{x-2} + ... + U_2 \cdot x + U_1$ and extended by l 0-bits to $\hat{U}(x)$. Afterwards, $\hat{U}(x)$ is divided by the l-th order CRC polynomial $P(x) = p_l \cdot x^l + p_{l-1} \cdot x^{l-1} + ... + p_2 \cdot x^2 + p_1 \cdot x + 1$. The remainder $c(x) = c_{l-1} \cdot x^{l-1} + ... + c_2 \cdot x^2 + c_1 \cdot x + c_0$ is the CRC polynomial. Since all coefficients are either 0 or 1, the can be written as binary vector. Thus, the $n = k + l$ bits long codeword is $\vec{X} = [U_1, U_2, ..., U_k, c_0, c_1, ..., c_{l-1}]$. Hence, the coding rate is $R = \frac{k}{n} = \frac{k}{k+l}$. When decoding, again the CRC of $\hat{\vec{Y}} = [Y_1, Y_2, ..., Y_k]$ is calculated and compared for equality to $[Y_{k+1}, Y_{k+2}, ..., Y_{k+l}]$. The received data is corrupted, if the comparison fails, otherwise, the data is assumed to be right. An example calculation for encoding is given in the remainder of this paragraph [26].

Source data block of length $k = 8$

$$\begin{aligned}\vec{U} &= [1,0,0,1,0,1,0,1] \\ U(x) &= 1\cdot x^7 + 0\cdot x^6 + 0\cdot x^5 + 1\cdot x^4 + 0\cdot x^3 + 1\cdot x^2 + 0\cdot x + 1 \\ &= x^7 + x^4 + x^2 + 1\end{aligned}$$

Used CRC polynomial of length $l = 5$

$$\begin{aligned}P(x) &= 1\cdot x^5 + 0\cdot x^4 + 0\cdot x^3 + 1\cdot x^2 + 0\cdot x + 1 \\ &= x^5 + x^2 + 1\end{aligned}$$

Extended source data of length $\hat{k} = k + l$

$$\begin{aligned}\hat{\vec{U}} &= [1,0,0,1,0,1,0,1,0,0,0,0,0] \\ \hat{U}(x) &= x^{12} + x^9 + x^7 + x^5\end{aligned}$$

Remainder of $\hat{U}(x)$ by $P(x)$

$$\begin{array}{llll} x^{12} & +x^9 & +x^7 & +x^5 \\ \cline{1-3} x^{12} & +x^9 & +x^7 & \end{array} \qquad \text{rem } x^5 + x^2 + 1$$

$$c(x) = \begin{array}{ll} x^5 & \\ \cline{1-2} x^2 & +1 \end{array}$$

Generating codeword $\vec{X}$

$$\begin{aligned}\vec{X} &= [X_1, X_2, ..., X_{13}] \\ &= [U_1, U_2, ..., U_8, c_1, c_2, ..., c_5] \\ &= [1,0,0,1,0,1,0,1,0,0,1,0,1]\end{aligned}$$

5.1.2 Error Correction Codes

Error correction codes are divided up, according to their encoding principle. On the one hand, block codes are able to encode a fixed length data vector. Block codes are mainly used for applications, where data can be divided without performance loss. For example, electronic memory devices use

block codes to protect their stored data. On the other hand, convolutional codes are mainly used for data transfer applications, since the output bit is described as function of the preceding input bits. Because the scope of this thesis is radiation mitigation and not data transfer, only block codes are investigated in this section [25].

5.1.2.1 Repetition Codes

The most simple code family in block codes are repetition codes. Repetition codes encode any source bit separately, by copying the bit value l times. Therefore, exactly two source symbols, 0 and 1, exist and k can be determined as $k = 1$. All other parameters are dependent on the number of repetitions l. Since $k = 1$, the codeword length n and the minimum Hamming distance d_{min} are fixed to $n = d_{min} = l$. Thus, repetitions codes have a code rate of $R = \frac{k}{n} = \frac{1}{l}$, can detect up to $d_{min} - 1 = l - 1$ errors and correct $\lfloor \frac{d_{min}-1}{2} \rfloor = \lfloor \frac{l-1}{2} \rfloor$ errors. For example, if one error should be corrected, the minimum distance must be three. Consequently, every bit is triplicated, and an increase in memory usage of 200 % for two extra bits can be observed. The decoding process is as simple as the block code itself. Because the decoded codeword $\vec{V}$ can either be 0 or 1, its value can be determined by majority voting of the received bits $\vec{Y}$ [27].

5.1.2.2 Parity Check Codes

All error correction codes add redundancy in a specific way. While repetition codes are copying bits, parity check codes are adding one or more single parity check bits for distinct subsets of the information bits. For example, two redundant bits $\vec{p} = [p_1, p_2]$ can be added to four information bits $\vec{U} = [1,0,0,1]$ by the parity equations $p_1 = u_1 \oplus u_3 = 1$ and $p_2 = u_1 \oplus u_2 \oplus u_4 = 0$. The $\oplus$-operator is used for a single-bit exclusive-or (xor) operation. As result, the codeword is $\vec{X} = [\vec{U}, \vec{p}] = [1,0,0,1,1,0]$, with a length of $n = k + m = 4 + 2 = 6$. The calculation of parity bits can be extended to codebooks by introducing a parity check matrix $\boldsymbol{H} \in \mathbb{B}^{m \times n}$ with the binary space $\mathbb{B} \in 0,1$. $\boldsymbol{H}$ has n columns, each containing a valid sequence of $m = n - k$ bits. To calculate parity check equations, columns can be swapped until $\boldsymbol{H}$ gets the systematic shape

$$\begin{aligned} \boldsymbol{H} &= [\boldsymbol{P}^T; \boldsymbol{I}] \\ &= \begin{pmatrix} P_{11} & P_{21} & \cdots & P_{k1} & 1 & 0 & \dots & 0 \\ P_{12} & P_{22} & \cdots & P_{k2} & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ P_{1m} & P_{2m} & \cdots & P_{km} & 0 & 0 & \dots & 1 \end{pmatrix} \end{aligned}$$

However, this reshaping is only possible, if all columns of the parity check matrix are distinct. In a next step, the parity check equation can be directly read from the matrix by row-wise xor-operations of the first k columns. For instance, the first parity check equation is $P_{11} \cdot U_1 \odot P_{21} \cdot U_2 \odot \dots \odot P_{k1} \cdot U_k = p_1$ and the second one is $P_{12} \cdot U_1 \odot P_{22} \cdot U_2 \odot \dots \odot P_{k2} \cdot U_k = p_2$ [25].

