Fault-tolerant satellite computing with modern semiconductors

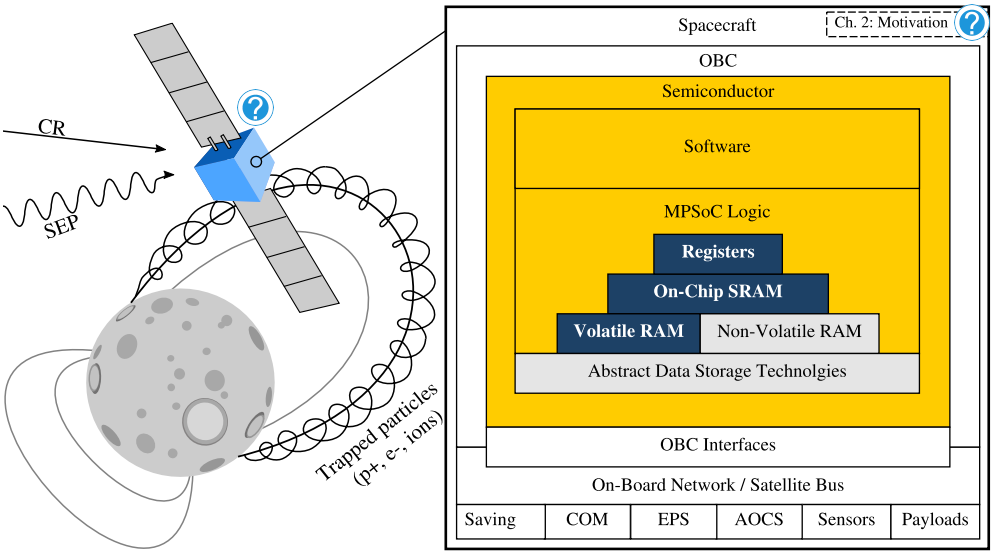
Fuchs, C.M.

Chapter 2

A Brief Introduction to Spaceflight and Fault Tolerance

Thesis Motivation and Legitimization

The research upon which this thesis is based does not come from one single field of science, but is interdisciplinary. It relies upon concepts and results from several different fields, including computer engineering, nuclear science, electrical engineering, physics and astronomy, as well as space engineering. In this chapter, we provide a brief and informal introduction to our application, its design constraints, as well as fault-tolerant computer architecture. We further provide an overview over the current status of small satellite space missions, as well as a review on satellite failures in the past and at the time of writing. This chapter therefore serves also as motivation and legitimization for our research, including mission success and failure statistics, which underline the lack of reliability of very small satellites today.



2.1 Spacecraft and Satellite Miniaturization

In this section, a brief introduction into the different kinds of satellites and satellite miniaturization itself is given, to provide general understanding for readers who are not familiar with this field. This section is meant as to give sufficient background information on the application for the research discussed in this thesis.

Satellites can be differentiated by mass in several classes.When thinking of space stations, satellites, and deep-space probes, we usually imagine large structures floating in space, weighing multiple tons, powered by vast solar panel arrays, radioisotope thermoelectric generators, or fission reactors [7]. Certainly, many early scientific, commercial, and military satellites were very large spacecraft. These are sometimes designed to operate for several decades in space. However, today, modern semiconductor technology, more efficient battery and photovoltaics, novel propulsion technologies, and robust lightweight materials enable the construction of much smaller, lighter, and cheaper spacecraft.

Spacecraft with a wet mass[1] of less than 500kg are therefore referred to as "miniaturized satellites", and can be constructed dramatically faster than large satellites. In Table 1, an overview over satellite classes and capabilities is given.

At the time of writing, several companies have achieved commercial success by operating large groups of miniaturized satellites in orbit. They have been successfully used to providing real-time earth observation data and help in disaster recovery [8], and in safety- and life-critical services [9] such as airplane traffic tracking and maritime shipping [10]. A broad variety of biological and chemical experiments [11] has been carried out using CubeSat platforms, which are also rather popular for testing and validating novel technologies in space [12, 13]. Several pico- and nanosatellite-based space-observatories [14, 15] have been launched, and nanosatellites were deployed by the Hayabusa 2 space probe at the asteroid *162173 Ryugu* [16]. In 2018, 2 interplanetary CubeSats traveled to the planet Mars as part of the MarCO mission [17],

[1]The mass of the spacecraft including payload and all consumables such as propellant.

	Weight		Minia-	Build as	Classical	Propulsion	Mission
Class	Max	Min	turized	CubeSat	Tech Usable	Available	Lengths
Large	-	1t	No	Absurd	Yes	Yes	Decades
Medium	1t	500kg	No	Absurd	Yes	Yes	Decades
Small	500kg	100kg	Yes	Limiting	Most	Yes	10 years
Micro	100kg	10kg	Yes	Common	Little	Yes	years
Nano	10kg	1kg	Yes	Standard	No	Yes	1 year
Picro	1kg	100g	Yes	Standard	No	Limited	months
Femto	100g	-	Yes	Inefficient	No	No	-

Table 1: Satellites can be classified in a variety of ways, with each type of spacecraft having different capabilities, technological limitations, and the capability to achieve different mission durations. In principle, almost any satellite could be manufactured to be a CubeSat, but only for some this makes sense due to the constraints of this form factor standard.

providing real-time telemetry during the arrival-phase of NASA's InSight Mars Lander. Several miniaturized satellite constellations for technology demonstration, and Earth observation, and positioning, and data relay purposes have been developed [18–21] and launched [8, 22, 23]. At the time of writing, scientists and engineers have even begun to develop CubeSat-based interferometers and composite space telescopes [13] that could outperform even the largest conventional space-observatories, and there are plan to use Nanosatellites even for gravitational-wave measurement [15].

2.1.1 Large Satellites based on Traditional Design Principles

Satellites with a wet mass above 500kg are at this point in time constructed in large projects with vast budgets quasi artisanally. Most "big-space" applications rely upon such satellites. Satellites of 500kg – 1000k are usually classified as *medium-sized satellites*, heavier spacecraft are designated as a *large satellites*. Development of such satellites is challenging, system architectures are complex, resulting in long development times, and the need to utilize well tested, proven technology, that is available over a very long period of time. This technology is usually space industry proprietary. Technology readiness, design maturity, and space heritage of a technology through prior use aboard other spacecraft are essential, and often seen a prerequisite for considering a technology for use within this satellite class.

