Decentralized Run-Time Recovery Mechanism for Transient and Permanent Hardware Faults for Space-borne FPGA-based Computing Systems

Victor Dumitriu, Lev Kirischian
Department of Electical and Computer Engineering
Ryerson University,
350 Victoria Street
Toronto, Canada
Email: vdumitri@ee.ryerson.ca, lkirisch@ee.ryerson.ca

Valeri Kirischian
MDA Space Missions
9445 Airport Road
Brampton, Canada
Email: valeri.kirischian@mdacorporation.com

***Abstract*—One of the most important problems for mission critical space-borne computing systems employing FPGA devices is fault tolerance to transient and permanent hardware faults. In many cases, the ability for run-time self-recovery from such faults is a vital feature. This paper presents a method and mechanism for run-time recovery of FPGA-based System-on-Chip (SoC) based on Collaborative Macro-Function Units (CMFUs). Each CMFU consist of a macro-function specific data-path, control unit and circuits providing self-integration, self-synchronization and self-recovery functions for the CMFU, without centralized control. The proposed mechanism allows run-time scrubbing or relocation of faulty components of the SoC providing much higher flexibility and reliability of the system. This mechanism was implemented and tested on a Xilinx Kintex-7 FPGA platform. It was determined that the proposed approach can provide seamless run-time recovery for pipelined SoCs, while being transparent to the application.**

Keywords*—fault-tolerance, reconfigurable computing, self-recovery, space-borne FPGA systems.

I. Introduction

During the last decade Field Programmable Gate Array (FPGA) devices have become the core component for space-borne high performance computing platforms. However, certain factors can result in hardware faults in FPGAs. The major hardware faults are radiation-induced faults which can be transient or permanent. Transient faults are usually caused by Single Event Effects (SEEs) affecting the memory units in SRAM based FPGAs. Mitigation of transient hardware faults in configuration memory requires full or partial re-programming of the configuration memory, to avoid improper functionality of logic blocks and/or routing topology in the FPGA [1]. Permanent faults can also occur in FPGA circuits due to Total Ionizing Dose (TID), and will also requires mitigation [2]. There exist certain radiation hardening technologies for CMOS circuits, implemented in space-grade, radiation hardened FPGA devices. These FPGAs can provide immunity for TID of more than 1 Mrad (Si) [3], but are one order of magnitude more expensive than the same FPGAs designed for terrestrial applications. As well, the availability of such FPGAs for commercial applications is limited by production and trade regulations.

978-1-4799-5356-1/14/$31.00

The growing demand for commercial space-borne computing platforms requires more cost-efficient solutions based on commercially available FPGAs. In this case the radiation immunity must be provided at the system and architectural levels of the design, instead of the technological level. Taking into account the fact that computing platforms deployed on Low-Earth Orbit (LEO) satellites may not need radiation immunity for TID exceeding 20-50 Krad (Si), the cost for space-borne FPGA-based platforms can be significantly reduced by avoiding space-grade FPGA devices in the system design. In this case, the computing platform should be equipped with a Built-In-Self-Recovery (BISR) system, to provide run-time recovery from transient and permanent hardware faults. Therefore, the objective of the presented research was the development of a robust BISR mechanism for stream processing systems. A novel methodology has been developed for this BISR mechanism based on the self-composition of a pipelined, stream-processing data-path, utilizing the concept of Collaborative Macro-Function Units (CMFUs) [4]. This methodology assumes that CMFUs - functional components of the data-path - can collaborate with each other to form a processing pipeline adapted to the requested mode of operation. Any hardware fault is considered a distortion of system functionality and the associated CMFU must be restored. Thus, the recovery process from a hardware fault consists of self-isolation of the faulty CMFU from the pipe-line, its self-recovery by scrubbing or relocation, and self-integration to the system, all without any central control. Lack of centralized BISR management allows for a significant increase of system survivability due to distribution of built-in self-test (BIST) and BISR functions among the components (CMFUs). The remainder of the paper presents an overview of the related works, and a presentation of the major aspects of the proposed mechanism, including experimental results gained using a prototype system based on the Xilinx Kintex-7 FPGA platform.

II. Background and Related Works

Mitigation of radiation-induced hardware faults in SRAM-based FPGAs should consider both transient and permanent faults. Transient faults are caused by Single Event Effects (SEE): a) Single Event Upsets (SEUs) and b) Single Event Functional Interrupts (SEFIs). These effects are induced by

high-energy subatomic particles from the sun and cosmic radiation. The faults may affect almost all resource of the FPGA: i) data memory blocks, b) logic blocks, c) clock management circuits, d) DSP-blocks and e) I/O blocks, etc. [5]. At the same time, SEUs and SEFIs can affect configuration memory and configuration management/port circuits of the FPGA [6]. This may jeopardize functionality of logic blocks and communication routing. The data memory content can be run-time corrected using error-correcting bits in RAM blocks. The configuration memory context can be restored by re-loading the configuration bit-stream to the affected region. This technique, called scrubbing of the configuration memory, became the de-facto standard procedure for SEE mitigation in SRAM-based FPGAs. An overview of potential SEUs and SEFIs, as well as suggested methods for mitigating the above transient faults in configuration memory and ports for Xilinx Virtex FPGAs, is provided in Xilinx Application Note 1088 (v1.0) [6]. Nonetheless, extensive research into tolerance of radiation effects for FPGA-based systems for space applications has been continued by many research groups [7].