For encoding, two different approaches can be taken. The first one uses a generator matrix

$\boldsymbol{G} = [\boldsymbol{I}_{k\times k}; \boldsymbol{P}] \in \mathbb{B}^{k\times n}$. With this generator matrix, the codewords can be directly calculated as $\vec{X} = \vec{U} \cdot \boldsymbol{G}$. The second approach deals with the parity check matrix $\boldsymbol{H}$ itself. Since k source bits out of n bits per codeword are known, they can be directly inserted into the parity check equations. Therefore, a further reshaping of $\boldsymbol{H}$ into the upper triangle matrix is convenient, because the computational complexity to solve the equation system is reduced. This reshaped parity check matrix is

$$\boldsymbol{H'} = \begin{pmatrix} 1 & H_{12} & H_{13} & H_{14} & \dots & H_{1n} \\ 0 & 1 & H_{23} & H_{24} & \dots & H_{2n} \\ \vdots & \ddots & \ddots & H_{14} & & \vdots \\ 0 & 0 & 0 & 1 & \dots & H_{mn} \end{pmatrix}$$

According to the $\boldsymbol{H'}$, a Tanner Graph with input nodes $\vec{X} = [X_1, X_2, ..., X_n]$ and check nodes $\vec{C} = [C_1, C_2, ..., C_m]$ can be drawn. The Tanner Graph connects the input node i to the check node j, whenever $\boldsymbol{H'}_{ji} = 1$. To encode data, all known source bits are assigned to the input nodes $[X_{n-k+1}, X_{n-k+2}, ..., X_n]$. Then, the unknown values for $[X_1, X_2, ..., X_{n-k}]$ are calculated with $x_l = -\sum_{j=l+1}^{n-k} \boldsymbol{H}_{l,j} \cdot x_j - \sum_{j=l+1}^{k} \boldsymbol{H}_{l,j-n+k} \cdot x_j$, starting from $l = n-k$. This second approach has performance benefits for large codewords, but takes more time than multiplying data with a generator matrix for small codewords [25], [28], [29].

In order to decode received data, it is not sufficient to perform the inverse operation, because the data might be affected by the channel. For parity check codes, multiple ways of decoding exist, but not all decoding algorithms are working with all parity check matrices. First, an iterative approach is outlined. It consists of a Tanner graph with input nodes $\vec{Y} = [Y_1, Y_2, ..., Y_n]$ and check nodes $\vec{C} = [C_1, C_2, ..., C_m]$. The edges connect input nodes at position i to check nodes at position j, if the known parity check matrix $\boldsymbol{H}$ contains an one at row j and column i. To initialize the Tanner graph, bit values of the received codeword are assigned to input nodes. Next, the decoding algorithm starts by passing all bit values to the connected check nodes. In the check nodes, the parity is calculated and compared to the known parity value. Then, unknown bit values are set to satisfy the parity constraint, before the new signal values are propagated back to the input nodes. After that, the cycle restarts and continues, until the values in input nodes do not change [25], [29].

The second approach for decoding is called syndrome decoding. Unlike the first method, syndrome decoding is not iterative and can thus be performed in a constant time. The idea of syndrome decoding is, that all correctable errors are spanning a new set of modified codewords. If a codeword of a such coset is received, it can be decoded as the corresponding entry in the not modified set [25]. As example, a code with a Hamming distance of $d_{min} = 3$, which encodes $k = 4$ source bits into $n = 7$ codeword bits is used. Thus, the parity check matrix is computed to

$$\boldsymbol{H_{ex}} = \begin{pmatrix} 1 & 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 \end{pmatrix}$$

Because the Hamming distance is three, exactly one error can be corrected. Therefore, the coset leaders $\vec{e}$ can only differ from the zero-codeword in one bit. All other elements in each coset can be calculated by performing a xor-operation of the unaffected codeword with the leader of the corresponding set. To avoid searching and calculation of the whole coset contents, syndromes are introduced as coset indices. Each syndrome is calculated by $\vec{s} = \boldsymbol{H} \cdot \vec{e}^T$. For the example, the syndrome-coset-leader table is shown in table 5.2. If a codeword $\vec{Y}$ is received, its syndrome is calculated by $\vec{s} = \boldsymbol{H} \cdot \vec{y}^T$. In the next step, the corresponding coset leader $\vec{e}$ is determined by searching the syndrome-coset-leader table. Since the coset leader corresponds to the induced error, the sent codeword can be restored by $\hat{\vec{X}} = \vec{Y} + \vec{e}$, before the data is decoded to $\vec{V}$ by removing the m redundant bits from $\hat{\vec{X}}$ [25], [28].

Table 5.2: Syndrome-coset-leader table of $d_{min} = 3$, $n = 7$ and $k = 4$.

Syndrome	Coset leader
000	0000000
110	1000000
101	0100000
011	0010000
111	0001000
100	0000100
010	0000010
001	0000001

Apart from encoding and decoding algorithms, parity check codes have most degrees of freedom in the selection of the parity check matrix $\boldsymbol{H}$. For **Hamming codes**, $\boldsymbol{H}$ is designed in a way, such that the parameter $l \geq 3$ determines the dimensions to $\boldsymbol{H} \in \mathbb{B}^{m \times n} = \mathbb{B}^{n-k \times n} = \mathbb{B}^{l \times 2^l - 1}$ and $d_{min} = 3$. Thus, Hamming codes are able to detect two and correct one error in a transmitted codeword of size $n = 2^l - 1$, regardless of the parameter l. The code rate is increasing with increasing block size and determined by $R = \frac{k}{n} = \frac{2^l - l - 1}{2^l - 1}$. Nevertheless, more errors per fixed number of bits can be corrected and the encoding and decoding efficiency is higher for small block sizes. Furthermore, direct encoding with a generator matrix and syndrome decoding are applicable for all Hamming codes [25], [30].

The Golay code is the only known code, which can correct three or less errors at random locations. It takes $k = 12$ information bits and generates codewords of size $n = 23$ with a minimum Hamming distance of $d_{min} = 7$. In contrast to Hamming codes, which are a code family, the Golay code is one single code. Therefore, there is no need to know the construction process of $\boldsymbol{H}$, because is can be copied [30], [31].