Construction of these satellites in practice often takes many years [24], sometimes even decades [25]. To provide an example, the James Webb Space Telescope (JWST) is designed to have a wet mass of approximately 6620kg. It is a multinational project involving hundreds of stakeholders, and has been in construction for more than 25 years at the time of writing, and its precise date of completion and launch has not been announced yet. The cost of the electronics used aboard such a spacecraft is small compared to the funds required to meet legal requirements, for salaries, tooling, testing, management, certification, insurance, and launch. Spacecraft testing also requires access to specialized facilities [26, 27] including:

- thermal/vacuum chambers to analyze the behavior of the spacecraft in a space-like environment at high or low temperatures (often 173K and 373K) [28],
- radiation testing facilities using radiogenic sources or particle accelerator to simulate the radiation environment a satellite's components have to operate in, and to verify their correct behavior and, if available, effectiveness of fault tolerance measures, and
- a broad variety of other heavy machinery, e.g., to perform mechanical stress and vibration tests.

Most modern major launch vehicles can carry much heavier and bulkier loads than just one satellite [29, 30]. Often a substantial amount of volume and mass remains available which in the early days of spaceflight remained vacant to not endanger the primary payload [31]. To reduce costs, organizations often either sell this excess capacity, or hand the entire launch process over to a "launch broker", which then can combine multiple satellite launches into one "ride-share" launch [29]. An example of a ride-share launch with multiple satellites of various classes is depicted in Figure 3. The main spacecraft launched on a launch vehicle is then referred to as "primary payload", with other, often smaller satellites becoming "secondary payloads". Today

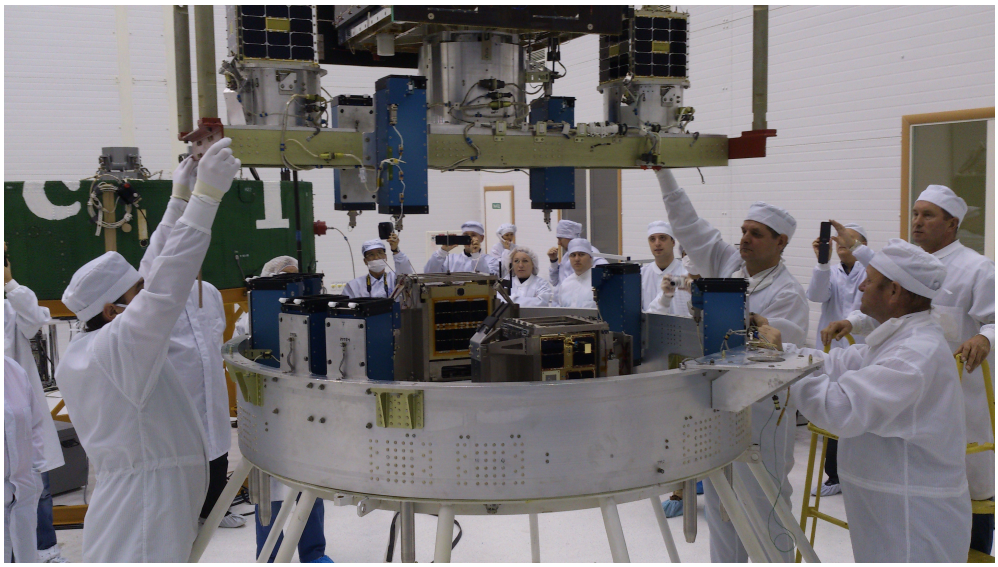


Figure 3: A ride-share satellite launch with the Earth observation SmallSat DubaiSat-2 (top center) being the primary payload. Secondary payloads were 4 microsatellites (top left and right, 2 bottom center) and 26 other nanosatellites which are located in the blue deployer boxes. The CubeSat First-MOVE (see Section 2.1.4) is located in the top right deployer.

even small start-up companies, and universities can bring their spacecraft into orbit at comparably low cost.

2.1.2 Small Satellites

SmallSats, or Minisatellites, weigh between 500 and 100kg, and traditionally were used for brief science and commercial missions. Historically, SmallSat missions used to be shorter than those realized with large satellites [32]. They can be constructed and launched at drastically lower cost, and in general also more quickly. The term SmallSat is colloquially also used to refer to *all* satellites lighter than 500kg in this field. Due to technological evolution in recent decades, the capabilities of the SmallSats have increased, and today they increasingly much replace larger satellites.

2.1.3 Microsatellites

MicroSats between 100kg and 10kg are today widely used for a variety of low cost commercial and novel scientific missions. The upper and lower boundaries between Nanosatellites, MicroSats, and SmallSats are fluent. MicroSats with a wet mass approaching 100kg differ little from lighter SmallSats, and usually carry fewer or lighter payloads and lighter components (e.g., smaller batteries, lighter and smaller solar cell array structures, ...) [33]. Light MicroSats become similar to a Nanosatellite and may even utilize Nanosatellite form factor standards, while larger ones can offer very similar capabilities to SmallSats. Many missions that a few decades ago required SmallSats can today be performed by MicroSats, which can be manufactured more rapidly and

launched at lower cost. Compare also [34] for a market assessment for a corporate view on this increasing down-scaling trend.

2.1.4 Nanosatellites and CubeSats

Nanosatellites weigh between 1 and 10kg and became popular for educational projects, especially due to the CubeSat standard. The CubeSat standard was originally intended to cheaply launch student projects into space at the beginning of the 21st century [35]. Today, it has become the standard form factor for Micro-, Nano-, and Picosatellites, and an example of a CubeSat is depicted in Figure 5. It requires a satellite to conform to certain design restrictions, e.g., banning the use of explosive substances within the satellite, and otherwise implies a stackable standard form-factor consisting of 10x10x10cm CubeSat units (U) and a maximum of 1.33 kg per 1U. CubeSats are designed to fit a standardized CubeSat *deployer*. Figure 4 depicts such a deployer consisting of a spring, and electric latch, which once the latch is released allows CubeSats to be safely be deployed by pushing them out of the box. This enables even heavy 12U or 24U designs (3x2x2 or 4x2x3U stacked) to be launched at reduced cost, and allows testing requirements to be reduced for launch qualification, as the failure of a CubeSat during launch will not interfere with the deployment of other satellites aboard the same launcher.

