Guaranteed Fault Recovery Time for FPGA-based TMR Circuits Employing Partial Reconfiguration

Ediz Cetin, *Member, IEEE*, Oliver Diessel, *Member, IEEE*, Lingkan Gong, *Student Member, IEEE*, and Victor Lai

Abstract—Space-based *Field-Programmable Gate Array* (FPGA) systems are increasingly susceptible to radiation-induced *Single Event Upsets* (SEUs). Techniques for partially reconfiguring a corrupted module of a *Triple Modular Redundant* (TMR) implementation have been described in the literature. In this paper we outline a correct-by-construction method for partitioning any digital circuit so that it is protected using a TMR implementation and can be guaranteed to recover from any single FPGA memory error within a specified maximum fault recovery time. Our method ensures high reliability and guarantees availability assuming the minimum period of time between error events is greater than the specified maximum fault recovery time. The performance and overheads of the proposed method are evaluated for a number of representative circuits. The results indicate that area and clock frequency overheads as well as the time needed to recover from faults are acceptable for earth observation applications.

Index Terms—Fault tolerance, radiation induced errors, high availability, redundant design, reconfigurable hardware, dynamic partial reconfiguration.

1 Introduction

Increasingly, the space sector plays a central role in the efficient functioning of modern societies and their economic development. The use of satellite technology in navigation, communications, meteorology and earth observation is giving rise to a growing stream of applications in diverse areas such as air traffic control, transport, natural resource management, agriculture, environmental and climate monitoring and entertainment, thereby resulting in the creation of new downstream uses and markets for space-based systems [1]. Furthermore, space is increasingly seen as an important potential source of economic growth, social wellbeing and sustainable development [1].

Future earth observation missions are expected to have very high data rate requirements ranging from 10–60 GBit/sec and processing requirements nearing 1000 *Giga Operations Per Second* (GOPS) [2] – [4]. The next generation of on-board processing is also required to be flexible and re-programmable in-orbit and during active service. This brings about challenging requirements for future on-board processing systems that cannot be met with space-qualified processors available today [2]. The implementation devices most suited to meeting these requirements are *Field-Programmable Gate Arrays* (FPGAs). Not only do they, like custom hardware chips, provide the means for implementing high-speed digital systems, they can also be reconfigured on demand, like software running on a processor, to perform new or different functions. Furthermore, by reusing the same device to implement a variety of functions, FPGA reconfiguration can be exploited to reduce mission-critical parameters, such as the system's size, mass and power requirements, that must be kept as small as possible. In addition, FPGAs provide adaptable and evolvable platforms in which systems can autonomously adapt to changes in operating conditions and provide means for planned upgrades to processing algorithms during system life-cycles. The flexibility and scalability afforded by FPGAs thus provide the foundation for future, sustainable space technologies.

Benefits associated with *Commercial-Off-The-Shelf* (COTS) SRAM-based FPGAs in terms of processing performance, large logic area, low-cost and widely available design, simulation and synthesis tools make them ideal candidates to meet the demanding requirements of future on-board processing needs. The main challenge of using FPGAs for space applications is mitigating the effects of radiation-induced *Single Event Upsets* (SEUs). Given their reliance on configuration memory, FPGAs are more prone to these upsets when compared to *Application Specific Integrated Circuits* (ASICs), and this has been their Achilles heel detracting from their widespread use in space-based applications. Hence, detection and mitigation of effects of SEUs for satellite-based FPGA systems is of paramount importance.

- *E. Cetin is with the School of Surveying & Geospatial Engineering, University of New South Wales, Sydney, Australia. E-mail: e.cetin@unsw.edu.au*
- *O. Diessel, L. Gong and V. Lai are with the School of Computer Science & Engineering, University of New South Wales, Sydney, Australia. E-mail: {odiessel, lingkang, victorl}@cse.unsw.edu.au*