Low-density parity check (LDPC) codes are another parity check based code family. The low-density term in the name comes from the fact, that the parity check matrix is sparse. They are characterized by three parameters from the equation $m = n \cdot \frac{l}{r}$, with the number of redundant bits m, the codeword length n, the number of 1's in each column l and the number of 1's in each row

r. The coding rate is defined as $R = 1 - \frac{m}{n} = 1 - \frac{l}{r}$. Since the general layout is not much constrained, LDPC codes can be designed to fulfil special needs, for example a specific minimum Hamming distance. LDPC-codes are exclusively used in combination of iterative encoding and decoding algorithms. This is due to the sparse nature of the parity check matrix and hence a dense generator matrix, which is not efficient for longer codewords [25], [29], [31].

5.2 Choosing EDAC Approach for EIVE

In EIVE, EDAC codes are used at two different locations. First, CRC codes are used to ensure correct transmission of telecommands and telemetry packages between the on-board computer and the MPSoC. This code was already used before this research thesis was started. The second code is used to protect E/W-band sample files in the eMMC flash memory and was implemented in the scope of this thesis. Therefore, it was necessary to select the used code and the way, in which one or more codes are applied.

5.2.1 Selection of Error Correction Code

The most important constraint is an excellent error correction capability. To compare the codes from above, table 5.3 shows the percentage of correctable bits in one codeword. As seen there, repetition codes have the highest error correction capabilities, but they need by far the most additional redundancy bits. In terms of error correctability, a Hamming code with $l = 3$ can correct only 14.3 % of all errors on average, but the size increase is much lower than for repetition or Golay codes. Furthermore, Hamming codes are not able to correct more errors if the codeword size increases. The Hamming distance for these codes remains exactly three, regardless of the parameter l. The last one, the Golay code can detect the most errors in one codeword, but because the codewords are comparably long, the correction percentage is not as good as for Hamming codes with $l = 3$. LDPC codes are omitted in table 5.3, because there is no general calculation for the minimum Hamming distance and thus for the number of correctable errors. However, LDPC codes can have arbitrary error correction capabilities, but with higher numbers of correctable errors, the codeword size increases fast.

Table 5.3: Comparison of relative amount of correctable errors.

Code	Parameters	n	d_{min}	Correctable bits	Correctable [%]	Overhead [32]
Repetition code	$l = 2$	2	2	0	0 %	100 %
Repetition code	$l = 3$	3	3	1	33.3 %	200 %
Repetition code	$l = 4$	4	4	1	25 %	300 %
Hamming code	$l = 3$	7	3	1	14.3 %	75 %
Hamming code	$l = 4$	15	3	1	6.7 %	36.4 %
Golay code		23	7	3	13 %	91.7 %

When comparing all four types of error correction codes to encoding and decoding speed, repetition

codes will be the fastest group. This is due to their simple majority voting logic. LDPC codes are again strongly depended on their design, but for huge source block sizes, they are faster than Hamming or Golay codes. This is, because LDPC codes are computed iteratively and expensive matrix computations are thus avoided. However, for small block sizes, LDPC codes are slower, because matrix computations on small blocks are faster than iterative algorithms. If the parameter l of Hamming codes is constrained to $l \leq 4$, Hamming codes are faster than Golay codes, because their parity check matrix is smaller. Vice versa, if $l \geq 5$, the parity check of Hamming codes will be larger. Hence, Golay codes are faster than Hamming codes for $l \geq 5$.

A problem for all error correction codes are burst errors. These are multiple bit errors in one codeword. To avoid their impacts on stored data, bit interleaving, known from section 2.3.1.1, is mandatory.

For EIVE, Hamming codes with $n = 7, k = 4$ and $d_{min} = 3$ are used as basis for encoding and decoding. This decision was made, because Hamming codes have the best relative error correction rate (14.3 %) over size overhead (75 %) quotient of 1,907 and the highest decoding speed, compared to other codes [33]. Therefore, Hamming codes are the best compromize between error correction and space overhead. Furthermore, Hamming codes can be encoded by a generator matrix and decoded by syndrome decoded in a constant time. This is important for EIVE, because decoding of samples is instructed by a telecommand, which has a specified timeout. Also, for a block size of four source bits and a codeword length of seven bits, encoding and decoding by iterative methods will be slower and will thus increase the power consumption.

5.2.2 Selection of Coding Sequence

In this section, encoding and decoding is performed by Hamming codes and bit interleaving is applied to some approaches. In total, five different encoding and decoding approaches were implemented in the scope of this thesis and compared according to their error correction capabilities and processing time. The five approaches are explained in table 5.4.

To evaluate all five approaches, they are simulated in the testbench, depicted in figure 5.3. There, a sample file is read, before small blocks of four bits are created for encoding. Then, the code under test (CUT) is used to encode the blocks and random errors, according to defined probabilities, are injected. Finally, the affected data is decoded and compared to the small data blocks for error correction analysis. To analyze the computation time, stopwatches are applied to the encoding and decoding steps. Because the computers cache is not hot at the beginning and other running programs may affect the runtime, the execution times may vary from try to try. Therefore, timing is averaged over ten encoding/decoding cycles. For evaluation purposes, timing statistics and error statistics, which contain the number of injected and corrected i-bit errors, are written to a CSV-file. To identify test runs, additional testbench configuration information like error probabilities for i-bit errors, the sample file name and the approach name are also written to the CSV file.

Figures 5.4, 5.5 and 5.6 show the simulation results of all five approaches with an base bit error probability of $B = 0.05$, $B = 0.1$ and $B = 0.2$, respectively. In all three figures, the two subplots on

Table 5.4: Hamming code based coding approaches for E/W-band sample protection.

Number	Encoding	Decoding
1	Blockwise Hamming encoding	Blockwise syndrome decoding
2	Blockwise Hamming encoding Interleaving	Deinterleaving Blockwise syndrome decoding
3	Blockwise Hamming encoding Interleaving Blockwise Hamming encoding	Blockwise syndrome decoding Deinterleaving Blockwise syndrome decoding
4	Make blocks of size 4 by 4 Row-wise Hamming encoding Column-wise Hamming encoding Interleaving Blockwise Hamming encoding	Blockwise syndrome decoding Deinterleaving Make blocks of size 7 by 7 Column-wise syndrome decoding Row-wise syndrome decoding Comparison
5	Make blocks of size 4 by 4 Row-wise Hamming encoding Column-wise Hamming encoding Interleaving	Deinterleaving Make blocks of size 7 by 7 Column-wise syndrome decoding Row-wise syndrome decoding Comparison

the top are showing the percentage off corrected errors on the y-axis over the error probabilities for i-bit errors on the x-axis. The higher the percentage, the more induced errors were corrected. The two subplots at the bottom share the same x-axis, but the y-axis shows the sum of averaged measured encoding and decoding time. Below, the x-axis is labelled with probabilities for error injection. For instance, if in 10 % of all coded words an 1-bit error and in 2.5 % a 2-bit error was injected during simulation, the corresponding x-label row contains the values 10.00, 2.50 and five time 0.00 from top to bottom. However, for the plots on the left side, only on kind of errors was induced with the base error probability B, while the right side shows statistics of combinations of induced errors with probabilities $b = \frac{B}{i^2}$ for i-bit errors.