At the time of creation of the CubeSat standard, nanosatellites were intended to perform only simple and short missions in Low Earth Orbit (LEO), e.g., student education, or on-orbit concept validation. They rely on cheap commodity technologies and COTS components, such as lithium-polymer based batteries, and solar-cells intended for ground use. However, due to the rapidly increasing performance of embedded

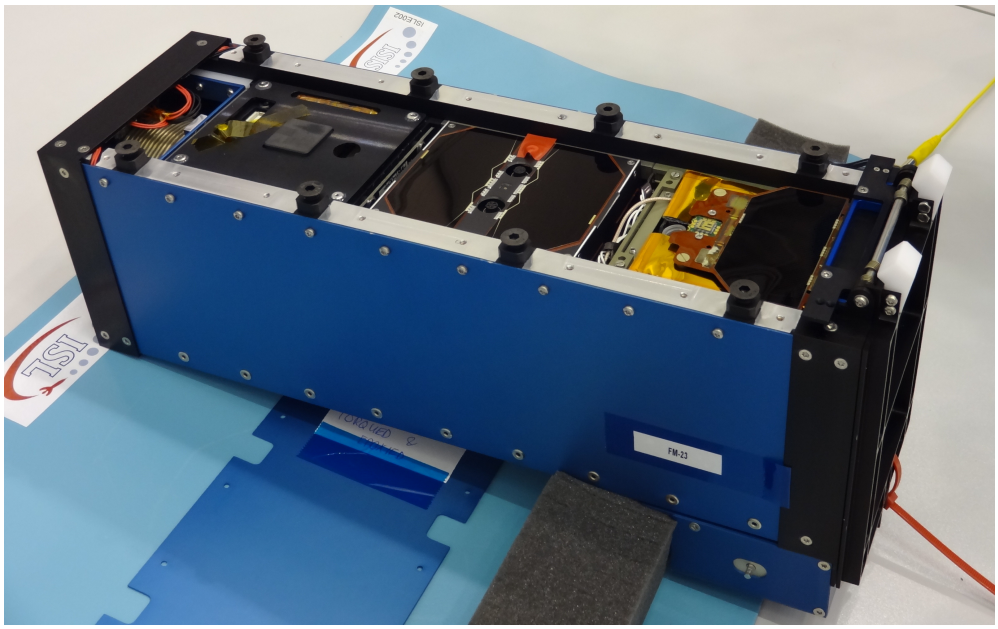


Figure 4: A 3U-CubeSat deployer holding First-MOVE (right), and two other 1U CubeSats.

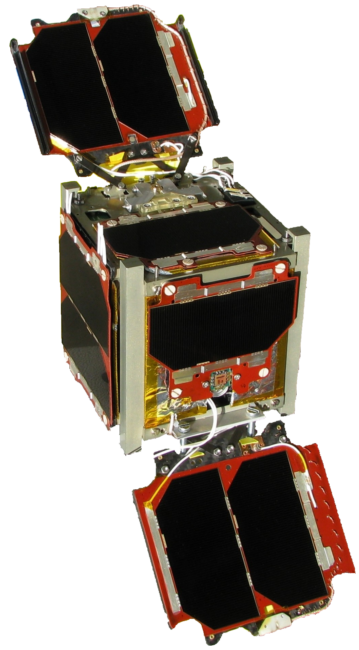


Figure 5: The 1U-CubeSat First-MOVE.

and mobile-market hardware since the early 2000s, the capabilities of nanosatellites have evolved considerably. At the time of writing, a diverse ecosystem of ready-to-use CubeSat components has developed. A variety of commercial companies of varying technical capabilities provide a customizable solutions of mixed quality, with ample launch opportunities into different orbits being available for 1–12U CubeSats.

The CubeSat First-MOVE (depicted in Figure 5) was one of these educational projects [36]. In 2013, I joined a research group developing this satellite at Technical University Munich, Germany, as a master student. Like many other first-generation educational CubeSats, First-MOVE was designed, constructed, and tested primarily by university students at the PhD, Master, and Bachelor levels. Planning of the First-MOVE mission began in 2006, a time when modern smartphones had just arrived in the consumer market, and construction in earnest began around 2010. It was launched into LEO on November 21st, 2013, and its malfunction, which is further described in Section 2.2, was the origin of the author's research on satellite fault tolerance.

2.1.5 Picosatellites and PocketQubes

PicoSats range in weight from between 0.1 to 1kg, and are today used for education or very brief proof-of-concepts. The PocketQube form factor and many 1U CubeSats fall into this category, and the electrical architecture of such PicoSats is often similar or even identical to that of light Nanosatellites. The main difference is lower mechanical complexity, and a further constrained power budget due to reduced solar cell surface (often ranging around or below 5W). In practice, this implies limitations especially for transceivers and payload, which are the main power consumers aboard modern miniaturized spacecraft.

2.1.6 Femtosatellites

FemtoSats are the smallest miniaturized satellite form factor and weigh less than 0.1kg. The concept of FemtoSats was theoretical until recently without allowing productive satellite designs that can take a productive role in a space mission. However, in the 2010s, first proof-of-concepts and practical applications have emerged [37]. FemtoSats usually consist of a single PCB using wireless energy harvesting or carrying a single solar cell on one side of the PCB, and electronics on the other [38]. With the emergence of more advanced energy harvesting and battery technologies in the future and an increasing level of semiconductor miniaturization, the basic character of FemtoSats could therefore change. Future FemtoSats will therefore find new niche use-cases, for which these lightest, cheapest, and expendable spacecraft will be optimal.

2.2 Early CubeSat Reliability and Motivation

Miniaturized satellite design is driven by the principle of designing a "good enough" spacecraft to do a job. Most Nanosatellites utilize COTS microcontrollers and application processor SoCs, FPGAs, and combinations thereof [39–41]. These components can offer one to two orders of magnitude more processing performance, are equipped with up to three orders of magnitude more memory, and an abundance of non-volatile storage capacity in comparison to classical space-proprietary components intended for larger satellites, while requiring less energy. Therefore, even a 5kg CubeSats can support a broad variety of commercial payloads and sophisticated scientific instruments, if these can be be fit into a smaller satellite chassis.

However, miniaturized satellites suffer from lower reliability, which discourages their use in long or critical missions, and for high-priority science. Most nanosatellites launched in the first two decades of the 21st Century (until the time of writing) still experience failure within the first months of their missions [39]. As depicted in Figure 6, even in late 2018 satellite malfunctions and early mission failures are widespread. The First-MOVE CubeSat is also representative in this regard, and we will use it as a case study to showcase the problems that still plaque this field.

