Run-Time Recovery Mechanism for Transient and Permanent Hardware Faults Based on Distributed, Self-organized Dynamic Partially Reconfigurable Systems

Victor Dumitriu, Student Member, IEEE, Lev Kirischian, Member, IEEE, and Valeri Kirischian,

Abstract—Field-Programmable Gate Arrays (FPGAs) are rapidly gaining popularity as implementation platforms for complex space-borne computing systems. However, such systems are exposed to cosmic radiation with levels orders of magnitude higher than terrestrial levels which can cause transient and even permanent hardware faults in on-board computing platforms. Because of this, development of effective fault mitigation methods and self-repair mechanisms has become a vital aspect for FPGA-based space-borne computing platforms. This work presents a novel method for transient and permanent fault mitigation and run-time fault recovery for commercial-grade FPGA devices with partially reconfigurable tile-based architectures. The proposed method ensures the same pre-determined recovery time for transient and permanent hardware faults through dynamic on-chip component relocation regardless of the fault type. The method makes use of fully distributed control, communication, self-synchronization and self-integration mechanisms embedded in each on-chip hardware component. Run-time collaboration between components provides relocation & fault mitigation procedures. The distributed nature of the above mechanisms excludes most central failure points which could cause non-restorable system faults. This method has been implemented, tested and verified on a Xilinx Kintex-7 FPGA platform. Results show that the proposed method is significantly more resource efficient when compared with Triple-Module Redundancy or central, software-based control mechanisms.

Index Terms—Fault tolerance, Field programmable gate arrays, Reconfigurable architectures, System-on-chip.

1 INTRODUCTION

As the capabilities of programmable logic devices, and in particular Field-Programmable Gate Arrays (FPGAs) have progressed, they have been adopted as implementation mediums for digital logic systems in a number of fields. One such field is that of space-borne high performance computing platforms. However, these types of systems are exposed to harsh radiation environments, and can suffer from faults, both transient and permanent in nature. Transient faults are usually caused by Single Event Effects (SEEs) affecting the FPGA configuration memory, and are mitigated via full or partial re-loading of said configuration memory [1]. Permanent faults can also occur, due to the Total Ionizing Dose (TID) the device is exposed to [2]; such faults must also be mitigated. To address this problem, radiation hardening technologies for CMOS circuits have been developed, and have been incorporated into space-grade, hardened FPGAs; such devices can withstand a TID of up to 1 Mrad (Si) [3]. However, such devices can be one order of magnitude more expensive than their terrestrial-grade counterparts; still more problematic is the fact that availability for such devices is often limited by production and trade regulations.

For the growing field of commercial space-borne computing platforms, a more cost-effective solution, based on commercially available FPGAs, is required. For systems deployed in Low-Earth Orbit (LEO) the TID is expected to fall in the range of 20-50 Krad (Si); the cost of such systems can be drastically reduced by avoiding space-grade FPGAs. For such systems, the radiation immunity is provided at the architectural level, as opposed to the technological level; such systems rely on Built-In-Self-Recovery (BISR) mechanisms to address both transient and permanent faults. One of the most promising means of implementing BISR in FPGA systems relies on the use of Dynamic Partial Reconfiguration (DPR). This feature allows for a portion of the FPGA configuration memory to be written while the rest of the device continues normal operation. This feature can allow a system to self-recover from radiation induced faults by changing the on-chip architecture of the system.

With this in mind, the purpose of the presented research was the development of a robust BISR mechanism for stream processing systems, based on DPR methods. A novel methodology has been developed, which relies on the self-assembly of a pipe-lined stream processor using the concept of Collaborative
Macro-Function Units (CMFUs). The CMFU is a self-contained macro-function processing element; multiple such elements can self-assemble into a larger processor based on the operating mode of the system. The support for this self-assembly process is provided by a Distributed Communication and Control Infrastructure (DCCI), which provides communication links between components, and allows CMFUs to negotiate their own connectivity. The primary mitigation mechanism is CMFU relocation, such that a faulty CMFU is moved to a different region of the device; this approach limits the mitigation time to a single known quantity (the relocation time) and allows for the mitigation of both transient and permanent faults via the same process. By adopting a distributed mechanism for both processing and control, including BISR, single points of failure are minimized; both BISR and BIST (built-in-self-test) functionality is distributed, increasing the redundancy and fault resilience of the system. The remainder of this paper will analyze related works in this field, present in more detail the concepts of the CMFU and DCCI, and conclude with an experimental system and associated results, based on the Xilinx Kintex 7 family of FPGAs.

2 BACKGROUND AND RELATED WORKS

Any fault-tolerant FPGA-based space-borne system must be able to mitigate both transient and permanent faults. Transient faults are caused by high-energy subatomic particles striking the FPGA, and are characterized as Single Event Effects (SEEs); they are divided into Single Event Upsets (SEUs) and Single Event Functional Interrupts (SEFIs). Such events can affect any part of the FPGA, including logic (look-up tables and DSP blocks), on-chip memory blocks, clock distribution and I/O blocks [4]. These effects can also affect the device configuration memory and associated configuration circuitry and ports and can be corrected by re-writing the configuration memory of the device [5], referred to as scrubbing the device.

Permanent faults generally have one of three causes: A) radiation effects such as Single Event Latch-up (SEL) and the Total Ionizing Dose (TID) the device is exposed to; B) hidden manufacturing defects; C) aging of the die. The reduction in core voltages has reduced the probability of Single Event Latch-ups (SEL) to being almost negligible. However, progress made in chip manufacturing methods has allowed for the reduction of feature sizes in FPGAs, which has led to an increased permanent fault probability due to manufacturing defects, die aging and TID exposure.