The figures 5.4, 5.5 and 5.6 are looking similar in a first view. Especially the bottom subplots are nearly identical. Thus, the computation time can be considered as constant, except for the two outliers at approach four and the small outlier at approach three. From a timing perspective, the approach one is the best, closely followed by approach two, where additional bit interleaving is applied. Furthermore, these two lines point out, that the computational complexity is mainly caused by encoding and decoding, not by interleaving and deinterleaving. When adding a second encoding step in approaches three and four, the execution time is doubled up. Because both of these approaches are using bit interleaving, no difference in execution time can be measured. Since the fifth approach encodes and decodes all data a third time, its computation time is multiplied by four, compared to the first and second approaches.

By comparing the error correction rate, an 100 % correction of zero- and one-bit errors can be observed for all approaches, except for the second approach with one-bit errors. This is caused by the final interleaving, because injected errors can now affect multiple interleaved bits of one

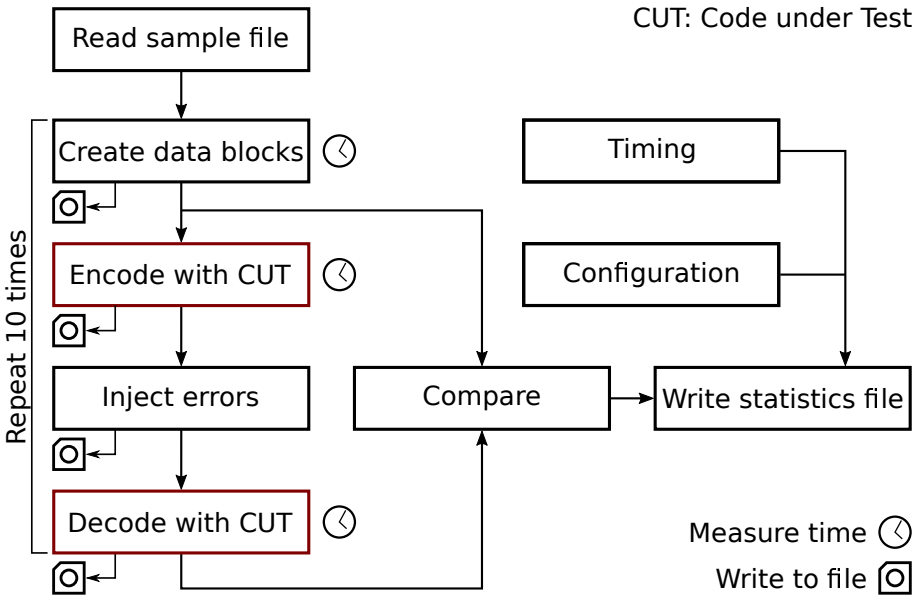

Figure 5.3: Testbench to compare coding approaches.

word. Thus, deinterleaving takes this distributed errors and may move them into one single code word. For all higher numbers of bit errors per codeword, none of the used code has a guarantee to correct them. Nevertheless, approach one has the worst performance, followed by approach two. Approaches three and four are equally good and approach five has the best error correction capabilities. While the error correction rate curves have the same overall tendencies and behaviours, their values are decreasing with and increasing base error probability B.

One advantage of approach four and five, which is not visible in the plots is the comparably high error detection capability. While the used Hamming code can only detect two bit errors in a codeword of seven bits length, approaches four and five are able to effectively detect more flipped bits, depending on their alignment. This is, because the blocks are two-dimensional with row-wise and column-wise encoded data, like schematically shown in figure 5.7. If the comparator detects a mismatch between row-decoded and column-decoded source bits, an error is captured [34].

By weighting all advantages and disadvantages, the approach four is the most suitable code. This is caused by the encoding and decoding speed, which is not as slow as for approach five, and the high error detection capabilities of two-dimensional data blocks.

5.3 Software Design

All error correction code approaches were implemented in the scope of this thesis. The programming language was determined to C, because C is used for the whole software system. Since the PLOC software is bare metal, conversion of MATLAB code into C-code is error-prone and therefore, all five approaches were directly implemented in C.