Numerous approaches dealing with the detection and mitigation of SEUs in SRAM-based FPGAs have been reported in the literature [5] – [26]. *Triple Modular Redundancy* (TMR) is one of the most popular approaches used in mitigating the effects of SEUs. With this approach, the user design is triplicated and the outputs from each module are fed into a voting circuit to determine the correct output value. TMR significantly improves system reliability and availability in the presence of SEUs [5], [6]. However, TMR results in substantial overhead in terms of area, clock frequency, and power consumption. Attention has focused on minimizing the overhead associated with TMR, in terms of area, clock speed and power consumption by utilizing selective/partial TMR and voter insertion [9] – [15]. With these approaches system reliability and availability is traded for a reduction in the TMR overhead. For some applications, the consequences may not be desirable. This paper proposes a novel TMR partitioning methodology aimed at maintaining a user-defined *system availability* requirement. In order to achieve high *system reliability* we advocate the use of TMR to protect all user circuitry. The methodology incorporates partial reconfiguration to quickly recover from SEU errors. The proposed methodology aims to find the best partition on which the TMR voters are inserted such that maximum delay in detecting and recovering from an SEU induced fault meets the user defined *system availability* requirement.

The paper is organized as follows: Section 2 provides a brief overview of FPGA technology that can be exploited for SEU detection and mitigation and discusses FPGA SEU susceptibility. Section 3 details various FPGA SEU mitigation techniques reported in the literature. Our proposed methodology for dealing with SEUs in FPGAs is given in Section 4. Section 5 presents the results of applying our proposed methodology to a number of representative circuits, while concluding remarks are provided in Section 6.

2 FPGA Architecture and SEU Susceptibility

This section provides a brief overview of FPGA architecture and SEU susceptibility to facilitate ease of understanding of SEU detection and mitigation strategies that will be detailed in the following sections.

2.1 FPGA Architecture

FPGAs are popular, generic devices for implementing any digital circuit. Their increasing popularity is based on their relatively low price, low development cost, quick time to market and flexibility. Due to the inherent regularity of their circuit structure, the density and speed of FPGAs, while lower than that of ASICs, track Moore's Law very closely. FPGAs are manufactured using the most advanced processes and are therefore increasingly used for high-speed, compute-intensive applications, particularly in low-volume applications.

In essence, an FPGA is comprised of a two-dimensional array of configurable logic blocks, typically employing lookup tables to implement combinational functions and flip-flops for implementing register transfer operations. The logic blocks are embedded in vertical and horizontal routing channels that provide a hierarchy of wires for interconnecting the logic via programmable connections. A circuit is implemented by configuring the logic blocks to implement desired functions and programming the switches to form the required interconnections. Modern FPGAs typically embed a number of additional structures in the array to enhance the density and provide commonly needed resources for target markets. These resources include RAM blocks, dedicated multiplier (and accumulator) blocks, clock management circuits and high speed IO blocks. These resources are also configured by setting associated control bits.

This paper is concerned with mitigating the effects of transient errors caused by radiation-induced SEUs. In a digital circuit, SEUs most commonly cause the state of individual flip-flops and RAM bits to toggle to an incorrect value. When digital circuits are implemented using FPGAs, there is a significant additional source of error due to the configuration memory used to implement the circuit. When the configuration memory is affected, an implemented user circuit may cease to function due to errors in the functional logic or wiring faults. This paper examines the use of so-called partial reconfiguration to selectively reconfigure a circuit component whose implementation has been corrupted. Currently only certain devices manufactured by a single vendor, Xilinx, are capable of such partial reconfiguration at run time. Competitors such as Altera have announced they are providing such capabilities but have not yet released details of their architecture and tools. Available information suggests they will offer a very similar approach to that employed by Xilinx [27]. The remainder of this paper therefore refers specifically to currently available Xilinx Virtex devices (Virtex-4 family and later) when using the term FPGA and reconfiguration in a general sense.

2.2 FPGA Configuration Memory

Recent Xilinx Virtex devices are divided into clock regions under the management of a local clock controller and provide dedicated, low-skew clock distribution nets. Each clock region spans a fixed number of *Configurable Logic Block* (CLB) rows e.g., 16 CLB rows in Virtex-4 [28], 20 in Virtex-5, 40 in Virtex-6 and 50 in Virtex-7 [29]. A Virtex clock region spans half the width of a device and a number of clock regions are stacked vertically to span the height of a device. Significantly, the configuration memory of Virtex devices is divided into so-called *frames* that span the height of a clock region. In Virtex-4 and Virtex-5 a frame is 41 (32-bit) words long (81 words in Virtex-6 and 101 words in Virtex-7) and represents a bit-slice of the configuration memory for the resources

located in a specific column of a particular clock region. In a Virtex-4 device, a CLB column and its neighboring routing switches are configured using between 22 and 23 consecutive frames. Other resources such as RAM and multiplier blocks are configured using sequences of consecutive frames as well.