TID effects can be addressed at the technological level, system level and architecture level. At the technological level, radiation hardening techniques can be used in the manufacture of space-grade devices [3]. However, these devices are costly when compared with regular commercial devices in the same family, and are often face trade and/or technological export restrictions. At the system level, full or partial shielding can be used, based on materials with high atomic weight (Tantalum, Tungsten) [6]. Using this approach presents a number of problems, starting with the added weight of the shield. As well, thick shielding can lead to an increase in SEEs due to the interaction between cosmic rays and the shield [6]. Lighter shielding (such as aluminum) can also be used, to obtain a reduction in the TID affecting the device [7], although this approach does not reduce the rate of SEEs. Thus, shielding by itself offers only incomplete fault protection from radiation effects, and no protection for faults caused by die aging or manufacturing defects.

Radiation hardening at the architectural level relies on the use of Built-In-Self-Recovery (BISR) systems, which in turn assume the use of Built-In Self-Test (BIST) methods for fault detection and location. The topic of fault detection and BIST methods has received extensive attention, and a large body of work exists in this field (for example, see [8]). It will not be considered in detail in this paper. Self-recovery methods for FPGA-based systems are a much newer problem, and the research conducted in this field is more limited. BISR methods at the architectural level must address both transient and permanent faults. Transient fault mitigation using scrubbing is a relatively well-known process ( [5]); permanent fault mitigation is currently provided using module redundancy methods, although module relocation using Dynamic Partial Reconfiguration (DPR) is also possible.

To support scrubbing and relocation, a system must support run-time partial reconfiguration of selected system modules, and the possibility to allocate modules to different regions of the device (in the case of component relocation). A number of system architectures have been proposed which support one or both of these features. An architecture oriented towards automotive cabin control, but exemplifying the principles of many dynamic partially reconfigurable systems is presented in [9]; the presented system consists of system slots which can accept various various modules, as well as a shared communication infrastructure and a configuration management and control system based on a central processor.

A more general architecture is presented in the form of the Erlangen Slot Machine [10], [11]; the proposed architecture defines both on-chip as well as board-level elements, and consists of multiple FPGAs on multiple boards, as well as a discrete processor acting as the control center of the system (responsible for both FPGA configuration as well as communication infrastructure control). In a similar vein, a multi-board and multi-device architecture is presented in [12], where a satellite processing system is presented. As with other presented works, the system relies on central control elements, both at the system level (in the...
form of a central configuration management system) as well as at the on-chip level (local infrastructure and control elements).

Finally, a dynamic multi-processor system on-chip, the RAMPSoC, is presented in [13], [14]. Here the authors present an on-chip system consisting of multiple on-chip processing elements, consisting of instruction-based general processors (with or without local accelerators) as well as dedicated, custom processors. All processing elements are interconnected using the Star-Wheels network-on-chip [15]; the system relies on a central management and control processor which is responsible for the deployment of processors into the system, as well as the distribution of tasks to instruction-based processors.

Moving away from general architectures and towards explicit fault-tolerance oriented methods and architectures, a method for SEU mitigation is presented in [16], [17], where system modules can be scrubbed on an individual basis using dynamic partial reconfiguration; the scrubbing procedures are accomplished by a single on-chip controller. The authors of [18] propose a system which incorporates both scrubbing as well as architectural hardening via redundancy methods such as triple-module redundancy (TMR) or duplication; the level of hardening is varied by adding multiple copies of a module into the system. A central control sub-system, based on a MicroBlaze soft processor, is used to oversee all mitigation operations. A similar approach is presented in [19]; once again, the proposed method relies on a central control sub-system based on a sequential processor. The above examples can mitigate SEUs via scrubbing; however, permanent faults are not directly addressed, and require traditional solutions (such as TMR). In addition, all presented solutions rely on central control elements, which cannot be scrubbed without interrupting the whole system, to act as system overseers.

Given their popularity, a number of research efforts have been directed towards fault mitigation methods for sequential, instruction-based processors and multi-processor systems. In [20], a fault mitigation method is presented, based on the idea of relocation. The processor pipeline is divided into modules, any of which can be relocated; the configuration and control activities are delegated to a secondary, on-chip controller. In a similar vein, [21] proposes CPU fault mitigation based on dividing the pipeline into modules. Transient faults are addressed by scrubbing. Permanent faults rely on a reduced form of relocation; bit-stream variations are generated for the same device region, each of which has a different topology. The aim then is to find the correct bit-stream, such that faults are avoided. The authors of [22] present an alternative to CPU fault tolerance by relying on complete duplication of the CPU. Faults are detected by configuration memory read-back. Transient faults are mitigated via scrubbing, while permanent faults are mitigated using bit-stream variations with different topologies (a method the authors describe as tiling). Finally, [23] presents a multi-processor system consisting of four CPUs connected by a network-on-chip (NoC); all CPUs run self-tests periodically, and each CPU can scrub any of the other three. In the case of permanent faults, the software tasks mapped to the faulty CPU are migrated to the healthy part of the system; as such, this approach provides mitigation via degradation. All of the above approaches are aimed specifically at CPU architectures; in addition, all rely on static controllers to perform the required mitigation methods.

The authors of [24], [25] present an evolving hardware architecture; the proposed system consists of a two-dimensional systolic array. The distribution and connectivity of this array is controlled via DPR methods, and evolutionary algorithms are used to “train” it; this training can be used to deal with system faults, among other things. As is the case in previous examples, a CPU based sub-system is used to control both device configuration and the evolutionary algorithm. A more general approach is presented in [26], which deals with permanent device faults due to aging effects. The authors describe an algorithm for implementing a given design using multiple bit-stream variations which avoid different regions of a device. This is similar to the tiling technique described in [22]. The main drawback of this approach is that it can lead to a large number of configuration cycles and an over-long mitigation process.