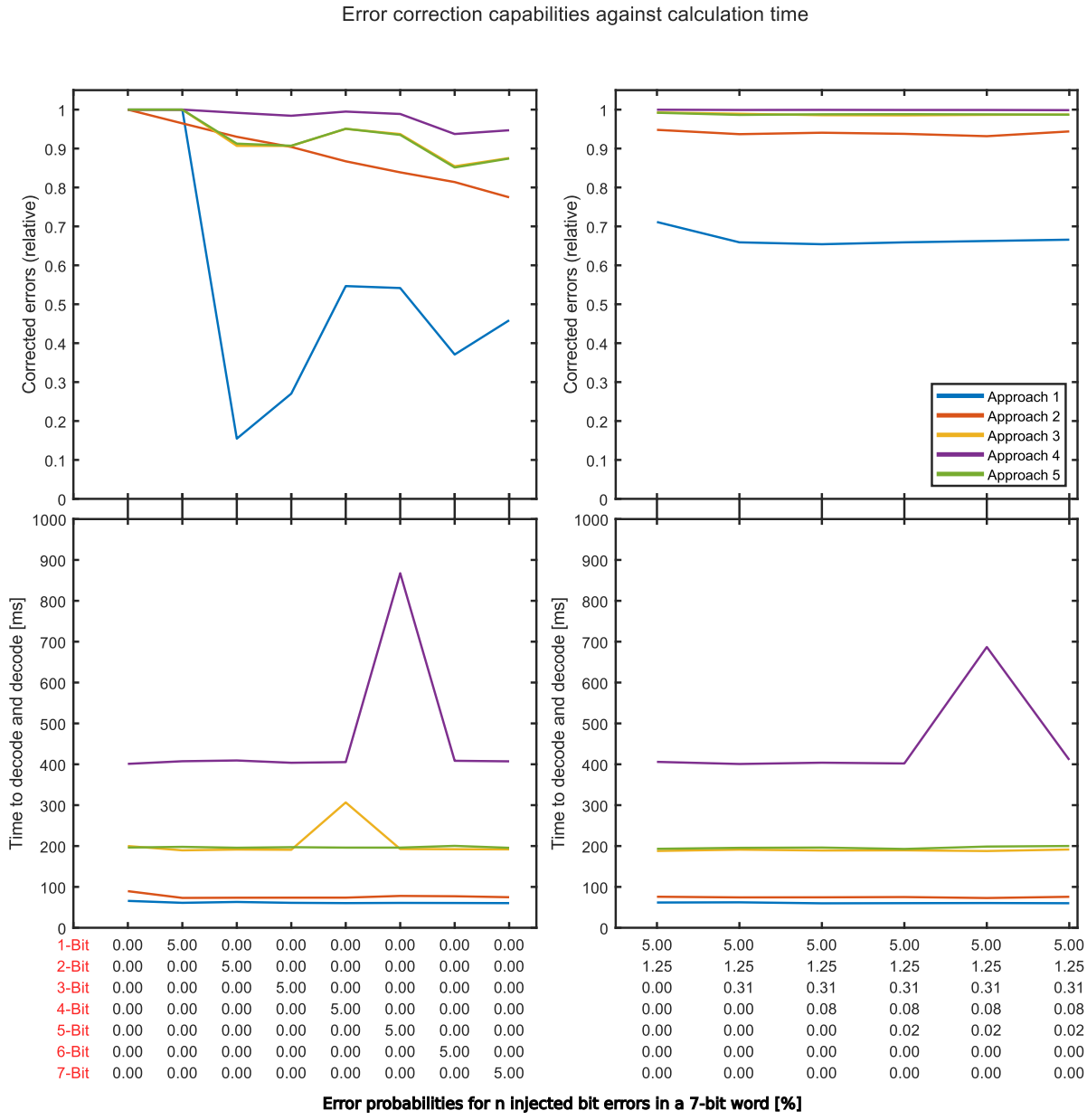

Figure 5.4: Error correction capabilities against computation time at $B = 0.05$.

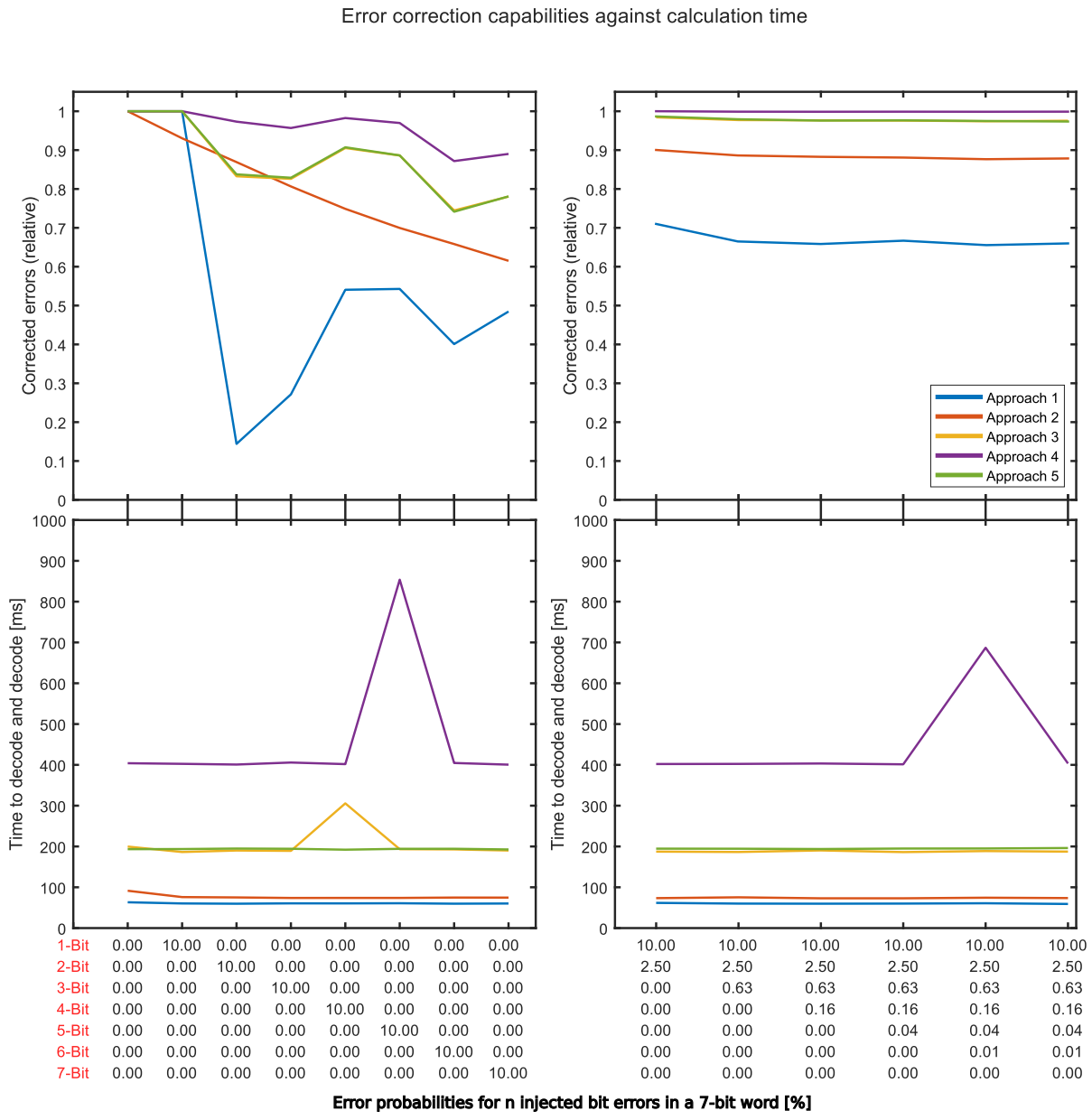

Figure 5.5: Error correction capabilities against computation time at $B = 0.1$.