First-MOVE: A Case Study

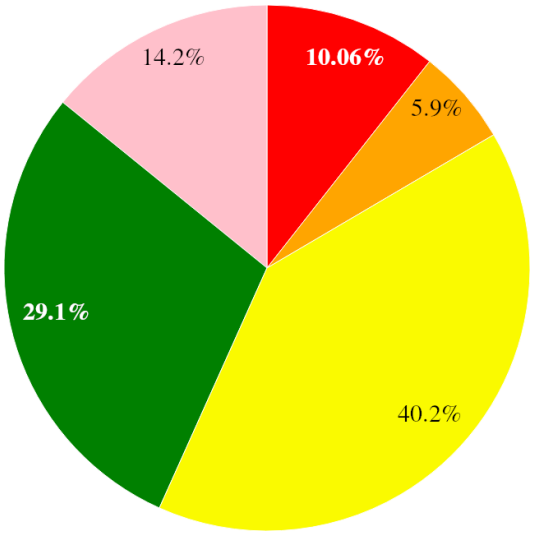
As a stereotypical late first-generation CubeSat, First-MOVE's design consisted of several microcontrollers. Its OBC was driven by a ARM926 based ATMEL microprocessor, utilized SDRAM, MRAM and NAND-flash memory, and is overall similar to a contemporary embedded device or smartphone. This fragile system architecture is representative for an entire generation of CubeSats built at that time.

At the time First-MOVE was designed little information was available on which components were expected to perform well in space, and which were likely to fail early on. During the actual construction phase, considerable information on these aspects became available continuously, and so its OBC was adjusted and retrofitted several times. E.g., the introduction MRAM was a retrofit to the original NAND-flash based design, as commercial MRAM was discovered to perform well aboard several earlier first-generation CubeSats. Further information on this First-MOVE's OBC is available in [Fuchs17].

First-MOVE successfully conducted its mission in LEO for two months after launch.

Towards the end of the mission, the OBC began to experience random reboots, which gradually increased over time. As of early 2014, the satellite could no longer be commandeered, and the mission was declared over. Both the funding organization (the german space agency DLR) and the CubeSat community considered the satellite performance and lifetime positive, and as the overall survival rates for CubeSat at that time were very low.

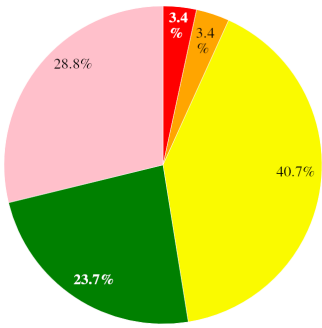
Subsequently, a team of three researchers, one of them being the author of this thesis, conducted a formal review of the First-MOVE project [Fuchs17]. This showed that if First-MOVE's system architecture had been fault-tolerant, the satellite could potentially have been recovered to a safe state. Otherwise, only minor organization issues related to the special setting of academic environments, which is a widespread prob-



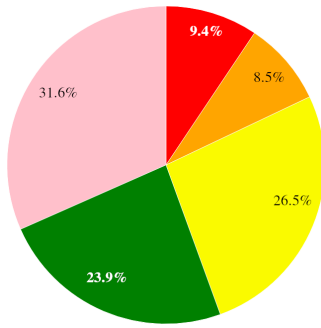
Full Mission Success
Partial Mission Success
Early Failure
Dead on Arrival
No Data/Unknown

Documented Launches	
Industry:	
Individual	59
Constellation	435
Professionals	234
University	223
Total	951

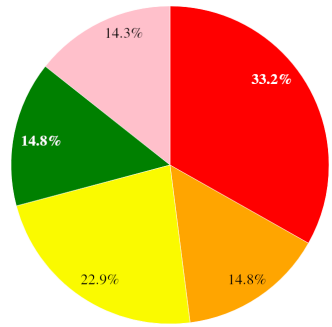
(a) CubeSat Mission Success



(b) Space Industry



(c) Professionals



(d) University & co.

Figure 6: CubeSat Mission success and failure for the time span 2000 to 2018. Bottom 3 charts show only data for individual CubeSats without satellites in constellations and swarms due to data quality reasons. It is reasonable to assume that developers of unsuccessful CubeSat missions also choose to not share information about the status of their satellites.

Image Credit: Charts produced through the CubeSat Database by Swartwout M. [42]. Military and other sensitive missions are often not publicly documented.

lem in academic satellite and instrumentation projects. A majority of first-generation Nanosatellite failures back then [43] could be attributed to design issues and manufacturing flaws due to developer inexperience (e.g., negative power budgets or dysfunctional communication channels) [39]. At the time of writing, failures caused by inexperience and design flaws have reduced drastically due to project professionalization and an increased staff of full-time developers in small-scale professional projects and academia.

2.3 Nanosatellites Today and Legitimization

Development on a second satellite, MOVE-II, began in late 2014 and the finished flight model is depicted in Figure 7. Since work on First-MOVE began in 2006, miniaturized satellite development has professionalized and fewer satellites fail due to practical design problems. Instead, the main source of failure aboard CubeSats today are environmental effects encountered in the space environment: radiation, thermal stress, and launch issues [2].

Mission result data shows that technological limitations are the main limiting factor regarding miniaturized satellite reliability at the end of 2018. Figure 6 shows that even experienced, traditional space industry actors who design such satellites "by the book" with quasi-infinite budgets struggle to reach 30% mission success. This lack of reliability and brief mission lifetimes curtails miniaturized satellite usage for critical and long-term space missions, as well as for high-priority science missions for solar system exploration, deep-space probes, and space observatories. During development

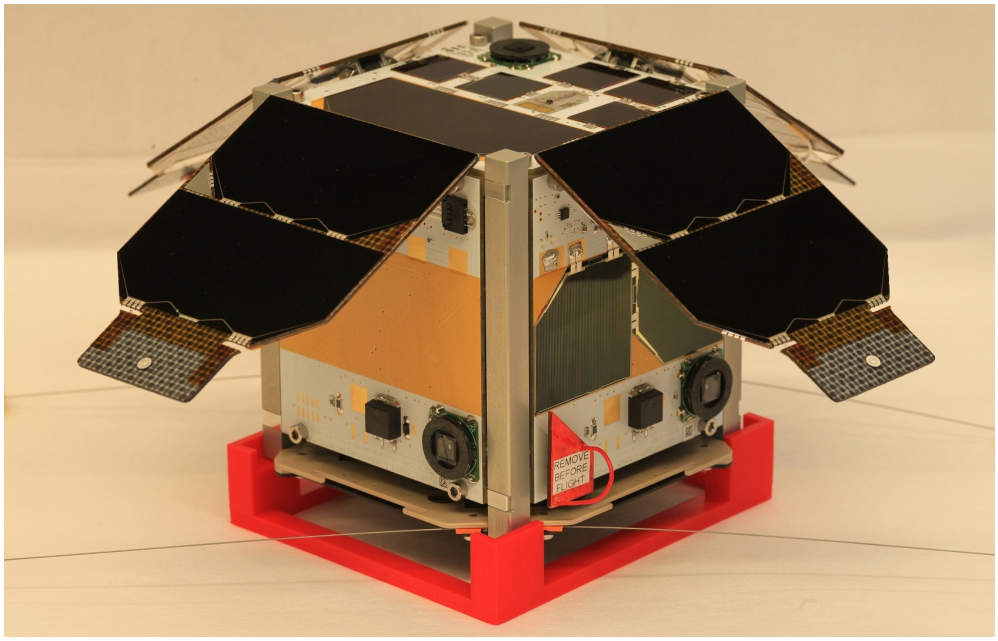


Figure 7: The MOVE-II CubeSat, which was part of the author's master thesis research and the design challenges faced during development initiated the research in this thesis.

of MOVE-II, it became clear to us as spacecraft designers that there were simply no fault-tolerant OBC solutions that could be used to achieve a more reliable satellite design within the constraints of a CubeSat.