An FPGA is configured when it is powered on by writing a complete configuration bitstream to one of several configuration ports. The so-called SelectMAP interface has the highest bandwidth of up to 32-bits at 100MHz. The bitstream is typically stored in off-chip flash memory, and is loaded at system boot time. The bitstream can be generated to include *Error Correcting Code* (ECC) bits for each frame. The ECC bits are needed to verify the stored bitstream is not corrupted and to correct both single and double bit errors in a frame. After the bitstream is loaded, the device is switched into user mode, whereby the ability to reconfigure the device while it is powered up can be selected during bitstream generation. Typically, the ability to reconfigure the device at run time is exploited to dynamically implement new functionality. In this paper, the ability to reconfigure the device is exploited to selectively overwrite the corrupted configuration of a circuit component with a correct one.

2.3 Partial Reconfiguration

It is possible to selectively reconfigure part of a Xilinx Virtex FPGA while it is operating (at run time). This is called dynamic partial reconfiguration, or just partial or dynamic reconfiguration. In order to partially reconfigure a device, a partial bitstream needs to be created and written to one of the configuration ports of the device. A partial bitstream includes address information that specifies which configuration frames are to be written.

The partial bitstream is either written to an external port, or if it is written at run time, it is commonly written to an *Internal Configuration Access Port* (ICAP) by user logic. As the ICAP is an internal replica of the external SelectMAP interface, it has the same bandwidth (400MB/s in all Virtex devices).

The time needed to partially reconfigure an FPGA is proportional to the number of frames comprising the partial bitstream. The number of frames to be written when reconfiguring a circuit is minimized by constraining the circuit height to one or more complete FPGA clock domains (the height of a frame) and by minimizing the width of the circuit layout (the width of the region in frames to be configured).

2.4 FPGA SEU Susceptibility

Given their reliance on configuration memory, FPGAs are more prone to SEU upsets when compared to ASICs. An SEU occurs when deposited charge causes a change of state in dynamic circuit elements. In FPGAs, SEUs can modify both the data being processed and the implemented digital circuits by modifying the configuration memory. Table I lists the worst-case SEU rates predicted for the configuration memory of a Xilinx Virtex-4 (XC4VLX200) device at various orbits [30].

Table 1
SEU rate predictions for Xilinx Virtex-4 device for various orbits [30].

Orbit	SEUs per device/day	Mean time to Upset (s)
LEO (560 km)	4.09	2.11×10^4
Polar (833 km)	1.49×10^4	5.81
GPS (20,200 km)	5.46×10^4	1.58
Geosynchronous (36,000 km)	6.20×10^4	1.39

Table I illustrates that the occurrence of SEUs increases with the radius of the orbit, and that at GPS orbits, for example, errors can occur almost every second. It should also be noted that the estimates of Table I do not include the error rates for the data processing memory of the device, and may therefore be more than 20% below the total SEU rates that could be experienced by the device[1].

SEUs manifest themselves in the form of *Single-Bit Upsets* (SBUs), where only one bit is affected by the SEU. However, with diminishing transistor sizes the likelihood of a proton or heavy ion causing *Multi-Bit Upsets* (MBUs) rather than just SBUs due to SEUs increases. Table II depicts the distribution between SBUs and MBUs for various Xilinx devices due to proton-induced radiation (63.3 MeV) [31].

As can be observed from Table II, as the FPGA capacity increases, more MBU events occur. MBUs further complicate SEU mitigation. MBUs violate the assumption that only one error exists in the system at a time, since an MBU can affect redundant circuit copies. Given that more than one error can exist due to MBUs, the likelihood of an SEU affecting two out of three modules in a TMR-based approach increases. Under such circumstances, since two out of three modules have matching but incorrect results, the voter selects the wrong value and the system fails to detect the error [32].

3 FPGA SEU Detection and Mitigation Techniques

Numerous approaches have been reported in the literature dealing with the detection and mitigation of SEUs [5] – [26]. Given the complex nature of the FPGA fabric and configuration memory architecture, finding an efficient technique in terms of area, performance and power is very challenging. SEUs not only affect the user system operation but also the configuration memory of the FPGA. There are various ways of dealing with SEU effects; most of these approaches utilize some means of redundancy in the spatial, temporal and information domains or a combination of these methods.