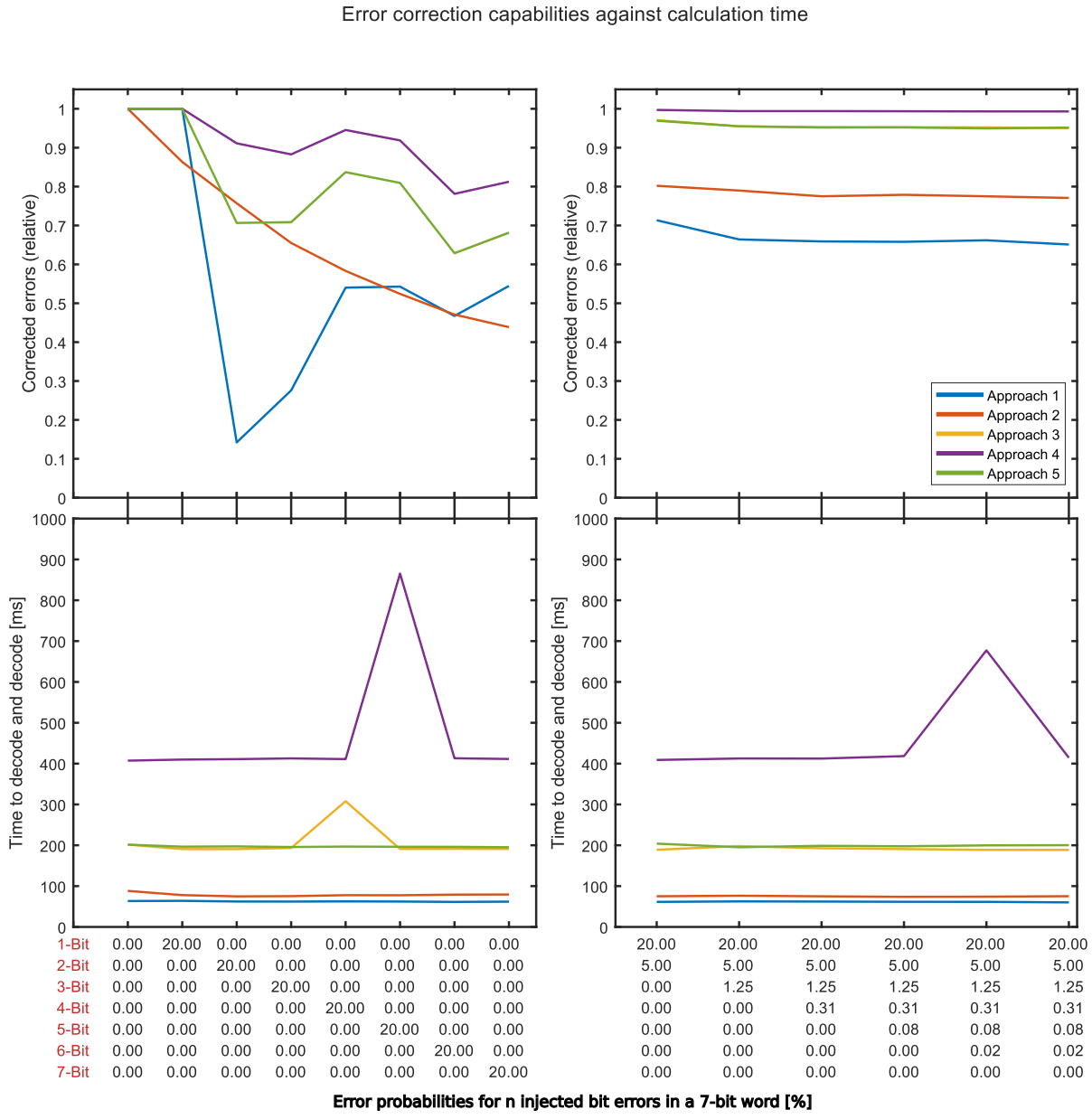

Figure 5.6: Error correction capabilities against computation time at $B = 0.2$.

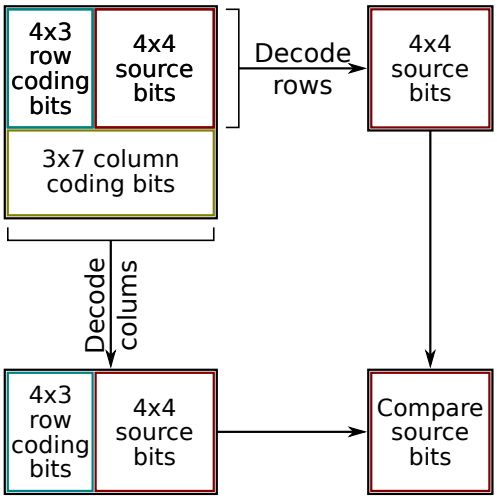

Figure 5.7: Decoding of two-dimensional blocks.

5.3.1 Modularization

For efficient and rapid implementation of all five coding approaches, modularization into building blocks was necessary. In general, the whole E/W-band sample coding system is built from data conditioners, interleavers, hamming codes, matrices and file interfaces. All five approaches can be easily implemented, by following the steps in table 5.4, since the building blocks must only be concatenated.

Matrices are not only necessary for encoding and decoding. They are used throughout the whole coding system to represent data and abstract from raw bit handling. Furthermore, they reduce the potential of errors, because each matrix has a defined interface and well-tested functions are used to access data and perform operations. Needed operations are listed in table 5.5. However, the matrix blocks are not visible to the user of encoding or decoding approaches. They only exist for internal representation of data. The matrix library inside the coding system is also the largest building block.

Hamming codes are the core for encoding and decoding. The Hamming code module requires an initialization to generate the parity check matrix $\boldsymbol{H}$, generator matrix $\boldsymbol{G}$ and the syndrome lookup table. For encoding, 8-bit wide integers, where only the least significant four bits are valid, are required as source bits $\vec{U}$. After encoding, one 8-bit wide integer value, with the least significant seven bits valid, is returned as codeword. For decoding, an 8-bit wide integer value with seven valid bits is required and a 8-bit wide integer with four valid bits together with an error indication variable is returned.

Interleavers are used to distribute neighbouring bits to distinct codewords, to avoid burst errors. They get a data matrix and the interleaving distance and return a matrix with interleaved data of

Table 5.5: Needed matrix operations for the coding subsystem.

Operation	Description
Horizontal concatenation	Concatenate matrices of dimensions $n \times m$ and $n \times l$ into $n \times l + m$
Copy	Copy matrix
Create matrix	Allocate space for new matrix and fill it with 0
Identity matrix	Create identity matrix
Multiply matrix	Matrix multiplication of two matrices
Number to row vector	Copy n least significant bits of an integer to a $1 \times n$ bit matrix
Row vector to number	Generate number with n valid bits from a $1 \times n$ bit matrix
Sub-matrix	Generate a sub-matrix from a defined area of a larger matrix
Transpose	Transpose matrix
Equality	Check two matrices or a scalar and a bit matrix for equality
Elementwise modulo	Perform modulo operation to any matrix element
Print	Print matrix to console
Swap rows	Swap rows in a matrix

exactly the same size. The same applies for deinterleaving, which needs an interleaved data matrix and a deinterleaving distance and returns the original data matrix.