Finally, a complete design flow for fault tolerant FPGA-based designs is presented in [27], [28]. The potential hardening methods considered are various types of redundancy (TMR, duplication) and scrubbing procedures. The authors assume the use of an external configuration element (residing off-chip) which is hardened separately. This work is expanded to address permanent faults via relocation in [29]; however, the approach is similar to the “tiling” method described above, and relies on down-loading a complete bit-stream to the system. Therefore, the mitigation process can be slow, in particular for large devices.

Two main characteristics can be attributed to the works presented above. The first and most prominent is that support for the mitigation of permanent faults is either not available, or limited to the use of “tiling” methods using full device bit-streams. Component relocation using partial bit-streams is supported only in [20], where it is limited to selected portions of the design, and in [24], [25], where the “relocation” process can be very slow, as it is based on an evolutionary mechanism. In contrast, the mechanism presented herein supports full component relocation for the majority of system modules, as well as scrubbing operations. The second shared characteristic is the reliance on a large set of singular elements for the support of the mitigation procedures (central processing
units, shared communication mediums with central control units) which form single points of failure. Hardening these potential failure points via module redundancy (such as TMR) can become very costly in terms of resources. The presented mechanism moves away from this approach in an attempt to minimize such points of failure, by relying on a distributed communication and control infrastructure. The proposed approach makes use of Collaborative Macro-Function Unit (CMFU), semi-autonomous processing elements which, when interacting with a Distributed Communication and Control Infrastructure (DCCI) can automatically self-organize into complete systems.

3 Mitigation of Hardware Faults by Dynamic Component Relocation

3.1 Fault Model

As discussed above, on-chip faults can come from one of three primary sources: high energy particle impact, Total Ionizing Dose (TID) accumulation and aging of the chip die (possibly aided by thermal cycling and the existence of manufacturing defects), leading to transient SEUs and MEUs as well as permanent faults. SEUs and MEUs are localized in the region where the particle impacted the device. TID-based faults can, nominally, affect large areas of the device, given a long enough time frame; however, if observed over short periods of time, they are expected to affect only a small region of the device. The same is assumed to apply to die aging; over short-to medium periods, the faults are expected to be localized to a limited region of the device. Thus, an expected temporal and spatial distribution of faults can be derived: A) while multiple faults are expected to occur over the lifetime of the device, no more than one fault is expected at a time; and B) each fault will be localized to one region, thus affecting only one module at a time.

3.2 Modern Programmable Logic Devices and Dynamic Partial Reconfiguration

As outlined in the introduction, modern programmable logic devices are being adopted as implementation platforms for digital systems; Field-Programmable Gate Arrays (FPGAs) are the largest types of such devices currently being used. These devices contain large amounts of logic blocks (which can perform basic logic operations), as well as dedicated blocks such as memories, multipliers and input-output circuits. These components can be configured to provide a variety of different functions in the system; this configuration is stored using local memories (either volatile or non-volatile), in the form of a configuration bit-stream. Dynamic Partial Reconfiguration (DPR) is an extension of the programming capability of modern FPGAs, whereby portions of the configuration memory can be programmed while the remainder of the device continues to operate normally. In this way, part of the on-chip architecture can change while the system continues to operate.

DPR capabilities allow for the time-multiplexing of multiple circuits on the same underlying configurable logic. More importantly, DPR systems allow spare device regions to be reserved and be used for fault mitigation. Since these regions are physically distinct, DPR relocation (where the component on-chip location is changed) permits the system to recover from permanent faults, by avoiding faulty regions. If only transient faults in the configuration memory are considered, these can be mitigated via read-back, scrubbing and error correction and detection codes; these mitigation methods are external to FPGA, and assume the underlying circuitry is intact. However, this approach cannot address permanent faults, which take the form of damaged and malfunctioning circuitry; these faults can affect any portion of the FPGA fabric, including configuration memory, routing elements, logic resources (DSP units, logic blocks) or I/O resources. This is why relocation via partial reconfiguration is being proposed. Currently, the largest manufacturer of FPGAs with DPR support is Xilinx, although the Altera corporation has recently began supporting partial reconfiguration in some device architectures. Due to the maturity of the DPR support in Xilinx devices and CAD tools, the remainder of the paper will refer specifically to Xilinx devices, although it should be noted that the majority of proposed methods are device agnostic and will work for any FPGA with tile-based DPR support.

The proposed mitigation approach introduces an added cost, as spare resources must be reserved inside the targeted FPGA. These added resources will result in increased costs for the complete system, as larger FPGAs must be used. However, the increase in cost due to added resources are much smaller (around one order of magnitude) than the cost associated with using radiation-hardened devices. As an example, Virtex 5 LX devices can vary in price from 250 to 10,000 USD per device, while the radiation-hardened Xilinx Virtex 5 XQR5VF130-1C costs between 100,000 and 133,000 USD per device [30].

4 Distributed Dynamic Partially Reconfigurable Slot-Based SoC Architecture

For a system to be as fault resilient as possible, its functionality should be distributed amongst multiple components, such that faults in single elements will have a limited impact on the complete system. Second, the amount of redundancy should be maximized, such that a single faulty component cannot compromise the complete system. Third, the system would need some mechanism for fault detection. Finally, the system should implement some form of fault...
mitigation, whereby lost functionality due to faults can be fully or partially restored. These concepts shall be pursued in the architecture presented here.

4.1 The MACROS Architecture

The Multimodal Adaptive Collaborative Reconfigurable self-Organized System (MACROS) method and architecture are introduced. A MACROS has the general form shown in Figure 1. It is composed of multiple Collaborative Macro-Function Units (CMFUs), a number of spare system slots (used for mitigation activities), a Distributed Communication and Control Interface (DCCI) and a Bit-Stream and Configuration Manager (BCM). The aim of this architecture is to minimize single points of failure by distributing system functionality amongst multiple components.