1. We have extrapolated this estimate from the predicted impact on the datapath circuitry for smaller devices as reported in [30].

Table 2
Distribution between SBU and MBU effects for various Xilinx devices due to proton-induced radiation (63.3 MeV) [31].

Device Family	Technology Node	Total Events	1-Bit Events	2-Bit Events	3-Bit Events	4-Bit Events
Virtex	250 nm	241,166	241,070 (99.996%)	96 (0.004%)	0 (0%)	0 (0%)
Virtex-II	150 nm	541,823	523,280 (98.42%)	6,293 (1.16%)	56 (0.01%)	3 (0.001%)
Virtex-II Pro	130 nm	10,430	10,292 (98.68%)	136 (1.30%)	2 (0.02%)	0 (0%)
Virtex-4	90 nm	152,577	147,902 (96.44%)	4,567 (2.99%)	78 (0.05%)	8 (0.005%)

One of the most commonly used SEU mitigation approaches is TMR. With this approach the user design is triplicated, i.e. to provide spatial redundancy, with each module operating in parallel. The outputs of these three identical modules are then fed into a voting circuit that arbitrates between them. If any one of the triplicated modules suffers from an SEU, the other two circuits outvote its output provided that they continue to operate normally. The correct data will therefore be passed on to the next stage of processing or to the outside world. Under such circumstance, TMR not only provides the means to identify a faulty unit, but also provides system availability albeit with reduced reliability. However, given the fact that each design must be triplicated, there is a considerable hardware overhead associated with the TMR approach. Power consumption is also increased and the operating frequency may need to be reduced. This may preclude the use of TMR for some applications with stringent area, speed or power constraints. In such situations it may be beneficial to apply TMR to only part of a target design [9] – [15]. Another issue associated with TMR is that the voters are also susceptible to SEU effects and are themselves single-points of failure. This limitation can be alleviated by triplicating the voters [16], using self-checking voters [17] – [19], or using more resilient, fault-tolerant voter designs [33]. TMR can either be inserted by the designer or by utilizing automated tools [9], [16].

TMR on its own does not provide a means of correcting a faulty unit. In general, a triplicated module comprises a datapath and a control path. When a triplicated module contains no feedback paths, SEUs are detected as a transient error in the output. However, when an SEU is trapped in a feedback path, or the configuration memory is affected, errors can persist, and some method for eliminating the effects of the SEU needs to be provided. A simple method for clearing the user circuit is to reset the system. However, a reset may delay the operation of time-critical systems too much. In [20] – [24], TMR is combined with *Partial Dynamic Reconfiguration* (PDR) to overwrite a corrupted configuration. One of the challenges of such an approach is how to re-synchronize the newly re-configured TMR module. One approach for dealing with this is presented in [23], who use checkpoint states to allow the sequential synchronization of the recovered module. The synchronization of the recovered module is performed while others are kept running by predicting a future state to which the system will converge. This technique can be applied to simple Finite State Machines, without complex cyclic structure, but is not applicable to general data processing components for which the future state cannot be predicted.

Our work aims to provide a general method that can be applied to any type of circuit so as to detect, reconfigure and re-synchronize the newly reconfigured component within a specified time period. Our method will not identify errors that cause two or more of the modules in a TMR system to fail simultaneously. This problem has been studied in [23].

One of the most commonly used techniques to correct SEU errors in the configuration memory of FPGAs is *scrubbing* [24]. In its simplest form, frames of the configuration memory are re-loaded i.e. scrubbed, periodically irrespective of SEU presence or absence. The on-chip hardware overhead associated with this approach is minimal, although a golden model of the configuration must be stored off-chip. This external memory needs to be protected as well to ensure it is not corrupted by SEUs. Given large memory sizes, it is not desirable to triplicate large memories with TMR to provide protection via spatial redundancy. Instead, *Error Detection And Correction* (EDAC) techniques are used to achieve this. With EDAC techniques, extra redundant information is added to the original data and it is encoded. When the data is retrieved from the memory, it goes through a decoding process and the redundant bits inserted during the encoding process help identify any bit-flips in data and eventually correct them. There are many different types of EDAC codes, such as Hamming and Reed-Solomon, each of them with different detection and correction capabilities as well as different computational complexity [25], [26]. EDAC approaches provide redundancy in the information domain.