Fault-tolerant computer design for spacecraft still relies upon radiation tolerant special purpose hardware These designs primarily rely upon proprietary fault-tolerant chip designs manufactured in technology nodes with a large feature size (radiation-hardening by design – RHBD) [44] and specialized manufacturing techniques and materials (radiation-hardening by manufacturing and process – RHBM/RHBP) [45]. Often, both of these techniques are combined and a RHBD chip design is manufactured in a RHBD process based with much more coarse feature size than commercial technology. Due to the lower energy efficiency and larger size of and greater distance between transistors, as well as less refined electrical properties, these components also require more energy, and offer less compute power compared to consumer hardware due to decreased clock frequencies and smaller memory sizes.

The use of traditional RHBM/RHBD components at the time of writing is limited to the civilian and military atmospheric aerospace industries, laboratory instrumentation for very large particle experiments run by well funded organizations (e.g., particle accelerators, radiation-testing sites) and traditional space-industry applications in long-term projects where cost considerations are not of primary concern. Especially in nanosatellites, the energy consumption, physical size, and cost of these components are prohibitive, making their use technically impossible and usually uneconomical. Therefore, nanosatellite computing has historically taken two paths: very simple onboard computers (OBCs) based on one single or few microcontrollers and very complex custom-tailored systems. This approach works to a certain extent, as there are a handful of COTS microcontrollers which are designed and manufactured in a way so that they unexpectedly turned out to be radiation hard (radiation-hard by serendipity – RHBS) [46].

At the time of writing, sophisticated fault tolerance capabilities are still absent in Nanosatellites. Instead CubeSat designers try to mitigate faults at the system level using custom mitigation circuitry [47], and thereby achieve "workarounds" to still somehow handle faults encountered in the space environment. The practical effect of this lack of viable fault tolerance techniques and the use of workarounds is reflected in the mission success statistics for miniaturized satellites depicted in Figure 6. However, a few CubeSats have also operated successfully in space for a decade or longer [48]. In practice, this shows that there is no hard technological limitation that would prevent the use of COTS technology in satellite missions with a much longer duration.

Many issues in other fields of spacecraft design can be overcome through engineering-based solutions. Such solutions work well, e.g., for addressing resonance issues, assuring a suitable thermal design and heat-distribution, and for deployable mechanical structures. Engineers therefore attempted to solve the lack of reliability of CubeSats similarly, by constructing custom fault tolerance computer design through component-level redundancy with commodity components. Practical flight results showed that such designs are fragile due to high complexity [39, 49], and tend to perform worse than much simpler designs without fault tolerance capabilities.

Today, nanosatellite designers have to forego fault tolerance in the hope of minimizing failure potential and thereby meeting satellite lifetime requirements for a given space missions by chance [50]. Designers are aware that such satellites may fail at any given point in time during a mission.

Figure 8: The launch of MOVE-II aboard SpaceX SSO-A: SmallSat Express on December 3rd, 2018 from Vandenberg Air Force Base, USA.

Image source: SpaceX SSO-A press material for public use.

MOVE-II was launched into LEO on December 3rd, 2018 with Space-X "SSO-A: SmallSat Express" (depicted in 8), where it operates successfully until at the time of writing this thesis. It utilizes only a few basic fault tolerance techniques that were available in commodity embedded components and COTS CubeSat subsystems. Its overall system architecture is still not fault-tolerant. Risk acceptance at this level is a viable approach only for educational, and uncritical, low-priority missions with brief duration. To construct future, more reliable miniaturized satellites, a robust, fault tolerance on-board computer architecture is needed. However, such an architecture do not exist yet, and with the research in this thesis I intend to change that.

2.4 Fault-Tolerant Computer Architecture

Fault tolerance in the most abstract sense, implies the capability of a system to overcome and gracefully handle failures. It is crucial for satellite computer design and a practical necessity to assure reliable operation of a satellite computer during space missions with an extended duration. As described in the previous section, the lack of such functionality within contemporary miniaturized satellites has become a major constraint to increase adoption of these spacecraft.

Fault-tolerant computer architecture, which is discussed briefly in this section, covers only a small part the entire field of fault tolerance and reliability engineering. Among others, systems can be designed to tolerate human error [51] and external attacks, which would require the discussion of aspects of psychology and human interface

design. In the remainder of this section, we discuss fault tolerance modes, measures, and testing from the perspective of computer architecture for spaceflight applications to provide the necessary background for this thesis. A more complete look on the different aspects and sub-fields of fault tolerance are available in literature, e.g., in [52].

Considering fault-tolerant computer architecture, the faults we must protect a system from depend on the application, the environment it operates in, as well as practical operating conditions (e.g., temperature and system load). Besides that, faults can occur due to technological wear and aging, and sometimes by chance. Many protective measures can be used to achieve fault tolerance for computer systems [53,54]. Often, the practical purpose for the application of these techniques is often not fault tolerance itself, but the need to increase scalability [55,56], manufacturing yield [57,58], higher clock frequencies and data throughput [59–61].

Different industries apply different fault tolerance techniques due to a variety of practical reasons, and today often maintain their own, proprietary implementations to tackle their domain-specific challenges. For proprietary fault tolerance implementations in different industrial applications, there is usually no immediate incentive to share and generalize such fault tolerance techniques by themselves, unless they can be patented, commercialized, and thereby protected [62]. This gap in turn is covered by scientists and researchers in industry and academia.

Today there is an entire field of science that tries to generalize application specific fault tolerance techniques, to produce new fault tolerance concepts through recombination. Unfortunately, this recombination is often done without considering the original application and its boundary conditions. As we show in Chapters 4 and 6, academic research and publications covering this topic are kept very abstract and do not consider a specific real-world application anymore. This works well for certain fields of science and even some fault tolerance topics[2]. However, for practical applications to system-architecture this is not the case, as generic solutions without proper boundary conditions and a realistic fault profile, can usually not be applied anymore to a real system. Today, academic fault tolerance research has produced a vast amount of publications and generated many theoretical concepts. But, only a handful of fault tolerance concepts envisioned by academic fault tolerance research have been implemented and tested in practice, and most have been ignored entirely by the industry. One could argue that this is the way science works, but knowingly publishing invalid and research without validation can also be seen as dishonest and only hinders publication of actually valuable research.