![Fig. 1. MACROS General Architecture](Image)

The presented mechanism and architecture are explicitly oriented towards stream processing systems (in particular those exhibiting a high data-rate) using dedicated, application-specific processing modules. The processed data is assumed to be an infinite sequence of data elements organized using a specific hierarchy (e.g. packets, frames); this data may or may not have a strict temporal arrangement, but is usually expected to incorporate some non-zero time gaps in the stream structure.

The CMFUs implement all application functionality, and are built to operate semi-autonomously. They automatically process incoming data without external control, by using indicators embedded in the data stream. In addition to stream data, CMFUs receive mode information defining the operations currently performed in the system, and are responsible for making requests regarding their own connectivity in the system. Finally, CMFUs are responsible for local fault detection, and for communicating the presence of faults to the rest of the system. This fault detection is accomplished using Built-in Self Test (BIST) units embedded into each CMFU. Using this approach, functional faults (be they transient and affecting only the configuration memory or permanent and affecting any portion of the device architecture) will be detected. Given that BIST methods have received significant attention in the past [8], they will not be considered further in this paper.

All CMFUs are implemented as Partially Reconfigurable Modules (PRMs) [31], and are housed in partially reconfigurable regions (PRRs), referred to as system slots. Spare slots are reserved inside the device, connected to the DCCI, which can house CMFUs should a fault occur somewhere else in the system. Currently, each system slot must be sized to contain enough resources to house any CMFU in the system; this is the simplest approach to take, but may not always be the most efficient in terms of area. More complex resource division schemes could include the use of multiple tiers of system slots (of different sizes). It must be noted that the partitioning of a potential system into modules is very much application-dependent, and can be considered a topic of research by itself; as such, this topic is considered beyond the scope of this paper and will not be addressed further.

The Distributed Communication and Control Infrastructure (DCCI) is responsible for all data communications in the system, component connection and disconnection, and the distribution of mode data; its internal structure is shown in Figures 2 and 3. The DCCI is distributed in nature, and consists of a crossbar, multiple Local Connection Control Units (LCCUs), a control data broadcast network and the mode broadcast bus. The system crossbar is responsible for providing data links between CMFUs while each LCCU is responsible for programming the crossbar connectivity for one port (Figure 2). The control data broadcast network is used to transmit control information between LCCUs (Figure 3); this information is used by each LCCU to determine what the local connectivity settings should be. The mode broadcast bus is used to broadcast operating mode data to all system CMFUs. All DCCI elements are implemented as PRMs, to permit partial scrubbing operations, and incorporate local BIST circuits.

The final element of the system is the Bit-stream and Configuration Manager (BCM), which is responsible for keeping track of the application bit-streams, making decision regarding which bit-stream should be loaded, and performing the actual configuration operation, by reading bit-streams from storage and loading them to the FPGA. In addition, the BCM is responsible for translating fault information into operating modes, as shown in Figure 3.
4.2 Fault Mitigation Methods

For the presented system, the primary method of fault mitigation is CMFU relocation. Relocation consists of changing the physical location of a faulty CMFU, by loading a new bit-stream to the device, targeting a different slot (location); separate bit-streams are needed for each slot in the system. This approach leads to an increase in non-volatile storage requirements for bit-streams; however, this method for relocation is directly supported by the existing CAD tools [31], and simplifies the relocation process (as only configuration is needed, without additional processing). Additionally, the cost of non-volatile memory storage (such as FLASH) is much smaller than the cost of the FPGAs being used; thus, the small increase in cost is considered cost-efficient given the obtained benefits.

CMFU relocation relies on the existence of spare partially reconfigurable regions (PRRs) which act as CMFU slots, as was shown in Figure 1. These slots will normally be used to support changes in the regular mode of operation [32]. When not in use, the slots are populated with blank bit-streams (i.e. no local logic), which ensures no activity in the region, thus minimizing dynamic power consumption.

If a fault is detected, either in a CMFU or a port of the DCCI, the corresponding CMFU can be relocated to a spare slot, and resume operation. As stated previously, the proposed system is aimed specifically at stream processors, and makes use of the specifics of stream structure; the temporal distribution of processing tasks follows that of the processed stream elements (frames, lines, packets). Bit-stream configuration (loading of a CMFU to a new slot) can be overlapped with processing activities; the actual integration of the relocated CMFU into the system (the actual change in system architecture) is scheduled to happen only between processing periods. The integration process occurs over a period of 25 clock cycles (in the experiments shown below) meaning 350 ns; this time period can be reduced further via control of the operating clock. The inter-arrival time between frames in a 1080p stream is 0.67 ms [33], which is more than adequate for integration activities. In the case of network applications such as TCP-IP over Ethernet, no explicit timing structure is provided; in such instances, some form of buffering is needed, or data can be lost during integration. In the case of 10 Gb/s Ethernet, 350 ns translates into only 3,500 bits worth of data that may be lost or should be buffered. In this way, the need for data buffering is either minimized or eliminated, without affecting system processing activities. Data that was being processed at the time of the fault is assumed to be lost; this approach is oriented towards streaming systems where temporary errors in the data stream can be permitted (e.g. video applications, most networking applications). If no data errors can be permitted, then alternative and more costly methods, like TMR, can be used to ensure errors are detected and corrected.

Once the system resumes working correctly (the CMFU has been relocated and re-integrated into the system), scrubbing and testing procedures can be used to determine if the fault was transient or permanent. In the case of a permanent fault, the associated slot is marked as in-active, and is no longer used. In the case of a transient fault, the slot is marked as a new spare. Additionally, spare slots can also be used to house redundant versions of certain CMFUs in situations where the relocation-based mitigation time is considered too large; in essence, redundancy is applied at the module level as opposed to the complete system.