Recently, permanent hardware faults have became an important factor for space-borne platforms. There are several causes for the above faults: a) radiation effects such as Single Event Latch-up (SEL) and Total Ionizing Doze (TID), b) hidden manufacturing defects and c) aging of the die. All of the above cases should be considered for mission-critical and long-range space systems. The progress in deep sub-micron CMOS technology has allowed a higher concentration of resources in a single FPGA. In turn, the probability for on-chip permanent faults has increased accordingly. On the other hand, the reduction of transistor dimensions has led to a reduction of FPGA core voltages; therefore, the probability of Single Event Latch-ups (SEL) has become almost negligible. However, TID still remains as the major cause of radiation-induced permanent faults. TID mitigation can be done at three levels: a) technological b) system and c) architectural.

The technological level assumes the use of radiation hardening technology during FPGA production [3]. This approach relies on the use of manufacturing technologies for hardening CMOS gates which can provide a significant increase in tolerance to TID (e.g. [8]). However, the FPGA device cost can be an order of magnitude higher than the same commercial-grade FPGA. Another potential problem associated with utilization of TID hardened FPGA devices is market accessibility to these FPGAs. Quite often, different trade and/or technology restrictions pose a limiting factor for the utilization of radiation-hardened, space-grade FPGAs in commercial application.

Protection of circuits at the system level assumes shielding either the entire system, or only vital devices. Tantalum and/or Tungsten is often used instead of aluminum for shielding. These high electron number materials have about 6 times higher density in comparison to aluminum [7]. The obvious drawback of this approach is the mass/weight overhead. Added to that, a second negative effect of shielding can be an increased rate of SEEs, in case of thick shielding, due to multiple secondary particles caused by interactions of cosmic rays and the shield material [7]. On the other hand, according to [9], shielding can provide a ten-fold TID reduction (from 100 Krad to 10 Krad (Si)) in a 3 year mission. This approach requires 4 g/cm2 of shielding (for aluminum); TID can be further reduced to approximately 8 Krad (Si) if the above shielding is doubled (10 g/cm2). Furthermore, electron impact can be eliminated with a shield exceeding 3 g/cm2 [9]. Based on the above, shielding FPGA-based space-borne systems gives limited protection. It completely eliminates the impact of electrons and reduces TID of trapped protons by an order of magnitude through the use of relatively light and cheap aluminum shielding (4 g/cm2) [9]. However, shielding does not reduce the rate of SEEs caused by high-energy cosmic rays, and can even increase it. Likewise, shielding will not prevent permanent faults caused by hidden manufacturing defects and aging of the device die.

The third level for radiation-hardening an FPGA system is the architectural level. The main approach for radiation hardening an FPGA on the architectural level is the implementation of Built-In-Self-Recovery (BISR) systems which assume the incorporation of Built-In-Self-Test (BIST) methods. The presented research, however, is focused only on the self-recovery aspect. Methods for self-test, fault detection and location for digital systems are well known (e.g. [10]) and their development was not the subject of this research. Radiation hardening on the architectural level assumes that the recovery mechanism provides recovery from transient and permanent faults. As was mentioned above, most methods for recovery from SEEs utilize scrubbing of the affected parts of the FPGA [6]. On the other hand, recovery from permanent faults is possible only by avoiding the physically damaged regions of the FPGA. If the faulty region is considered at the functional unit level, such as in distributed systems, recovery may take the form of re-allocation of tasks from a faulty unit to a backup/redundant unit. In the case of multi-processor SoCs, tasks mapped to a faulty processor could be re-assigned to other processors (e.g. [11]).

If the fault region being considered is scaled down to part of the FPGA, the recovery methodology is oriented towards relocation of components from faulty regions to reserve regions on the FPGA. Taking into account that the nature of the fault (transient or permanent) is unknown at the moment of detection, BISR must provide both recovery procedures: from transient and from permanent faults. Since transient fault happens more often, BISR should perform scrubbing procedures first. A multi-level approach for on-chip BISR was proposed in [12]. Three levels of recovery were considered: i) transient fault recovery by scrubbing; ii) permanent recovery by relocation of Virtual Hardware Component (VHC) to a reserved part (slot) of the FPGA in the case when scrubbing is not successful and iii) permanent fault recovery when the reserved slot is already used through allocation of a smaller VHC to the faulty slot, avoiding the damaged half of the slot (recovery with performance degradation). A similar approach for recovery was used for on-chip distributed systems [13]. This SoC consist of four processors, each of which is capable of reconfiguring the others. This reconfiguration takes the form of scrubbing and in case of a permanent fault, the tasks associated with the faulty processor can be re-allocated to another processor. Another approach for self-recovery from permanent faults caused by aging effects is given in [14]. This approach assumes the creation of multiple configuration bit-streams, each of which avoids a certain region of the FPGA. Therefore, an SoC can be recovered by multiple re-configuration cycles. However, this recovery method may take quite a long time due to multiple

configuration iterations.