Data conditioners can extend 8-bit integers matrices with eight valid bits per element into larger matrices, where less bits per 8-bit integer are valid. Conditioners are needed to prepare data for encoding and decoding, since there are four or seven bits expected, respectively. Also, after encoding or decoding, fully populated 8-bit integers are required to use the disk space efficiently or to make the coding transparent for further data processing. For example, after decoding, integers with four valid bytes must be transformed into fully populated 8-bit integers, to get valid data.

File interfaces are used to read files from a storage into a matrix or to write data from a matrix into a file. They abstract from basic I/O and basic byte copying from and to matrices.

Testing blocks are not part of the payload software. They are just used offline for testing. In sum, there are three testing blocks available. The approach wrapper, already known as code testbench, is introduced to automate testing and gain performance and error correction statistics of the used codes. To induce errors, the random error injection block is used. It is possible to give probabilities for one to seven bit errors per codeword. The last testing block is an ordinary timer, which is used to precisely measure time of encoding and decoding.

5.3.2 Integration

Integrating the coding system into the EIVE payload software is much more simple than integrating the SEM-IP core. This is caused by the software-design of the payload software, which is split up into an initialization stage and a while-loop for waiting until an UART interrupt is received. In contrast to the SEM-IP core, the coding system does not require any additional interrupts. The

coding system or more precisely the Hamming code is initialized during the initialization stage. Whenever an UART interrupt is received, the message is read by the exception handler and transformed into a telecommand structure. Thereafter, the telecommand is processed. The only telecommand, which is important for the coding system is *TC_ReplayWriteSeq*, which loads a file from the eMMC flash memory and writes it into the downlink memory buffer. When the samples are encoded, the file needs to be opened and additionally decoded, before the decoded content is written to the downlink memory buffer. If the file cannot be decoded, an error is returned by the *TM_TcExeFailure* telemetrie package and another file can be tried. In summary, the integration of the coding system only needs one change in the initialization routine and one change in the TC_ReplayWriteSeq routine.

6 Conclusion and Further Work

This thesis started with research of radiation environments, to define tasks related to radiation, which are faced by the EIVE mission. Then, in order to get an overview of possibilities to deal with the defined radiation tasks, an exhaustive research of radiation mitigation techniques was done. This research includes techniques, based on information-, spatial- and temporal redundancy, as well as techniques without need or redundancy and general designing guidelines. Based on this knowledge, all investigated radiation mitigation techniques were evaluated, according to the requirements of EIVE. Furthermore, the techniques are assigned to design stages, where they are applied and thus, assigned to an implementing institution. In sum, two techniques are implemented in the scope of this theses. The first implemented technique is the integration of the SEM-IP core into the EIVE hardware and software design, which prevents the MPSoC from malfunctions and potential damages. As a second technique, error correction codes were further investigated and Hamming codes with bit interleaving were implemented to prevent changes in the E/W-band validation samples.

However, not all selected techniques could be implemented in the scope of this thesis. Therefore, possible future works can implement information redundancy for all data files. This can be done by a global use of error correction codes for file reads and file writes. Another possibility will be to store each file three times, majority vote the contents on file reads and write the correct data back to the corrupted file. In a third solution, the two available eMMC memory chips can be configured in a redundant way. This will neither prevent from errors, nor enable majority voting, but in case of the E/W-band validation samples, the correct copy can be determined by the used error correction code.

For the case, when both eMMC memories have irreversible damages and no data can be read from or written to these storages, EIVE's mission is over. To cover this case, the payload software should be extended to not only use samples from memory. The software must gain the ability to generate a subset of the stored samples without the need for memory reads.

Another possibility to make the payload more robust against radiation impacts is to introduce more watchdog timers or heartbeat signals. For example, each subsystem, like the camera subsystem or the downlink subsystem, can be observed by a dedicated watchdog timer or generate its own heartbeat signal. In case of timeouts, partial reconfiguration can be used to rewrite the configuration memory contents of the defective subsystem.

Furthermore, partial software redundancy can be implemented, because the MPSoC's application processing unit contains four processor cores. Therefore, interrupts can be processed on one core

and forwarded to two other cores, which are running the same payload software, but without interrupt processing routines. In addition, to detect execution errors, the memory bus must be observed and in case of mismatches, the software must be reset to the latest checkpoint. However, this approach is very hard to implement, because the entire existing software must be reimplemented and comparison hardware must be built.

Bibliography

[1] I. Kallfass, L. Manoliu, B. Schoch, M. Koller, S. Klinkner, J. Freese, A. Tessmann, and R. Henneberger, "Towards the Exploratory In-Orbit Verification of an E/W-Band Satellite Communication Link," in *2021 IEEE MTT-S International Wireless Symposium (IWS)*, p. 1–3, 2021.

[2] L. Manoliu and F. Wiewel, "EIVE Payload Computer (PLOC) System Engineering Document," tech. rep., Institut für Robuste Leistungshalbleitersysteme, May 2020.

[3] A. Pawlitzki, F. Steinmetz, and T. A. S. Germany, "multiMIND – HIGH PERFORMANCE PROCESSING SYSTEM FOR ROBUST NEWSPACE PAYLOADS | Thales Alenia Space Germany," 06 2021.

[4] F. Boehringer, *EIVE Payload Processing Node - User Requirements Document*, 0.3 ed., 02 2020.

[5] K. A. LaBel, "Radiation Effects on Electronics 101: Simple Concepts and New Challenges," Apr. 2004.

[6] NASA, ed., *Space Radiation*. NASA Human Research Program Engagement and Communications, June 2017.

[7] ESA, ed., *Techniques for Radiation Effects Mitigation in ASICs and FPGAs Handbook*. Space Product Assurance, ESA Requirements Standards and Division, Sept. 2016.

[8] ESA, ed., *Space Environment*. Space Engineering, ESA Requirements Standards Division, Sept. 2008.

[9] S. Luz Maria Mertinez, *Analysis of LEO Radiation Environment and its Effects on Spacecraft's Critical Electronic Devices*. PhD thesis, Embry-Riddle Aeronautical University, Daytona Beach, Florida, Dec. 2011.

[10] S. Kasap, E. Wachter, X. Zhai, S. Ehsan, and K. McDonald-Maier, "Survey of Soft Error Mitigation Techniques applied to LEON3 Soft Processors on SRAM-based FPGAs," *IEEE Access*, vol. 8, 01 2020.

[11] J. Hussein and G. Swift, "Mitigating Single-Event Upsets," May 2015.

[12] Xilinx Inc., *UltraScale Architecture Soft Error Mitigation Controller v3.1*, Oct. 2021.