By resorting to relocation for both types of faults, an upper limit is placed on the mitigation time. Furthermore, direct relocation is less time consuming than
scouring; a complete scouring operation can consist of multiple iterations of partial bit-stream loading, followed by test procedures. Since fault mitigation leads to a change in the architecture of the system, it can be thought of as a mode change; thus the fault mitigation process uses the same mechanisms that implement regular mode changes.

5 COLLABORATIVE MACRO-FUNCTION UNIT ORGANIZATION

CMFUs are autonomous macro-function processors which are responsible for implementing all application functionality. They perform processing operations, self-test procedures, and determine their own connectivity based on the application operating mode, without external intervention.

5.1 CMFU Architecture

At its core, a CMFU is based on a macro-function specific IP core; however, to support DPR operation, as well as fault tolerant behavior, this IP core is augmented, as shown in Figure 4. The complete CMFU is composed of the IP Core, an IP Core specific Built-In Self-Test (BIST) Unit, and a Co-Op Unit (COU). The BIST unit is based on well-established self-test methods [8], and will not be elaborated further here. The COU acts as the behavioral interface between the IP Core and the rest of the system; while the IP Core is macro-function oriented, the COU incorporates both macro-function information as well as application information.

![Fig. 4. CMFU General Structure](image)

The COU has four primary tasks to perform. First of all, it may have to schedule IP Core operations as stream data arrives; this is needed if the core is not built to operate repeatedly on stream structures. This scheduling is based on data stream control information (packet headers, flags, etc.); related to this aspect, the COU may have to generate output control information for use by down-stream system components. Secondly, the COU must trigger the local BIST circuitry, determine if a fault exists, and indicate this situation to the rest of the system using the Fault Reporting portion of the CMFU interface. BIST operations are undertaken between active work periods (periods when the IP core is processing application data) or when the component is disconnected from the system. This approach cannot automatically detect and correct errors as they occur, as is the case in TMR solutions; it can only ensure that future errors do not occur. As mentioned in Section 4.2, data affected by a fault occurring during a work period is assumed lost.

Third, the COU allows the CMFU to determine its connectivity by making requests to the DCCI (which incorporates no information regarding operating mode and connectivity). To permit this functionality, the COU incorporates a look-up table which lists a crossbar connection for every mode of operation, including modes associated with fault mitigation activities. Finally, the COU negotiates with the DCCI as to when a CMFU can be disconnected from the system; disconnection consists of driving a constant (usually 0) instead of active data to the CMFU inputs. Disconnection should occur when the local IP Core is not actively processing information, which is referred to as the safe state between work periods. The COU uses a signal to indicate when the CMFU is in its safe state and allows external elements of the DCCI to stall the local IP Core between work periods; this is done to accommodate more complex architectural changes.

The COU consists of a central FSM, the mode look-up table and additional counters and decoders, needed to track and generate control information for the data stream and control the flow of control information to the IP Core. The COU FSM operates in a cyclical fashion, as shown in Figure 5. Each cycle is triggered by control information in the incoming data stream; if the CMFU is stalled, this information is ignored. As part of each cycle, the COU triggers the IP Core (if necessary), generates output control information (again, if necessary), waits for the end of the work period, determines the connectivity based on the mode data, and triggers local BIST activity. For this process to work correctly, run-time BIST procedures must be capable of finishing a test sequence in the time available between active processing periods.

![Fig. 5. COU FSM Behavior](image)

The interface between the CMFU and the rest of the system consists of a stream and control component. The stream interface is application-dependent, and accommodates the data and control information of the application. The control interface is composed of safe and stall flags, a status bus, an error indicator (coded to reduce fault sensitivity), and the CMFU ID bus, used to identify the local CMFU and its up-stream connectivity.
5.2 Fault Effects and Mitigation

In keeping with the established fault model, faults are localized on a per-module basis. In the case of a CMFU, that means faults in the IP Core, BIST circuit or COU. IP Core faults are detected via the BIST process; the BIST circuitry and the COU can also use self-test procedures, or rely on duplication for fault detection. Given that the BIST and COU circuits are expected to be much smaller than a macro-function processor, the redundancy cost will not add much overhead to the CMFU.

In all fault cases, the mitigation process is always the same: relocation to a spare system slot, possibly followed by scrubbing and test procedures of the slot under test. If the original fault was found to be transient, then the original slot can become a new spare slot; otherwise, it is marked as permanently faulty. Since the BIST, COU and IP Core are all relocated together, the same relocation process will address faults in any of these sub-systems. As already discussed in [34], the relocation process can be performed without affecting the timing behavior of the system, which means that the relocation is transparent to CMFUs. The only exception are CMFUs which act as I/O interfaces, and are connected to I/O logic; this logic is locked down to a specific location on the device. Such components must rely on traditional hardening methods (for example TMR) to ensure correct, fault resilient operation.

6 DISTRIBUTED CONTROL AND COMMUNICATION INFRASTRUCTURE

6.1 DCCI Architecture

As established in Section 4, the Distributed Control and Communication Infrastructure (DCCI) is composed of four elements: a crossbar, a collection of Local Connection Control Units (LCCUs), a control data broadcast network, and a mode data broadcast bus. The crossbar is used to establish communication links between CMFUs; the collection of system LCCUs, the control data broadcast network and mode data broadcast bus are responsible for controlling the crossbar connectivity. The crossbar was selected as the underlying architecture for the communication system, as it allows for large aggregated data throughput, and is non-blocking. More detailed information on crossbar architectures and implementation can be found in [34]. Networks-on-Chip (NoCs) were eschewed as means of on-chip communication due to their blocking nature and added routing overhead.