Most existing approaches (including all of the above) are based on a central, CPU-centric diagnostic and reconfiguration mechanisms. The main drawback of this organization is that the central BISR management circuits can also be affected by radiation or aging effects. In this case, when the on-chip "immune system" is faulty, the next hardware fault may be catastrophic for the entire system. Utilization of the TMR approach for the entire BISR may cause very high hardware/power overhead. In addition, a CPU-based BISR management mechanism is relatively slow when compared to dedicated recovery circuits. Thus, decentralization of recovery procedures could be an effective solution which could increase system reliability and decrease recovery time. In this approach, function-specific BISR mechanisms should be embedded in function-specific units which compose the application-specific pipeline. This approach has been taken as the basis for the proposed research. The presented concept is based on Collaborative Macro-Function Units (CMFUs) [4] each of which can communicate with other CMFUs via a distributed bus. A fault detected by any CMFU is considered a change of mode of operation; all other components (CMFUs) will terminate their operation upon receiving notification regarding said fault. Therefore, the proposed approach does not require a central on-chip BISR mechanism, as all BISR functions are distributed between CMFUs. The following sections will describe the architecture of the proposed system, an example implementation and test results.

III. Distributed Architecture for Fault Tolerance and Mitigation

As per the above discussion, two primary types of faults must be considered and mitigated: transient and permanent faults. Transient faults are mitigated via full or partial scrubbing of the device. Permanent faults can only be fixed by changing the on-chip architecture of the system, such that faulty hardware is avoided. At the component level, this takes the form of relocation, whereby a component is loaded to a new region of the device, through the use of a partial bit-stream. Mitigation by itself is not enough, however: a fault tolerant system must be able to ensure that no single fault can irreparably cripple the system, and that fault effects are kept to a minimum.

To ensure that no single point of failure exists in the system, and that faults cannot affect the complete system, a distributed architecture is adopted; the general on-chip architecture is shown in Figure 1. The system is composed of Collaborative Macro-Function Units (CMFUs) loaded into *system slots* (reserved Partially Reconfigurable Regions), a distributed communication and control infrastructure, and a distributed configuration management system. The system CMFUs implement the application (or applications) deployed to the system. The distributed infrastructure ferries information between CMFUs, and arranges connection and disconnection activities in the system. Finally, two Configuration Engines are responsible for loading partial bit-streams to the system; the proposed approach makes use of the two Internal Configuration Access Ports (ICAP) [15] available in modern Xilinx devices. Only one of the two configuration engines is in active use at any time, but either can be used, if a fault is detected in the other.

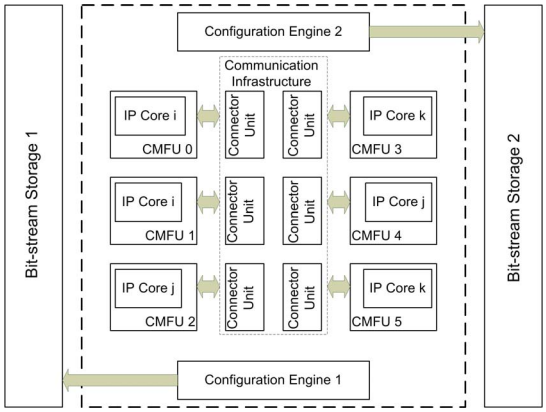

Fig. 1. General On-Chip Architecture

The distributed nature of the system implies no central control or synchronization entity; a fully working system is assembled via the interaction of multiple on-chip elements. The system CMFUs must be capable of functioning autonomously once stream information is presented to them (i.e. schedule processing operations as needed). The distributed elements of the communication infrastructure, in conjunction with system CMFUs, must be able to establish the correct data links, and be able to update said links as components are moved around the system, added to the system, or removed. Finally, CMFUs, distributed infrastructure and configuration engines must be able to detect faults in the system, and undertake the requisite mitigation procedures.

Because of the distributed nature of the architecture, each part of the system can be implemented as a Partially Reconfigurable Module (PRM), including the configuration engines. Like-wise, each PRM is expected to incorporate Built-In Self-Test (BIST) capabilities, and can broadcast to the rest of the system its health status. The health status information is translated into mode information by the configuration engines, which is then used by the rest of the system. Generally, different modes of operation dictate different architectural organizations of a given system. When applied to fault tolerance, these architectural variations take the form of A) relocated components or B) removed components (degraded performance due to lack of available resources). The core mitigation process activity is the relocation of a component to a separate, backup *system slot* (partially reconfigurable region). This approach is used whenever a fault is detected, since it is not known if the fault is permanent or transient. Once the component is relocated, scrubbing operations can be undertaken on the vacated system slot; if scrubbing operations fail, then the fault is marked as permanent. What must be noted is the fact that mitigation via scrubbing consists of more than simply downloading of a partial bit-stream; the process must also include testing of scrubbed region, and multiple scrubbing cycles may be required. Using the approach proposed in this paper, the mitigation cost for any single fault is the single component relocation cost, regardless of the type of fault.

To support the above behavior, all system building blocks must be designed to operate in a distributed fashion. In the

following two sections, the architecture and behavior of CMFUs and the distributed infrastructure are described. In each case, the required behavior is first described, followed by a description of the implemented architecture. The configuration engines are not discussed in this paper; their design and implementation will be more thoroughly covered in future works.