The path to validate such concepts is long, time-consuming, costly, and requires large amounts of engineering work [64–66]. The obtained validation results are often not considered publishable by academics, as they require a high degree of labor just to achieve one brief paper, while multiple theoretical journal publications could be produced in their stead. Industrial users are aware of such research [67], but are often skeptical. In the space industry, for example, concerns regarding validity, testability, verifyability and a perceived general lack of maturity of academic research has caused an entire industry to conservatively use very old technology [1].

When designing fault-tolerant systems, we must consider an application's operating environment, its fault profile, and system design constraints [68]. Generic fault tol-

[2]E.g.: erasure codes and performance overhead calculations to achieve quality of service under faults [63] can largely be discussed without a specific application in mind, as long as key parameters match.

erance concepts can serve as building blocks to design a comprehensive fault-tolerant architecture, assuming they are validated in a realistic manner.

2.4.1 Terminology and Fault Tolerance Objectives

Today, scientists and engineers use the terms ECC, EDAC, FDIR, and error correction almost interchangeably, while reliability, redundancy, fault tolerance, and robustness are surrounded by a shroud of marketing. In practice, error detection and correction (EDAC), fault-detection, isolation, and recovery (FDIR), redundancy, and failover all are distinct tools. They can be applied to achieve different kinds of fault tolerance, e.g., computational correctness, continuous non-stop operation, failover, and simple error correction.

Error detection and correction (EDAC) implementations usually utilize one or multiple erasure codes [69] to implement error correction coding (ECC), which allows errors in stored and transmitted data to be corrected. EDAC is efficient only for protecting the integrity of frequently access data, and may do so passively in the background without requiring a computer system to actively handle a fault in software. These limitations can be mitigated only in combination with other design measures such as error scrubbing and by generating error syndromes to notify the system about a fault [70].

FDIR instead assures that a fault-induced error is not just detected and corrected, but also that side-effects are isolated and resolved (e.g. discussed in [71] for space applications). In contrast, in case EDAC logic encounters errors when decoding data, it may inform the system about the result through an ECC syndrome and corrects data passing through. FDIR does not necessarily imply computation correctness, usually utilizes fault tolerance measures to achieve error detection and correction, but otherwise implies only that a fault is corrected and the system is restored to a working state.

Fail-over, in contrast, can be implemented as one-shot measure, e.g., with simple redundancy as discussed in [72], by falling from a primary to a secondary system instance and do not have to assess correctness, but only need be capable to detect faults. One of the most common applications for this approach is RAID1 with 2 memories or disks [72], but similar applications exist for avionics and network architecture in spaceflight and atmospheric aerospace applications [73].

2.4.2 Fault Detection and Correctness

To facilitate fault detection, we can exploit algorithmic measures as well as result comparison achieved through component replication (spatial redundancy) or repeat-execution (temporal redundancy). With algorithmic approaches detected errors can be reconstructed using parity data (informational redundancy) information, or by utilizing an alternative result generated through spatial or temporal redundancy. We refer to this type or error correction as forward error correction (FEC) [74]. Alternatively, backwards error correction (BEC) can be achieved with temporal redundancy and algorithmic measures, and implies message retransmission or re-execution of a failed operations [75].

Algorithmic Fault Detection and Informational Redundancy

The algorithmic approach exploits an inherent property of a system to detect faults. It can only be used if there is an inherent property in a system or protected data that can be used to judge the occurrence of a fault [76, 77]. Fault detection then does not imply the ability of the system to determine a correct result, but only the ability to asses if the protected data or system is faulty.

Algorithmic fault detection often exploits informational redundancy, but it may also use other inherent mathematical properties of data or logic-design properties of a system [77]. To a limited extent, algorithmic fault-detection can also be used to protect a program's data and control flow, e.g., by computing or modifying checksums for each executed instruction passing through a CPU's pipeline [78]. However, this requires a non-standard processor pipeline [79], a custom compiler toolchain [80], and therefore is feasible only for embedded software with a very specific structure.

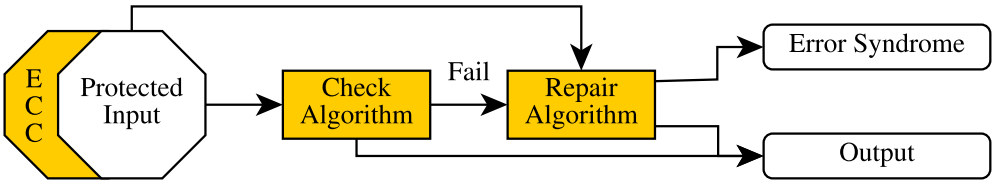


Figure 9: An example of algorithmic redundancy where extra algorithmic information is indicated separately as ECC. This extra information could also be an inherent property of the input data, instead of separate.

Spatial Redundancy

When utilizing spatial redundancy, we can realize fault-detection by comparing the output of multiple redundantly implemented system modules or equivalent but differently implemented variants of a subsystem run in parallel. Spatial redundancy can be implemented at all scales: for individual transistors and circuits, sets of logic, logic blocks, IP-cores, IP-core groups, ICs, components, to even an entire computer. At

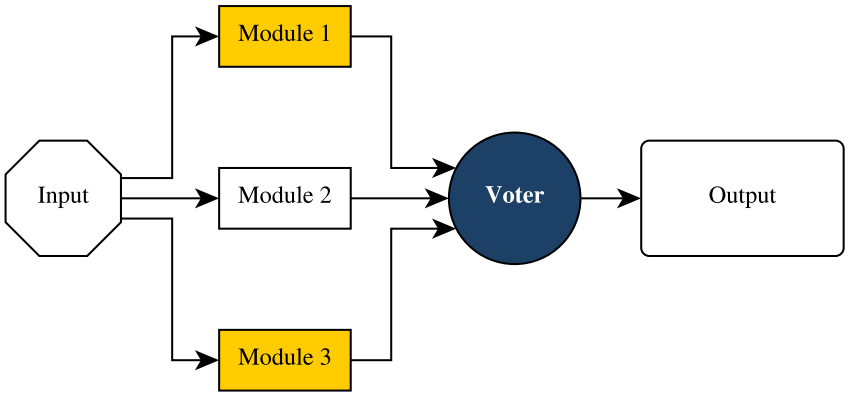


Figure 10: An example of spatial redundancy with 3 replicated modules in a TMR setup.

different scales spatial redundancy will offer different protective properties and different a level of scalability. A simple implementation of spatial redundancy requires replication of the actually protected modules as depicted in Figure 10.