The process of assembling a full, application-specific processor from CMFUs is shown in Figure 6. The process begins with a CMFU making a connectivity request based on the mode information it is receiving (Step 1); this request specifies how the local crossbar port should be programmed. The request is received by the LCCU, which must then negotiate when the connectivity change can take place (Step 2). This negotiation takes place between all system LCCUs, and is accomplished through the control data broadcast network. Once the local LCCU determines that it is safe to do so, it will make the requisite connectivity change (Step 3).

Fig. 6. Connectivity Change Process

To be able to correctly disconnect and/or reconnect a CMFU to the rest of the system, the CMFU must be in its safe state (as defined above). All CMFUs downstream from it must also be in their safe state, so that the up-stream connectivity change does not occur in the middle of any work periods. For large pipe-lined systems, the safe states of all CMFUs overlap for a period of time, based on the latency of the pipe-line and the nature of the stream data being processed. Connectivity changes can be applied safely at this time. The control data broadcast network is used to determine when all CMFUs have finished processing and find themselves in their safe state.

Each LCCU uses the control data broadcast network to send out the ID and connectivity information of their local CMFU to the rest of the system. The structure of the control data network interface is shown in Figure 7 below. The local CMFU ID describes the local CMFU; the disconnection request signal is used to request changes to the architecture. Finally, the error status is used to communicate fault information (this information is coded using error detecting codes) The fault information is used by the bit-stream and configuration manager (BCM) to determine the mitigation activities needed.

The LCCU proper consists of a control FSM and storage registers for the local ID and crossbar control. The FSM behavior is shown in Figure 8. Starting from the idle state, if a global or local connectivity change occurs, the FSM enters a wait state, where it remains until all LCCUs indicate they are in their safe state. Depending on the status coming form the CMFU one of three things will happen: no changes are made (the CMFU connectivity remains the same), the CMFU is disconnected, or the local crossbar port is reprogrammed to reflect a new connectivity requirement.
6.2 Fault Effects and Mitigation

Faults can occur in one of three places in the DCCI: the crossbar, LCCU (including the control data broadcast network interface) and the mode broadcast bus. As with CMFUs, it is assumed that BIST circuitry is included locally with each crossbar port; this circuitry detects faults in the local crossbar circuitry, LCCU and control data network interface. Both CMFU and DCCI faults are reported to the BCM using the control data broadcast network.

By relocating the CMFU associated with a port, the faulty portion of the DCCI is avoided, as all DCCI structures are implemented on a per port (and therefore per slot) basis. By relocating to a separate slot, a new, fault-free port of the DCCI is used. Once the relocation takes place, scrubbing and test procedures can be applied to the faulty elements of the DCCI; if the fault was found to be transient, the vacated slot becomes a new spare slot.

7 BIT-STREAM AND CONFIGURATION MANAGEMENT

The Bit-stream and Configuration Manager (BCM) is responsible for reading partial bit-streams from non-volatile storage and loading them to the FPGA in response to detected errors, as well as generating mode information which will then be broadcast on the mode broadcast bus. The general structure of the BCM is shown in Figure 9, and consists of an Architecture Decoder, Allocation Engine and Configuration Engine. The Architecture Decoder is responsible for determining how the architecture should change for a given fault based on received fault information. The Allocation Engine is responsible for determining which partial bit-streams are present in the system, which bit-streams need to be loaded and where, and which system slots are faulty or require further testing; it also generates the operating mode for the rest of the system. The Configuration Engine performs the actual configuration operations; it receives bit-stream addresses from the allocation engine and proceeds to read said bit-streams from storage and program them to the FPGA.

The BCM, by its very nature, is a central point of failure, since it is tied to the configuration circuitry of the device, which is not distributed in nature. It can be implemented either as an on-chip system, or as a separate device. If implemented on-chip, it performs configuration activities via the Internal Configuration Access Port (ICAP), and if external, it will make use of the SelectMAP32 parallel interface [35]. If the BCM is external, it can be hardened at multiple levels (architecture, technological). The BCM itself is not a complex circuit, as a lot of the system control is off-loaded to the DCCI and CMFUs, which means that it can be implemented using small devices (compared with the main processor). Utilizing a small radiation-hardened device for this system is a viable option.

If the BCM is integrated on-chip, two main points must be considered. First, an off-chip configuration solution is still necessary for the initial configuration of the device; if the BCM is external, it can handle this initial step. Second, an on-chip BCM cannot mitigate any of its own faults without outside “help”. The BCM can be implemented as CMFU and relocated to target either of the ICAP ports; however, this can only be accomplished through configuration by a second agent. Modern Xilinx devices contain two ICAP interfaces, one in each vertical half of the device; faults in the primary BCM can be mitigated by a secondary BCM using the second ICAP interface. However, the second ICAP interface is activated by editing configuration
register CTL0 [35]; this will not be possible from inside the device if the BCM is faulty, and a secondary external circuit will be needed.

8 Framework Test and Verification - Video Processor

A number of experimental systems were constructed and tested, to analyze the proposed methodology; one such system, a video processor, is presented and analyzed in the following section. Three primary topics of inquiry are being considered: basic functionality (whether the proposed method actually works), timing of the proposed mitigation activities and resource cost overhead associated with the proposed method.

8.1 Video Processing System

The implemented video application is shown in Figure 10 - A; it mimics the video pre-processing stages of a feature extraction system, by performing median filtering and Sobel edge detection operations on a video stream. For testing purposes, a 720p60 video pattern is used. The architecture of the system is shown in Figure 10 - B; the system contains 7 system slots, 3 of which are reserved for static components and four of which are used for relocatable components.