IV. Collaborative Macro-Function Unit

CMFUs are macro-function-level processors. They are usually based on customizable (to a certain extent) IP cores; these cores are themselves macro-function-specific, built to perform an operation on a specific data structure. These cores are augmented with additional functionality, so that they may operate in an autonomous fashion . This functionality is implemented in the form of a *Co-Op Unit* which is added to the IP core to construct a CMFU, as shown in Figure 1. The Co-Op Unit acts as the behavioral interface of the IP core to the system infrastructure, and must perform three tasks: *scheduling of processing and BIST activities*, *scheduling of connection and disconnection activities* and *mode monitoring and connectivity requests*.

A. Scheduling of Processing and BIST Activities

IP cores are most often built to perform a certain operation over a certain, finite data structure. A data stream is taken to be infinite in length, which means an IP core must be triggered repeatedly, at the right time, to process new incoming data. The Co-Op Unit can use the data stream itself to determine when IP cores must be triggered. Most data streams (video, audio, network information) have a certain structure, and are often accompanied by supplementary information, such as start-of-line flags, or packet headers. The Co-Op unit uses this stream information (henceforth referred to as *indicators*)to determine when to trigger processing operations in the IP core; it also generates output stream indicators, for use by down-stream CMFUs. Each CMFU is also expected to incorporated BIST capabilities. BIST operations must be undertaken periodically, usually when regular processing operations are complete. The Co-Op unit is responsible for triggering the BIST circuitry in the system, collecting the test results, and distributing them to the rest of the system via the distributed communication infrastructure.

B. Scheduling of Connections and Disconnections

CMFUs can be connected and disconnected to and from the system, depending on the mode of operation; disconnected CMFUs can be scrubbed or relocated, as part of mitigation procedures. Connection consists of creating a physical link to the CMFU input port such that up-stream data can be reach the CMFU; disconnection consists of isolating the CMFU from the system, so that no external data can reach it. To ensure that components are not disconnected during processing operations, a *safe state* is defined when the IP core is not performing processing operations; at this time, the CMFU may be disconnected from the system. The Co-Op Unit is expected to know how long processing operations take (either through hard-coding, or through feed-back from the IP core). Using this information, the Co-Op unit can negotiate connections and disconnections using a two-way hand shake protocol, shown in Figure 2; the *disc* signal is driven by the infrastructure, while the *safe* signal is driven by the Co-Op Unit. Two versions of the protocol can be used, depending on whether the communication infrastructure initiates the protocol (Figure 2 - A), or whether the Co-Op unit initiates the protocol (Figure 2 - B).

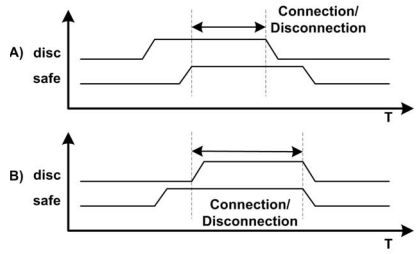

Fig. 2. 2-Way Hand-Shake Protocol for Connection/Disconnection

C. Mode Monitoring, Connectivity Requests

Each CMFU is responsible for requesting its own connectivity for a given mode. The communication infrastructure is not expected to have any implicit information regarding the CMFU distribution in the system. As such, each CMFU must monitor the system mode. Based on the mode, the CMFU must indicate whether it should remain in the system or be disconnected, and whether a change in connectivity is required. To accomplish this, the Co-Op unit incorporates a small look-up table of connectivity information, associated with the mode input, and specific to the CMFU in question. The mode is sampled at the end of processing operations, immediately before entering the safe state.

D. Co-Op Unit Structure and Behavior

Following the above guidelines, an architecture for the Co-Op Unit emerges: it is a Finite-State Machine, possibly augmented with counters and comparators, and a look-up table. This means that the Co-Op unit will not add much in the way of resource overhead to the IP cores being used. The behavior of the FSM is shown in Figure 3. The co-op unit starts in its idle state, which is also its *safe state*. It will stay here if the *disc* signal from the infrastructure is asserted. If the CMFU is connected to the system, it will wait for the right combination of indicators, trigger the IP core, wait for processing to complete, sample the mode, and determine if connectivity stays the same, or if it must change (either disconnection or change in data source). If the CMFU is disconnected, but still in the system (it has not been replaced by a blank bit-stream), then it periodically samples the mode, and determines if it must be re-connected to the system.

To accomplish connectivity changes, the CMFU makes use of a dedicated control interface to the system infrastructure. The control interface contains a 3-bit status, a 5-bit system ID (meaning that up to 31 components are supported in this protocol; value 11111 is reserved), the *safe* and *disc* signals, and a 2-bit error indicator. The status can have one of three values: 101 (component is operating normally), 010(component requests connectivity change) and 110 (component requests disconnection from system). A fourth value, 111 is reserved for when the component is replaced by a blank design.