Given the limited throughput associated with the SelectMAP interface and the ever-increasing FPGA sizes,

the amount of time required for scrubbing continues to increase. Furthermore, given the fact that scrubbing is applied periodically regardless of an SEU being present or not, it results in wasted power dissipation. These disadvantages may preclude the use of scrubbing for applications with stringent processing time and power consumption requirements. A further drawback of configuration memory scrubbing is that the method cannot detect errors in dynamic user-defined memories such as in flip-flops and RAM. Furthermore, if an SEU happens right after a scrub cycle, a full readback cycle, potentially taking hundreds of milliseconds depending on the FPGA size, passes before it is picked up and corrected.

4 Proposed SEU Detection and Mitigation Framework

Our approach considers the use of TMR to detect and mitigate transient errors in FPGA-based application circuits and focuses on techniques to minimize the overheads of using TMR, particularly when persistent errors are cleared using partial modular reconfiguration. In order, we are concerned with:

- Design correctness,
- Meeting application performance requirements,
- Providing maximal fault resilience, and
- Cost of implementation.

To enhance the likelihood of developing a correct method, we develop our approach from a consideration of correctly computing abstract *dataflow graphs* (DFGs). For our purposes, a DFG represents a computation that is to be implemented in a fault-tolerant manner. DFGs are composed of nodes and directed edges. DFG nodes represent operations on logic or the storage of (partial) results in registers or on-chip memory. DFG edges represent the transfer of data between, to, or from DFG nodes. DFGs may be cyclic, i.e. contain feedback paths, or acyclic, in which case data being processed by the DFG visits each node at most one time.

Our goal is to develop methods for inserting TMR voter logic into DFGs such that SEUs affecting user logic and configuration memory are detected, and that the delay in recovering from a fault is less than a specified maximum fault recovery time. This problem reduces to partitioning the DFG into components, each of which are implemented in triplicate. Each triplicated component has associated voting and control circuitry, which represents an area overhead, and therefore affects the size of FPGA needed and the operating frequency of the resulting design. We employ a strategy to minimize the number of partitions needed (so as to reduce the overheads caused by inserting the voters) while ensuring that a fault in any component can be detected and mitigated, possibly via partial reconfiguration, within the specified maximum fault recovery time. In our work, we have assumed that at most one fault occurs at any time and that no second fault occurs within the system while it is already recovering from one. The maximum fault recovery time should therefore be chosen to be less than the reciprocal of the maximum error rate.

TMR ensures transient faults in a single replicated module are detected and "overruled" by majority vote and that the faulty module can be identified. Persistent faults can also be detected by observing two or more successive errors in a single replicated module. In this case, with precise design, the faulty module can be reconfigured while the unaffected modules continue to operate. For such partial reconfiguration of an FPGA at runtime (dynamic reconfiguration) to be feasible, careful consideration needs to be given to the spatial layout of the replicated modules so as to minimize the time needed to complete the reconfiguration.

4.1 Acyclic Graphs

Let us examine the proposed approach with an example circuit. The simplest type of circuit comprises a linear sequence of registers, possibly including some combinational function between each register. This situation is depicted in Fig. 1 as a linear DFG, or pipeline, and represents a component, such as a filter, through which data is streamed without feedback. The circuit has one or more inputs, one or more outputs and contains no cycles.

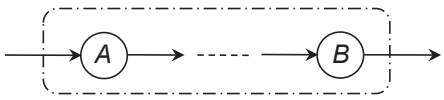

Figure 1. A linear chain of processing steps.

The entire circuit and its inputs can be triplicated and a majority voting circuit can be used to compare the corresponding outputs of each module. In the resulting TMR-protected circuit, a single transient error will clear itself after a single erroneous cycle since the data processed in the immediately following cycle won't have been affected by the SEU and there is no feedback path for the original error to be stored or circulated. A comparison of each module's output with the corresponding majority output can identify which, if any, module is in error. Two or more successive errors occurring in the one module in this type of circuit either indicates multiple SEUs occurring on successive clock cycles, or that some damage has been caused to the circuit's configuration. As the latter cause is significantly more likely in FPGA-based circuits, such an event can be used to trigger reconfiguration of the module.