The median filter and Sobel edge detector are implemented as relocatable components and can be loaded into any of these four slots. The pattern generator, HDMI display unit and color space converter are implemented as static components, although they all support scrubbing operations. The DCCI consists of individual LCCUs with control data network interfaces, the system crossbar and mode broadcast bus; each LCCU and network interface can be scrubbed individually, and so can the crossbar and mode broadcast bus, if needed. The test system was implemented on a Xilinx XC7K325T device, using the KC705 evaluation platform [36]. The BCM portion of the system was emulated, using a host computer; configuration activities were accomplished using the JTAG interface for the purpose of this experiment.

8.2 Fault Mitigation Analysis

CMFU relocation is the primary method for fault mitigation being proposed. Once a fault is detected, the CMFU relocation process consists of the following: i) download new bit-stream for the CMFU, targeting one of the spare system slots, ii) change operating mode to one which includes the new CMFU, and excludes the old (faulty) one and iii) wait for the DCCI to transition to a new mode. From the point of view of the application proper, $T_{mtg}$ is all the time needed to broadcast the new mode. Beyond this point, the system will undertake scrubbing and testing operations, with the aim of determining if the fault in question is permanent or transient. One or more rounds of scrubbing may be needed to determine if the detected fault was permanent or not.

The relocation process was tested for the implemented system, and the value for $T_{mtg}$ was found to be less than 1.11 ms. All three timing parameters are listed in Table 1 below, along with an explanation of how each was obtained. One video frame is processed every 16.67 ms (60 Hz); depending on the frame number, the mitigation process can be accomplished in either one or two video frames. If the fault is detected at the end of frame 1, the mitigation process extends into frame 2, which means that the system will be working correctly by frame 3.

By adopting relocation as the main mitigation approach, the mitigation time is limited to the time needed for the relocation activity. Additional scrubbing and testing procedures are overlapped with regular system operation. Previous mitigation methods for permanent faults fall into one of two main categories. The first is actual relocation of faulty components (as shown in [20] for example); however, approaches presented so far are limited to specific subsets of the system. The second approach relies on full reconfiguration of the device to accomplish relocation [22], [29], which is more time consuming. The proposed approach provides relocation support for all system components (excluding I/Os) which leads to reduced mitigation time, since partial bit-streams are used. Figure 11 shows the speed-up obtained by using partial versus full bit-streams for permanent fault mitigation in various Kintex-7 family FPGAs.
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Time</th>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T_{con f}$</td>
<td>1.1 ms</td>
<td>366 KB configured at a 3.2 Gb/s rate</td>
</tr>
<tr>
<td>$T_{pa}$</td>
<td>337 ns</td>
<td>25 clock cycles at 74.25 MHz</td>
</tr>
<tr>
<td>$T_{mb}$</td>
<td>13.5 ns</td>
<td>1 clock cycle at 74.25 MHz</td>
</tr>
</tbody>
</table>

### 8.3 Resource Costs and Overhead

The core application functionality is provided by various IP Cores, as previously established. The proposed method adds overhead to these cores, in the form of the COU and DCCI. In the system presented above, each of the 4 system slots contains 300 slices; this will be used as a yardstick of sorts to evaluate the MACROS overhead. Table 2 lists the resource costs for the COU, LCCU (including local elements of the mode broadcast bus and control data network interface), crossbar and one system slot. The COU and LCCU associated with a slot add up to less than a 16.67% of a system slot; the crossbar amounts to more than half. However, the crossbar is an unavoidable requirement of the system, given the data rates needed, and the changing nature of the system components.

#### TABLE 2
Resource Utilization

<table>
<thead>
<tr>
<th>Component</th>
<th>Slices</th>
</tr>
</thead>
<tbody>
<tr>
<td>Crossbar</td>
<td>240</td>
</tr>
<tr>
<td>COU</td>
<td>14</td>
</tr>
<tr>
<td>LCCU plus overhead</td>
<td>36</td>
</tr>
<tr>
<td>System Slot</td>
<td>300</td>
</tr>
</tbody>
</table>

As a more general indicator of the system overhead, Figure 12 plots the size of the crossbar as the number of system slots increases, up to 32. As expected, the size of the crossbar increases non-linearly with the number of slots. A potential solution is to separate the crossbar into a multi-layer Clos switch [37]. By using this approach, the size of a large crossbar can be reduced to some extent; the dotted line in Figure 12 shows a three-layer Clos architecture. The reduction in size is limited, because the system must still be non-blocking. Additionally, different sized crossbars (meaning number of ports) are implemented less or more efficiently depending on their bit-width and the target device micro-architecture. This varying degree of efficiency is shown in Figure 12, where the 16-port crossbar is implemented more efficiently than the 12-port version; this feature can be combined with multi-layer architecture implementations to further reduce resource cost in large crossbars (in excess of 32 ports).

![Fig. 11. Fault Mitigation Speed-Up Due to Proposed Method](image)

![Fig. 12. Crossbar Resource Cost Versus Number of System Slots](image)

Fig. 11. Fault Mitigation Speed-Up Due to Proposed Method

![Fig. 12. Crossbar Resource Cost Versus Number of System Slots](image)

Networks-on-Chip (NoCs) have been proposed as a potential alternative to the use of crossbars for high data-rate communication, under the assumption that they are more scalable. To explore this concept, 12, 16 and 32-port crossbars are compared with Star-Wheels NoC topologies [15]. Figure 13 shows the estimated cost of NoCs for 12, 16 and 32 slot systems (the estimate was obtained using data reported in [15]); since the NoC resource cost was reported in terms of Virtex 5 slices, the crossbar was implemented using the same device family. The crossbar requires far fewer resources than the NoC implementation, although the rate of growth it exhibits is larger. This is not surprising, given that the Star-Wheels NoC implements more complex behavior oriented more towards a general computing model, which may be unnecessary in pure streaming applications. The results also suggest that the crossbar can remain cost-efficient for even larger systems, while guaranteeing the same level of non-blocking performance.