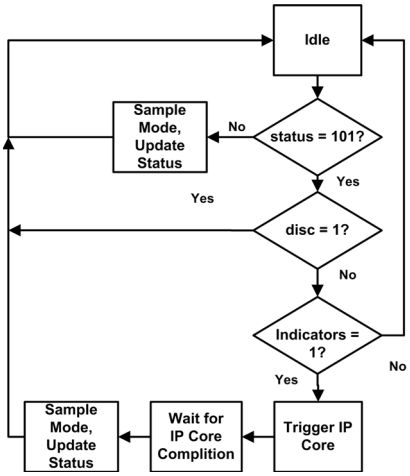

Fig. 3. Co-op Unit FSM Behavior

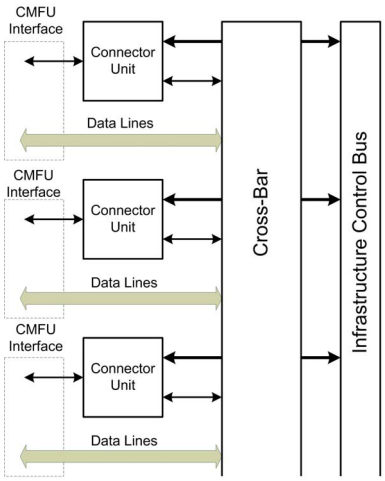

Fig. 4. Communication and Control Infrastructure Architecture

V. Distributed Control and Communication Infrastructure

The system infrastructure must accomplish three roles: 1) it must provide communication capabilities to the components in the system; 2) it must be able to implement the connectivity requirements of components and respond to connectivity change requests correctly (by correctly implementing connections and disconnections); and 3) it must distribute health information for the system components (including its own sub-systems). The infrastructure should also lend itself to fault tolerance and fault mitigation. Each of these three aspects is elaborated further below, and the final architecture of the proposed infrastructure is presented.

A. System Communication

One of the core requirements of the infrastructure is to provide data links between CMFUs. Given, that high data rate stream processing is explicit target application space, the communication infrastructure should provide maximum throughput on every link in the system. Second, given that some or all CMFUs in the system can be relocated, it should allow for changes in the established communication links, for preference with minimum overhead. With these requirements in mind, a fully connected cross-bar is selected as the base communication infrastructure; more details can be found in [16].

B. System Connectivity Behavior

When receiving a connectivity request, the infrastructure elements wait until all components enter their safe states and then make all requisite changes. This approach is simple in terms of infrastructure behavior, but relies on the implicit assumption that the safe states of all components will overlap at some point, and will overlap for enough time to allow the infrastructure to make all required changes. This assumption is satisfied in situations where CMFUs work on data structures which are "multiples" of each other (for example, a frame is a multiple of a single line in a video stream), and the pipeline latency is a fraction of the *safe* period for all CMFUs. This approach is adopted in the work presented here, as its reduced complexity lends itself more to fault-tolerant design.

C. Fault Tolerance and Mitigation by Component Relocation

When faults occur, any mitigation action will have some associated time penalty; until the mitigation action takes place, the best case scenario is that the fault is localized both in terms of location as well as effect. To facilitate this, the system functionality should be distributed, such that, if any one portion of the system is found faulty, the remainder can still operate correctly. As described above, the main mitigation activity is relocation of CMFUs, potentially followed by scrubbing. If the infrastructure is divided into sub-modules, each of which is associated with a system slot, then relocation from one slot to another will by-pass the associated infrastructure element; in other words, relocation can now mitigate faults in both the system slot, as well as part of the infrastructure logic. This same concept applies to the cross-bar, where logic is distributed on a per-port basis.

In the case of transient faults in the infrastructure, scrubbing operations should be applied to the faulty element, once relocation has been applied. At the same time, the system should keep operating normally. This implies that each element of the infrastructure should be implemented as a separate PRM, so that scrubbing operations can be applied on a per-port basis, without affecting the rest of the system.

D. Distributed Communication and Control Infrastructure

The complete communication and control infrastructure consists of a fully connected cross-bar switch, a number of *Connection Units* and a distributed Control Distribution Bus. The architecture of the infrastructure is shown in Figure 4. All connector units, cross-bar and Control Bus can be implemented as separate modules (and, thus, individual PRMs). The Connector Units are the main point of interface to system CMFUs; each Connector Unit is associated with one system slot (and will interact with the CMFU loaded there), and all Connector Units together collaborate to safely connect and disconnect components from the system. A Connector Unit responds to connectivity requests from the CMFU, negotiates with all other Connector Units, and finally makes local connectivity changes at the correct time (when all components are in a safe state).

For correct operation, each Connector Unit must know what CMFUs are loaded in the rest of the system (what system ID each CMFU loaded in the system has). Likewise, Connector Units must indicate if they are attached to a CMFU, or if they are associated with a blank bit-stream. This information is sent out in serial fashion by the Control Bus, which collects this information from all Connector Units, and then broadcasts it. In this way, each Connector unit can determine what link to establish in the cross-bar, based on the system ID of all CMFUs in the system. The Control Bus also broadcasts the *Global Safe* signal, which indicates when all CMFUs are in their safe state, and all Connector Units can perform connectivity changes.

The connector unit itself consists primarily of an FSM which accomplished the connection and disconnection process; a secondary control circuit is used to collect system information broadcast over the Control Bus. The behavior of the Connector Unit FSM is outlined in Figure 5. The connector unit spends the majority of its time in its Idle state; it only leaves this state if a change in connectivity was requested, locally or globally. If the change in status coming from the CMFU is 110 to 111 (or vice versa) no changes are made. If a connectivity change is otherwise required (disconnection, change in source, or global disconnection request), the FSM enters a wait state until the *Global Safe* signal is asserted. Depending on the status value of the CMFU one of three things will happen: no changes are made, the CMFU is disconnected, or the CMFU link in the cross-bar is changed to reflect a new connectivity requirement. To determine the correct setting for the cross-bar, system ID information for all other CMFUs in the system is used.