We refer to systems consisting of N modules collectively as NMR systems. With 2 redundant modules, we can detect faults through supervision or in conjunction with a watchdog, to which we refer to as dual modular redundancy (DMR). We can determine correctness through a simple majority vote, for which at least 3 modules are needed (triple modular redundancy – TMR). These systems can be scaled up to realize $2k+1$ redundancy, as an odd-number of modules is needed to avoid a draw during voting. With more than 3 modules, more sophisticated voting concepts can be realized which then do no longer require a centralized and guaranteed-correct voting oracle [81] or allow a distributed majority decision [82]. An NMR systems can also be outfitted with spare modules to handle multiple subsequent transient faults or permanent faults.

Temporal Redundancy

Temporal redundancy implies re-execution of an operation multiple times in sequence, and example of which is depicted in Figure 11. Like in spatial redundancy, this is often done in $2k+1$ setups to assure error correction. This favors checkpoint implementation in software and the use of software diversity [83]. It is suitable for protecting applications where failed results can be discarded, individual operations can be repeated, or where an application as a whole can be restarted in a side-effect free manner. For most control systems and software running on general purpose computers, however, this is not the case.

The use of temporal redundancy introduces a degree non-determinism to an applications application, which can conflict with a requirements for real-time guarantees [84]. Thus, temporal redundancy concepts usually can only be applied to real-time systems unless the protected software implements a very specific structure [85]. Protection of applications at the scale of an operating systems, and programs with a complex program state or structure may incur a high performance overhead [86]. Due to the time-dependent nature of temporal redundancy, this form of redundancy is vulnerable to faults occurring in bursts or groups [87].

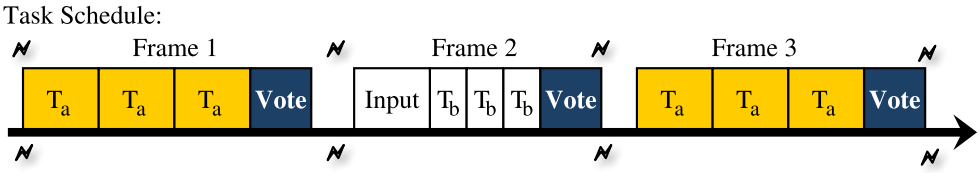


Figure 11: Task schedule of a temporal redundancy where every scheduled task (T_n) is executed 3 times with majority voting.

Fault Detection Granularity

The granularity and frequency for performing voting in spatial and temporal redundancy, as well as erasure coding parameters should be chosen based on the expected fault model and environmental conditions.

Most systems implementing spatial redundancy in use today implement instruction or clock-cycle bound lockstep for processor cores or larger system components [54]. This allows rapid error detection and correction without requiring the software or software to actively participate in fault handling [88]. Usually, the voter logic is combined with state-synchronization logic, to assure that all modules in a redundant set utilize the same input data. For more sophisticated computer designs, the level of complexity necessary to realize voting and state synchronization in hardware is non-trivial. Thus, such systems are limited to low clock frequencies than conventional designs [54].

As with temporal redundancy, we can also utilize software to realize lockstep functionality in spatial redundancy using checkpoints triggered through scheduling [89], or an external signal [90]. As we show in this thesis, lockstep-concepts implemented in software can enable more powerful dynamic, and runtime-configurable voting in conjunction with spatial redundancy to achieve FEC.

2.4.3 Effect Isolation

To achieve side-effect-freeness, the effect induced by a faults must be isolated, so they can not propagate within the rest of the system at large. However, the scope and way in which fault isolation can be implemented depends on the fault-detection measure, on the protected component, the high-level system architecture, as well as on the specific application scenario. For pure software-based measures utilizing temporal redundancy, this can be achieved by buffering results [91] and outputting a correct result after correctness has been assured.

Not all fault-tolerant systems require fault-isolation. The emission of incorrect data due to a fault can also be mitigated through a system architecture and instruction-set means [92], topological measures [93], or network-side [94]. Hence, a computer operating in such an environment does not have to be equipped with fault-isolation properties, as the overall system setup can already guarantee fault isolation.

2.4.4 Fault Recovery

In conjunction with or subsequent to effect isolation, the effects of a fault induced into a system should be resolved to prevent bit-rot and voter degradation due to transient faults [95]. It also reduces the need for over-provisioning redundant instances and parity data. For data storage, this can be achieved through parity in RAID- [72] or RAIF-like [96] systems, which can again be combined well with erasure coding [97]. It can make a system more robust especially if it has to operate for extended periods of time, or without maintenance.

Fault-recovery capabilities, thus, are not necessary for all applications, and may sometimes even be undesirable. For applications where maintenance can be performed frequently and the failure probability is low, simpler failover implementations can be of advantage since they are simpler, and therefore have a reduced failure potential. Examples include atmospheric aerospace applications for civilian use [98] or marine shipping [99]. This can allow a component to be implemented with lower complexity, thereby reducing overall failure potential, cost, and weight.

Depending on application requirements and if service interruption is acceptable, hot, cold, or warm [100] stand-by can be used to achieve failover [101]. Hot redundancy requires at least one redundant module executing in parallel to the primary module, to

allow the system to switch to failover without service interruption. Warm redundancy just implies a second module to be in standby mode, e.g., so it can rapidly take over operation by loading a correct application state. With cold redundancy, a redundant module is kept available but inactive, and has to be brought up when needed. This can allow energy saving and reduce wear in redundant module, but implies a time delay until regular operation can resume. In this thesis, we utilize warm standby when migrating applications from a permanently failed processor core to a new location.

Fault Recovery with Temporal Redundancy

In systems utilizing temporal redundancy to achieve backwards error correction, the generated incorrect application state of a failed operation has to be reverted. As temporal redundancy implementations usually require operations to be isolated or self-contained already, no further steps beyond discarding faulty data are necessary. By design, changes in the operating system state due to faults in temporal redundancy protected software will in practice be detected and subsequently not propagated.

Fault Recovery with Informational Redundancy

With informational redundancy, data containing a fault should be corrected and rewritten. In most memory-access based EDAC implementations, this step has to be performed independently from error correct, e.g., in software by an ECC syndrome or in hardware suitable error scrubber logic. In case of non-correctable erasure coding errors, or if backward error correction is used, data or a messages have to be retransmitted or rewritten. In memory-access based EDAC systems, non-correctable ECC errors can only be resolved with more redundancy and additional parity information, or through replacement and blacklisting.