[13] Xilinx, "Zynq UltraScale+ MPSoC Data Sheet: Overview," tech. rep., Xilinx Inc., May 2021.

[14] "EIVE Orbitanalysen," tech. rep., Institut für Raumfahrtsysteme, Feb. 2020.

[15] E. Howell, "International Space Station: Facts, Histroy & Tracking." `https://www.space.com/16748-international-space-station.html`, 2021. Accessed: 2022-03-22.

[16] "Radiation Effects and Mitigation - An ESA/TAS/Uni Stuttgart Workshop," Mar. 2021.

[17] D. White, "Considerations Surrounding Single Event Effects in FPGAs, ASICs, and Processors," Mar. 2012.

[18] P. Shirvani, N. Saxena, and E. McCluskey, "Software-Implemented EDAC Protection Against SEUs," *IEEE Transactions on Reliability*, vol. 49, no. 3, p. 273–284, 2000.

[19] L. Sterpone, M. Reorda, and M. Violante, "RoRA: a reliability-oriented place and route algorithm for SRAM-based FPGAs," in *Research in Microelectronics and Electronics, 2005 PhD*, vol. 1, p. 173–176 vol.1, 2005.

[20] University of Stuttgart, *Link-Budget Analysis of W- and E-band SatelliteServices*, 2019.

[21] CCSDS, "Space Packet Protocol," 2020.

[22] M. Meyer, "Entwurf digitaler Systeme."

[23] M. Welter, *Integrating LogiCORE SEM IP in Zynq UltraScale+ Devices*, Nov. 2018.

[24] M. Welter, *Integrating LogiCORE SEM IP with AXI in Zynq UltraScale+ Devices*, Feb. 2017.

[25] V. Aref, "Information Theory and Coding."

[26] A. Kirstädter, "Kommunikationsnetze 1."

[27] A. Orlitsky, "Information Theory," in *Encyclopedia of Physical Science and Technology (Third Edition)* (R. A. Meyers, ed.), p. 751–769, New York: Academic Press, third edition ed., 2003.

[28] C. Hillier and V. Balyan, "Error Detection and Correction On-Board Nanosatellites Using Hamming Codes," *Journal of Electrical and Computer Engineering*, vol. 2019, 2019.

[29] P. H. Siegel, "An Introduction to Low-Density Parity-Check Codes."

[30] S. K. Buddha, "Hamming and Golay Codes."

[31] J. K. Wolf, "An Introduction to Error Correcting Codes Part 1."

[32] S. Ahmad, M. Zahra, S. Z. Farooq, and A. Zafar, "Comparison of EDAC schemes for DDR memory in space applications," in *2013 International Conference on Aerospace Science Engineering (ICASE)*, p. 1–5, 2013.

[33] P. Shirvani, N. Saxena, and E. McCluskey, "Software-implemented EDAC protection against SEUs," *IEEE Transactions on Reliability*, vol. 49, no. 3, p. 273–284, 2000.

[34] S. Jeon, B. V. K. Vijaya Kumar, E. Hwang, and M. K. Cheng, "Evaluation of Error-Correcting Codes for Radiation-Tolerant Memory," *The Interplanetary Network Progress Report*, vol. 42, 2010.

List of Abbreviations

AM	Ante Meridiem
ASCII	American Standard Code for Information Interchange
ASIC	Application Specific Integrated Circuit
AXI	Advanced eXtensible Interface
BRAM	Block Random Access Memory
BSP	Board Support Package
CLB	Configurable Logic Block
CMOS	Complementary Metal-Oxide-Semiconductor
COTS	Commercial Off-The-Shelf
CPU	Central Processing Unit
CRAM	Configuration Random Access Memory
CRC	Cyclic Redundancy Check
CSV	Comma Separated Value
CUT	Code Under Test
DAC	Digital to Analogue Converter
DCs	Digital Circuits
DD	Displacement Damage
DRAM	Dynamic Random Access Memory
DSP	Digital Signal Processing
ECC	Error Correction Code
EDAC	Error Detection and Correction Code
EEPROM	Electrically Erasable Programmable Read-Only Memory
EIVE	Explanatory In-orbit Verification of an E/w-band link
eMMC	Embedded MultiMediaCard
FIFO	First-In First-Out
FIT	Failures In Time
FPGA	Field Programmable Gate Array
FSM	Finite State Machine
full-HD	Full High Definition
GCR	Galactic Cosmic Ray
GPIO	General Purpose Input Output
HZE	Highly-charged and Energetic Particle
ICAP	Internal Configuration Access Port
ILA	Institut für Luftfahrtantriebe
ILH	Institut für robuste LeistungsHalbleitersysteme
IP	Intellectual Property
IRS	Institut für Raumfahrtsysteme
ISR	Interrupt Service Routine
ISS	International Space Station
LDPC	Low Density Parity Check
LEO	Low Earth Orbit

LFA	Linear Frame Address
LUT	LookUp Table
MBU	Multiple Bit Upset
MCU	Multiple Cell Upset
MLSL	Minimal Level Sensitive Latch
MPSoC	MultiProcessor System-On-Chip
MTBF	Mean Time Between Failures
NAND	Not AND
NOR	Not OR
NVM	Non-Volatile Memory
OBC	On-Board Computer
PFA	Physical Frame Address
PL	Programmable Logic
PLOC	PayLoad On-board Computer
PM	Post Meridiem
PRBS	Pseudo-Random Bit Sequence
PS	Processor System
RAM	Random Access Memory
RHBD	Radiation Hardening By Design
RHBP	Radiation Hardening By Process
RoRA	Reliability-Oriented place and Route Algorithm
SDRAM	Synchronous Dynamic Random Access Memory
SEB	Single-Event Burnout
SEDR	Single-Event Dielectric Rupture
SEE	Single-Event Effects
SEFI	Single-Event Functional Interrupt
SEGR	Single-Event Gate Rupture
SEHE	Single-Event Hard Error
SEL	Single-Event Latch-up
SEM	Soft Error Mitigation
SESB	Single-Event Snap Back
SET	Single-Event Transient
SEU	Single-Event Upset
SPE	Solar Particle Event
SPI	Serial Peripheral Interface
TAS	Thales Alenia Space
TC	TeleCommand
TID	Total Ionizing Dose
TM	Telemetry
TMR	Triple Modular Redundancy
TNID	Total Non-Ionizing Dose
TTR	Triple Temporal Redundancy
UART	Universal Asynchronous Receiver Transmitter
UV	UltraViolet
VHDL	Very high-speed integrated circuits Hardware Description Language