VI. Framework Test and Verification - Video Processor

To test the proposed framework, an example stream processor was devised and implemented. The operations which were selected and implemented belong to the field of raw video processing, and were selected due to their high data rate and real-time nature. The following sections describe the processor implementation, the available support for fault mitigation procedures, and finally the resource cost associated with the proposed framework.

A. Video Processing System

Spatial color masks were selected as operations to be applied to a video stream; the proposed system consists of a 720P video source, a color-space converted (converts 24-bit RGB values to YCbCr 4:2:2 format), an HDMI display unit driving an HDMI transmitter and system slots which can accommodate two types of components: a Sobel edge extractor and a median filter. The basic processor supports one of three basic modes of operation: 1) no masks are applied, 2) only Sobel edge extraction is applied, and 3) both the median filter and the Sobel edge extraction are applied. A final mode is defined for some forms of fault mitigation procedures; in this mode all components are disconnected from the system. The three basic modes are duplicated, so that additional modes dictate relocated CMFUs. Each component of the processor must support a data rate of 1.23 Gb/s to meet the 720p60Hz standard. The system was designed for the Xilinx

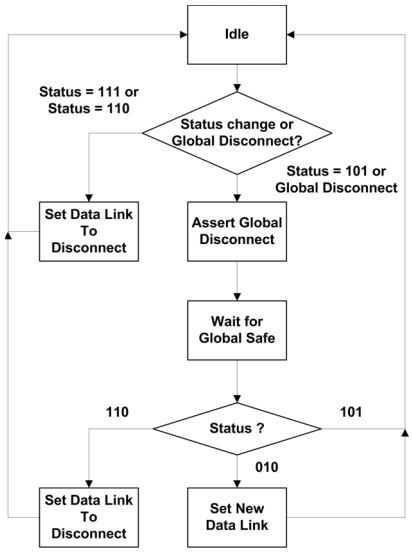

Fig. 5. Connector Unit FSM Behavior

XC7K325T device, and deployed on the KC705 evaluation platform. To generate the video stream, a pattern generator was implemented on-chip.

Figure 6 shows the implemented system architecture; the system consists of 7 *slots*, 4 of which can house any one of the two spatial color masks. Every component in the processing pipeline is implemented as a PRM and can be scrubbed (pattern generator, color space converted, display unit, as well as both masks); the spatial color masks can also be relocated between four of the system slots. All Connector Units, as well as the central cross-bar can also be scrubbed separately. Figure 7 shows the on-chip location of the PRRs associated with each system component.

B. Fault Mitigation

Fault mitigation is accomplished by relocation (temporarily, in the case of scrubbing procedures). By relocating a CMFU, the system changes the slot being used to house said CMFU, as well as the associated Connector Unit, and port logic in the cross-bar; this means that relocation of the CMFU can mitigate faults in a system slot, cross-bar logic, or connector units. Once the component is relocated and reintegrated into the working system, partial scrubbing of the faulty sub-system can be performed, followed by tests to determine if the fault was transient or permanent. This leads to the following process for mitigation, once a fault is detected.

1) Download new bit-stream for the component, targeting a different slot.
2) Change operating mode, such that the relocated component is used.
3) Load a test bit-stream into the faulty slot or connector unit PRR.
4) Run BIST procedures, determine if fault is permanent.

Each of the four steps listed above has an associated timing cost. However, only the first two items are *visible* to the application, and manifest themselves as a time delay T_{mtg},

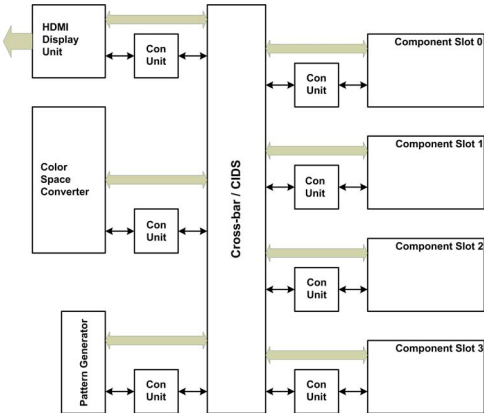

Fig. 6. Example System Architecture

the time from the detection of a fault to its mitigation. T_{mtg} is composed of the bit-stream download time, and the mode change operation. The download time for one system slot bit-stream is 1.1ms; this value is based on the assumption that a 32-bit interface is used (the ICAP port) at 100 MHz, and the bit-stream is known to have 366 KB. The actual connectivity changes require 25 clock cycles of the system clock (the pixel clock is used, at 74.24 MHz); this equates to an additional 400 ns. As described above, all connectivity changes occur when components are in their safe state; in the case of a video processing system, this happens at the end of each frame. Faults detected in the middle of a frame will be mitigated at the end of the frame; this means that a single fault will affect only a single frame. The last two steps are performed in parallel with regular operation, and do not affect regular system operation. Because all CMFUs can be loaded to any of multiple slots, once relocated, a CMFU can remain in its new location, and its previous location becomes a spare slot if it is found to be free of permanent faults.