Reconfiguring a triplicated module in a TMR system involves isolating the logic for the module, loading the partial bitstream for the configuration frames used by the module, reconnecting the reconfigured module to its source and sink ports, and waiting for the state of the module to be synchronized with the pair of modules that were not affected by the upset. The newly reconfigured module cannot be used for checking the correctness of

the other modules in the TMR system until it produces the correct outputs. When no feedback paths are present, it is easy to simply wait until the inputs to the newly reconfigured module have been completely processed by the circuit. A quicker method would require the state of internal registers to be copied from one of the other system modules. However, this approach is impractical because it requires a significant amount of control circuitry and wiring to implement.

The maximum time to detect an error occurs when an error presents itself in the input stage. If inputs require N clock cycles to be processed by the component, then the time to detect an error is at most $t_{D\_MAX} = N$ clock cycles. The time needed to reconfigure the module, t_R, is proportional to its size. This time includes some constant overheads as well as the time needed to read each frame from off-chip memory, check the frame contents for errors, and write it to the destination configuration memory. The configuration port on current devices is a fixed size (typically, 32 bits) and hence the time to reconfigure is proportional to the number of configuration memory frames spanned by the circuit. Including the time needed to re-synchronize the module, the time needed to detect and recover from a persistent error is at most $2t_{D\_MAX} + t_R$ clock cycles. The area of the module, the length of the pipeline and the clock frequency determine the actual fault recovery time. The actual fault recovery time can be reduced (increased) by partitioning the linear DFG component into shorter (longer) segments based on estimates of the area and latency of the cores implementing each node in the DFG.

4.2 Cyclic Graphs

If a DFG component contains cycles it is not possible to determine whether a persistent fault lies in the datapath registers of the configured circuit or in the configuration memory of the FPGA. Furthermore, if a module contains cycles, the correct state cannot be established simply by allowing inputs to flow through the newly reconfigured circuit. For this reason it is crucial that DFGs be partitioned so that all feedback edges are cut. Cut feedback edges give rise to an output from and/or an input to an otherwise acyclic component. That is, the feedback output is passed through the voter to determine whether it is correct or not, and feedback inputs are supplied from the checked outputs of voter circuits.

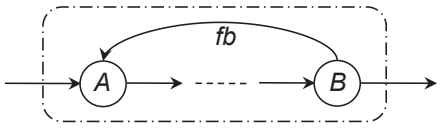

Figure 2. Cycle *AB* due to feedback edge *fb*.

In Fig. 2, a feedback edge, labelled *fb*, exists from node B to node A. This situation arises in components that include structures such as finite state machines or accumulators. In this case, the feedback edge is cut to produce an output that the majority voter in the TMR implementation of Fig. 3 needs to check. When a persistent error is detected, the circuit is reconfigured as previously described. When the reconfigured module is ready to recommence processing, the current *fb* value, as determined by the voter circuit, is provided to the module as an input. Note that this value is determined in the course of checking the feedback outputs of the other two functioning modules. In this case, the latency of the module also determines the delay until the correct state has been established in the module and until its outputs can be used to check the other replicas.

4.3 TMR and Reconfiguration Controller Design

All outputs of triplicated components are checked using a block of logic which includes a majority voting circuit, an error counter, a reconfiguration trigger, and a resynchronization counter. An illustration of the arrangement of this block relative to the modules of a TMR implementation is depicted in Fig. 3. The internal structure of the voter circuit and reconfiguration controller is depicted in Fig. 4.

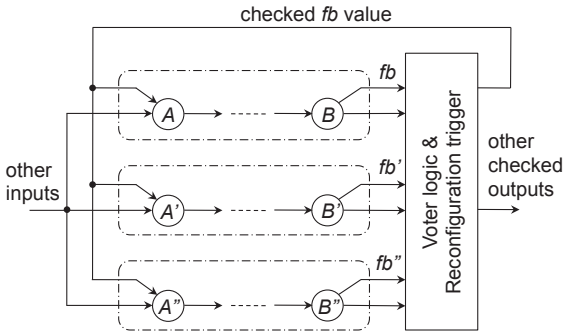

Figure 3. TMR implementation of cycle *AB* from Fig. 2.

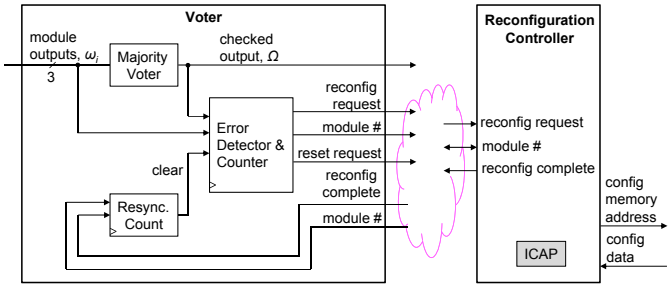

Figure 4. Internal structure of voter logic and interface with reconfiguration controller.