Composite erasure coding systems combine multiple layers of erasure codes, to achieve the advantages of multiple different types of codes or parameter configurations [102]. These enable us to achieve overall stronger protection and mitigate weaknesses of individual erasure codes, e.g., symbol based block-codes are vulnerable to single bit-rot degrading their performance [103]. We describe the practical implementation of a composite erasure coding system combined with RAID-like features in Chapter 7.

Fault Recovery with Spatial Redundancy

In systems exploiting spatial redundancy, a fault may cause a failure of a redundant module, resulting in redundant system to become degraded.

To recover from transient faults, a failed module can be recovered using data from another module [104]. For voters replicating processor cores or larger system structures, this can be done with or without performing a reboot. For some cases, just copying the application or software state from a healthy module is insufficient, requiring a reboot to recover from a transient fault.

Conventional semiconductors affected by permanent faults can become dysfunctional, or may ceasing to function completely. To allow a system to tolerate additional, subsequent faults, additional spare modules are needed. We refer to this measure as over-provisioning. In practice, this can lead to large and very complex voter designs with high energy usage and large logic footprint [54]. With ASICs, the need for over-provisioning can only be alleviated through hardened manufacturing,

which is expensive [44]. This approach today is widely used in spaceflight applications to reduce the impact of transient and permanent faults. By design, such systems still become defunct once no further spare resources are available and a fault has occurred in system with only two intact modules.

Programmable logic devices such as FPGAs allow more refined permanent fault handling: permanent faults in the FPGA fabric can be mitigated by utilizing a configuration variant where no functionality-critical logic is placed in defective regions [105]. This can be used to restore a redundant module to a functional state. In practice, this approach can be exploited to allow a system to age gracefully by adapting to accumulating permanent faults over time, instead of failing spontaneously.

2.4.5 Fault Tolerance in the Real-World

Individual fault tolerance measures can be combined, allowing a vast amount of possible combinations. However, not all possible combinations are effective and efficient for protecting a system operating in a specific application environment and threat profile [106]. Certain combinations can even reduce reliability, or cause an increased failure potential [107]. However, if done right, fault tolerance measures deployed systematically in appropriate locations across a system [108], can allow for certain a defense-in-depth effect [109, 110].

Many fault-tolerant systems in use today are meant to isolate and recover from faults within the bounds of what their design constraints specified. However, this means that most fault-tolerant systems are not actually tolerant to faults, but that they are systems that can not fail so long as faults adhere to the specifications and "obey the rules set by the designer." In practical system design, these systems are then instead often treated not as robust and reliable, but as infallible systems that always work correctly and do not malfunction [111].

Validating Fault Tolerance Measures

To assess the effectiveness and strength of a fault tolerance architecture for a specific application, it must be validated in a realistic setup with a representative fault profile [112]. Such a profile is not just a statistical distribution over time, but should consider the impact of all relevant expected fault types (transient, intermittent, and permanent).

A variety of different test methods are available to analyze fault tolerance measures implemented at different scales and levels in hardware, in software, and both [52]. Historically, these methods included fault injection into hardware and software at different scales [65, 66], circuit simulation [64], mathematical correctness-proofs [113], statistical modeling [114], and even prototype experimentation for technology validation in a representative environment [115]. However, mathematical and logical proofs for modern processor based computer systems are non-trivial [116] and have been done only for individual algorithms, simple software, protocol state machines, and for simple circuits [113], but not for complex, OS-scale applications.

However, properly testing and validating software- and hardware-implemented fault tolerance measures is not trivial, requiring considerable time and development effort. Due to these challenges practical applications in industry tend to rely upon just a few widely used standard measures and combinations thereof, and disregard science.

Applied Fault Tolerance

Memory-access based EDAC through ECC is widely used in critical and always-on applications [117] due to its scalability, simplicity and low cost [118]. Due to technology scaling effects, technological reasons, and for the sake of yield enhancement, it has also become increasingly popular in consumer products [119]. All popular conventional high-speed interface and connector standards such as USB3 [120], SATA [121], Ethernet [122], and PCIexpress [123] rely upon powerful erasure coding systems to achieve high clock frequencies on serial channels [124]. Traditionally, ECC has been applied widely to protect non-volatile data storage solutions (e.g., nvRAM, memory cards) [125]. However, to increase yield in microfabrication, ECC has become common also to protect on-chip memories with a short data lifetime such as BlockRAM, caches, registers and the various scratchpad memories [126]. Designing systems for high-performance computing or critical applications without it would be impossible without erasure coding.

Today, most space-borne systems rely strongly upon spatial redundancy [54]. Most such systems rely upon hardware-voting, and only since the turn of the century has there been an increasing drive to realize FDIR functionality in software [127, 128] and using network topology and functionality [94]. This is an ongoing development, and this thesis should be read in context of this shift from traditional hardware to software and co-designed fault tolerance concepts [129]. Software-implemented fault tolerance concepts, however, have existed since the emergence of mainframes [130]. Even for space applications, they identified as promising already in the early days of microcomputers [131], but it was considered technically infeasible and inefficient until recently.

Technological Evolution and Heritage

The high stakes involved in operating critical systems in different fields, encourages the use of old and less efficient, but well understood architectures instead of more modern, and more powerful ones [54]. Hence, different industries progressed in developing fault tolerance concepts at different paces. While some innovated rapidly to achieve functional systems (e.g., the industrial and high-performance computing market, and the new space industry), others try to maintain a balance between old and new (e.g., automotive and medical embedded applications). Some chose to remain very conservative, preferring to re-use decades old concepts at extreme cost over using cheaper but more novel designs (e.g., the traditional space industry [54, 104, 132]).

Ultimately, however, all of industries are pressed hard to innovate, as technology progresses. An illustration of this need to innovate is the beginning adoption of the CAN bus standard [55], which was widely used by the automotive industry. The traditional space industry has just begun to adopt this standard few years ago and will benefit from its advantages over older standards considerably, though the interface and protocol are is currently being replaced in automotive industry by Flexray [56] and the use of high-speed computer network standards such as Ethernet [73].

However, the risky but fast-paced transfer of cutting edge technology from the embedded- and mobile market to spaceflight has resulted in the emergence of an entirely different, “new space industry”. Relevant industrial players try hard to utilize modern technology which can enable innovative space mission concepts that were completely unrealistic and often unimaginable just a few years ago. To do so, this industry

accepts an increased level of risk for failure. At the time of writing, the reduced cost of this engineering approach and the thereby produced designed spacecraft designs has succeeded and left a mark on the industry as a whole.