Faults in the system cross-bar are primarily handled on a per-port basis, through relocation. However, it may be necessary in certain instances to scrub the cross-bar itself. In such cases, the most basic mitigation process follows the following four steps.

1) Enter mode 3 (all components are disconnected).
2) Scrub the cross-bar.
3) Return to regular operation.

Depending on when the procedure is implemented in relation to the video stream, Step 1 can take up to 16.67 ms, if the system has to wait a full frame for the next safe state. The connectivity change proper takes, as before, 25 clock cycles or 400 ns. The scrubbing operation takes 1.23 ms (due to the larger size of the cross-bar bit-stream). Finally, step 3 requires another 400 ns. The complete mitigation process requires slightly more than 17 ms, meaning that it affects 2 frames. Scrubbing of the cross-bar requires the disconnection of all components prior to the scrubbing operation; the bit-stream download time is not overlapped with the regular operation of the system.

By supporting relocation of components, fault mitigation for both permanent and transient faults is reduced to the same temporal cost, from the application point of view. Furthermore,

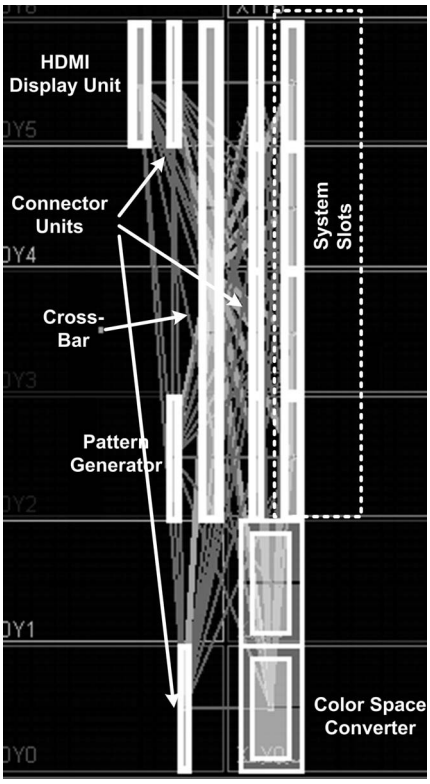

Fig. 7. Example System On-Chip Layout

adopting this approach towards fault mitigation, based entirely on the use of partial bit-streams, allows for a considerable speed-up in mitigation time when compared to full bit-stream scrubbing, or relocation using full bit-stream variations. When considering devices in the Xilinx Kintex line, the mitigation time can be reduced by at least a factor of 10, and by as much as a factor of 50, depending on the device used.

C. Resource Costs and Overhead

The proposed infrastructure requires the addition of circuitry to the system elements implementing the core functionality. This comes in the form of the Co-Op unit in the CMFU, the Connector Units and the system cross-bar. Figure 8 shows the number of look-up tables used per slot, as the number of system slots is increased; the number of d-flip-flops used per slot is constant, and consists of 52 d-flip-flops for the cross-bar, and 48 d-flip-flops for the combined Co-Op and Connector units. In each case, the overhead due to the Co-Op unit and Connector unit is nearly constant (as one of each is needed per system slot), and amounts to 115 look-up tables. The cross-bar overhead, as expected, increases more dramatically. However, when compared with the resources available in one system slot (300 slices, containing 1200 look-up tables and 2400 D-flip-flops), the per-slot overhead (in total) is less than 35%; this is the worst case, where 32 slots are used, and the more than half of the logic overhead comes from the cross-bar.

The proposed method was compared to traditional detection and mitigation methods such as duplex and triple-module redundancy (TMR) methods. In duplex systems, a more than 100% overhead is incurred (as everything is doubled), while TMR sollutions lead to more than 200% overhead. Table I lists

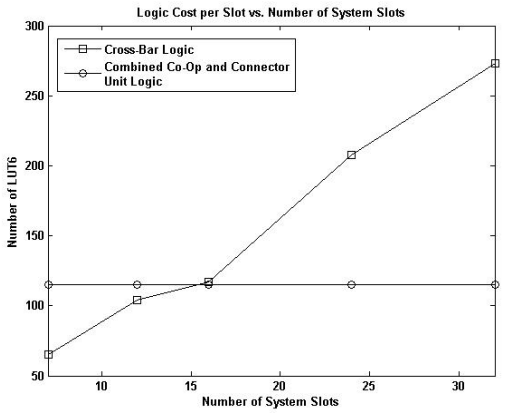

Fig. 8. Combinational Logic Cost per Slot, Versus Number of System Slots

resource number for various sized systems, ranging from 7 to 32 slots. In each case, 2 of the system slots are spare slots, while the remainder are used to house active logic. To facilitate resource reporting, the table lists resources in terms of Kintex 7 slices; one slot is assigned 300 slices, and one Co-Op and Connector unit pair were found to require 50 slices. The table lists the logic overhead due to Co-Op Units, Connector Units and spare system slots; it does not iclude the crossbar into this computation, as the crossbar (or some other high aggregate data-rate communication infrastructure) is a non-avoidable cost of such a system. As the final column shows, the overhead associated with the proposed method never exceeds 65%, and drops to less than 25% in large systems; this overhead is considerably smaller than either the duplex or TMR sollutions, especially in large systems.