Figure 14 shows the cost of COUs and LCCUs as the number of system ports increases; in this case, the increase is almost linear with the number of slots. Figure 15 shows the complete resource overhead (as a percentage) for 7, 12, 16, 24 and 32 slots. In each case, the same slot size (300 slices) is assumed; the figure shows the overhead for systems with both 1 and 2 spare slots. The overhead is based on the cost of the LCCU, COU, control data broadcast network and mode broadcast bus. The crossbar cost is not included in this overhead, as it is considered a non-avoidable system cost; system modules must have some means of communication, and this communication medium...
must be programmable, to reflect changes in system architecture. The comparison presented above demonstrated that the crossbar can be less costly than a network-on-chip. In effect, whatever the mitigation process, some type of communication infrastructure must be present in the system, hence why it is considered non-avoidable.

The worst case overhead occurs in small systems, and never exceeds 70%; as the size of the system increases, the overhead drops to less than 25%. In duplex solutions, a 100% overhead is expected, and in TMR solutions, a 200% overhead is expected. Each entry in Figure 15 lists the reduction factor for the proposed method compared to a TMR solution. Of course, TMR allows for error correction and detection of every data value generated (at least as long as only one module undergoes faults); however, in situations where temporary errors in the data stream are permitted, the proposed method offers better resource utilization and allows a designer to trade resource overhead for improved fault resilience by adding more spare slots. The presented data excludes the cost of BIST circuits, which will add to the system overhead. However, BIST circuits can be designed for various capabilities (fault detection, localization, correction); for the proposed mechanism, only fault detection is needed. The overhead due to BIST circuits is not expected to exceed 100% in the worst case situation; even with this overhead, in large systems the proposed method is more cost efficient than a TMR solution.

Finally, the proposed distributed control system is compared with a central solution based on a MicroBlaze processor, which has proven popular in past research (e.g. [22], [23]). The system is assumed to consist of a MicroBlaze processor, a crossbar and dedicated application-specific modules for all processing operations. Thus, the MicroBlaze processor is used only for programming the crossbar during the mitigation process; it performs no other processing tasks here. For a more in-depth comparison of the performance of such a processor versus application-specific solutions, please see [38]. Multiple MicroBlaze test systems were implemented to obtain their resource cost and attainable performance. Figure 16 shows the resource cost for the proposed distributed approach (LCCU, COU, and control data network interface), as well as the resource cost for single and duplex MicroBlaze systems. If a single MicroBlaze is used, the proposed method becomes more costly when more than 22 slots are used. If the MicroBlaze is duplicated, to permit fault detection, the proposed mechanism is less costly for up to 32 system slots.

At the same time, the mitigation time increases dramatically when the MicroBlaze is used as the control mechanism, due to the increased time cost of the connectivity change process ($T_{pa} + T_{mb}$). Experiments were conducted to determine how fast a MicroBlaze controller could program the ports of a system crossbar; using the highest compiler optimization level and excluding all index computations, 39 clock cycles (at 100 MHz) were needed per crossbar port. Figure 17 shows the time needed to change all connectivity in a crossbar, for various crossbar sizes, for both the central and distributed approaches. The distributed approach remains constant at 352 ns, regardless of the number of ports; on the other hand, the MicroBlaze-based solution starts at 2.7 µs and can reach more than 10 µs. The proposed control method offers a minimum speed-up of 8 times for the same or less resource cost.
9 CONCLUSION

FPGAs are emerging as a viable implementation platform for high performance space-borne platforms. Such systems, when deployed to various Earth orbits or interplanetary missions, are exposed to larger radiation dosages than those encountered in terrestrial applications, leading to the occurrence of transient and permanent system faults. Thus, FPGA-based space-borne computing systems must have some protection mechanism to ensure proper performance in radiation-intensive environments. This protection can be static, dynamic or a combination of both. Static protection assumes shielding or utilization of radiation-hardened FPGA devices; however, these solutions can lead to dramatic increases in the complexity and cost of the system. Dynamic protection entails real-time recovery of the system at its on-chip architectural level. Traditional TMR solutions entail over 200% overhead in hardware resources and power consumption; furthermore, fault accumulation in such systems reduces their fault tolerance capabilities. Most existing DPR mitigation methods address only transient faults via scrubbing of the configuration memory and offer limited support for permanent system faults.

![Image](66x100 to 246x233)

**Fig. 16. Cost Comparison Between Central (CPU) and Distributed Approach**

![Image](66x285 to 246x418)

**Fig. 17. Connectivity Change Execution Time for Central and Distributed Mechanisms**

With this in mind, the preceding paper has presented a fault mitigation methodology which allows run-time self-recovery of FPGA-based systems through the use of Dynamic Partial Reconfiguration (DPR); the method permits recovery from both transient and permanent faults, via component relocation. A distributed architecture is used, composed of Collaborative Macro-function Units (autonomous application-specific processing circuits), a Distributed Communication and Control Infrastructure and a Bit-Stream and Configuration Manager, which together collaborate to assemble application-specific processors. By CMFU relocation, the mitigation time is always limited to the relocation time. This approach is significantly faster than complete device reconfiguration and entails a temporal cost which ranges from hundreds of microseconds to units of milliseconds. The proposed method has been implemented and tested using a Xilinx Kintex-7 FPGA-based platform. It was found that the hardware overhead of the method amounts to 25% per component (CMFU); in contrast, triple module redundancy (TMR) methods add more than 200% overhead. The distributed control system was found to perform more efficiently (up to 8 times faster) than a software-based general purpose processor (Xilinx MicroBlaze), while leading to a reduction in resource overhead if the processor is duplicated (for fault detection purposes).

REFERENCES