TABLE I. RESOURCE OVERHEAD COMPARISON

System Size	Usable Logic	Overhead Logic	Overhead %
7	1500	950	63.34
12	3000	1200	40
16	4200	1400	33.34
24	6600	1800	27.28
32	9000	2200	24.45

VII. CONCLUSION

The adoption of commercial SRAM-based FPGAs for space-borne applications requires the use of new fault tolerance and mitigation techniques; FPGA sensitivity to transient Single Event Effects (SEEs), coupled with the potential for permanent faults due to radiation, manufacturing defects or aging require techniques for transient and permanent fault mitigation. This paper presents an attempt at solving this problem. The proposed approach is based on a de-centralized architecture for stream processors which eliminates single points of failure, and allows for mitigation of both transient and permanent faults by dynamic component re-location. Specifically, on-chip relocation through Dynamic Partial Reconfiguration is used as a mitigating procedure for both types of faults. By using direct relocation as opposed to multiple cycles of scrubbing, read-back and testing, the temporal cost of all mitigation procedures is reduced to the temporal cost for relocation, and is kept constant. Thus, from the application point of view, recovery from single point faults takes the same time and is therefore predictable. Furthermore, the relocation based recovery can be done on-the-fly seamless to data-execution processes. The proposed approach has been tested on a Xilinx Kintex-7 FPGA platform. The relocation time - now equal to the recovery time - has been reduced from one to two orders of magnitude when compared with traditional scrubbing procedures for the entire FPGA device. As well, the hardware overhead per slot does not exceed 35% of a single component slot for a 32-slot system while providing a run-time recovery mechanism for transient and permanent faults.

ACKNOWLEDGMENT

The authors wish to acknowledge the support of the Ontario Centers of Excellence (OCE), MDA Space Missions, and the Natural Sciences and Engineering Research Council (NSERC) of Canada.

REFERENCES

[1] F. L. Kastensmidt, L. Carro, and R. A. da Luz Reis, *Fault-tolerance Techniques for SRAM-based FPGAs*. Dordrecht, The Netherlands: Springer, 2006.

[2] F. Smith and S. Mostert, "Total ionizing dose mitigation by means of reconfigurable fpga computing," *IEEE Transactions on Nuclear Science*, vol. 54, no. 4, pp. 1343–1349, 2007.

[3] *DS192: Radiation-Hardened, Space-Grade Virtex-5QV Family overview*, Xilinx, 2012.

[4] V. Dumitriu and L. Kirischian, "Soc self-integration mechanism for dynamic reconfigurable systems based on collaborative macro-function units," in *Proceedings of the 2013 International Conference on Reconfigurable Computing and FPGAs*, Cancun, Mexico, Dec. 2013.

[5] G. Allen, *Virtex-4VQ dynamic and mitigated single event upset characterization summary*. Pasadena, CA: Jet Propulsion Laboratory, National Aeronautics and Space Administration, 2009.

[6] C. Carmichael and C. Tseng, "Xapp1088: Correcting single-event upsets in virtex-4 fpga configuration memory (v1.0)," Xilinx, Tech. Rep., Oct. 2009.

[7] R. Maurer, M. Fraeman, M. Martin, and D. Roth, "Harsh environments: Space radiation environment, effects, and mitigation," *Johns Hopkins APL Technical Digest*, vol. 28, no. 1, pp. 17–29, 2008.

[8] K. Hayes and R. Katz, "Design techniques for radiation-hardened field programmable gate arrays, rev 1," Actel Corporation, Tech. Rep., 1996.

[9] E. Stassinopoulos, G. Brucker, and C. Stauffer, "Crres microelectronics package flight data analysis," *Scientific and Technical Information Program*, 1993.

[10] M. Abramovici, M. A. Breuer, and A. D. Friedman, *Digital systems testing and testable design*. Hoboken, New Jersey: Wiley-IEEE Press, 1990.

[11] M. Fayyaz, T. Vladimirova, and J.-M. Caujolle, "Adaptive middleware design for satellite fault-tolerant distributed computing," in *Proceedings of the NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, Erlangen, Germany, 2012.

[12] L. Kirischian, V. Geurkov, V. Kirischian, J. Kleiman, and I. Terterian, "Multi-level radiation protection of partially reconfigurable fpga devices," *Journal of Spacecrafts and Rockets*, vol. 43, pp. 523–529, 2006.

[13] H.-M. Pham, L. Devaux, and S. Pillement, "Re2da: Reliable and reconfigurable dynamic architecture," in *Proceedings of the 6th International Workshop on Reconfigurable Communication-centric Systems-on-Chip*, Montpellier, France, 2011, pp. 1–6.

[14] H. Zhang, L. Bauer, M. A. Kochte, E. Schneider, C. Braun, M. E. Imhof, H.-J. Wunderlich, and J. Henkel, "Module diversification: Fault tolerance and aging mitigation for runtime reconfigurable architectures," in *Proceedings of the IEEE International Test Conference (ITC)*, Anaheim, CA, USA, 2013, pp. 1–10.

[15] *7 Series FPGAs Configuration (v1.7)*, Xilinx, 2013.

[16] V. Dumitriu and L. Kirischian, "A framework of embedded reconfigurable systems based on re-locatable virtual components," *International Journal of Embedded Systems*, vol. 4, no. 3, pp. 182–194, 2010.