The voter logic, illustrated in Fig. 4, employs a standard technique to determine a checked output value, $\Omega = x \cdot y + y \cdot z + x \cdot z$, given three output values, $\omega_i = \{x, y, z\}$, produced by the individual modules of the TMR component. The checked output value is compared

with the output of each module to determine whether any individual module is producing an erroneous output: $error_i = \omega_i \oplus \Omega$. The number of successive erroneous outputs is counted to determine whether a persistent error is present. Typically, two successive errors indicate the presence of a persistent fault. However, under certain circumstances (see Section 4.4) it is possible for an acyclic component to produce more than one erroneous output in response to a transient fault.

When the voter circuit detects a persistent fault, it isolates the module by disconnecting its inputs and outputs and raises a reconfiguration request on a centralized reconfiguration controller module. The reconfiguration controller determines the location and size of the requested module's partial bitstream, fetches the configuration data, and writes the configuration data to an *Internal Configuration Access Port* (ICAP) of the FPGA. While the configuration data is transferred from off-chip memory to the ICAP, it is checked using error-correcting code words. The process of writing the data to the ICAP updates the configuration memory of the corresponding region of the FPGA. When the transfer has been completed, the reconfiguration controller signals the voter circuit that partial reconfiguration of the requested module has been completed.

The voter circuit reestablishes the connection of the reconfigured module to its inputs and outputs and initializes a down-counter to determine when the reconfigured module has re-established its state (been resynchronized). The countdown value is set to the latency in clock cycles of the longest path through the component. The error detector ignores the outputs of the newly reconfigured module until the down-counter has reached zero, indicating that the module has been resynchronized and all three modules are again fully functional as a TMR unit.

A centralized reconfiguration controller is specified since current Xilinx Virtex FPGAs only allow one of two possible ICAP ports to be used while the chip is active. As each voter circuit needs to be connected with the reconfiguration controller, a simple token ring network can be used to transmit multi-bit signals, such as the ID of the module that is to be reconfigured, to the reconfiguration controller. Messages exchanged between the blocks can be protected with parity bits and a timeout/resend protocol.

4.4 Partitioning the Graph

The methods described in Sections 4.1 and 4.2 can be applied to a DFG of any size. However, the fault detection time increases with the number of clock cycles needed to traverse the longest path from input to output (the latency of the component). The time to recover from a persistent fault by reconfiguring a module increases with the number of configuration frames spanned by the circuit.

More general acyclic DFG components can readily be protected using the above approach as shown in Fig. 5. In particular, multiple inputs, such as those represented by edges a and b, and multiple outputs, such as those represented by edges c and d, are easily dealt with. The only requirement is that the majority voting circuit checks all outputs and that the triplicated inputs arrive at each replicated module in the same cycle. It is also necessary to ensure that the longest path (in terms of clock cycles) for signal passing through the component is used to determine when the newly reconfigured module has been re-synchronized.

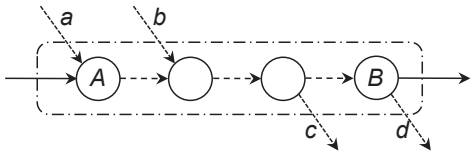

Figure 5. Linear DFG component with multiple inputs and outputs.

The internal structure of an acyclic DFG component does not affect the correctness of the method. For example, consider Fig. 6, in which there are multiple paths through the component. The sub-component resulting in output a, the sub-component commencing with input b and intermediate nodes such as node c may be present. These have the effect of increasing the area and latency of the module, but do not affect the ability of the voter to check the outputs to determine whether or not the module is faulty. However, if the length of the path involving edge a differs from the length of the path involving node c and a fault occurs upstream of both paths, then it is possible for a transient fault to generate multiple erroneous outputs. The maximum number of error cycles over which a transient error can be detected at the outputs determines the minimum number of cycles the voter needs to check for errors before it determines the error is persistent and triggers a reconfiguration of the module.

Partitioning a DFG to meet a specified maximum fault recovery time is achieved by estimating the number of frames needed to configure the circuit. The actual

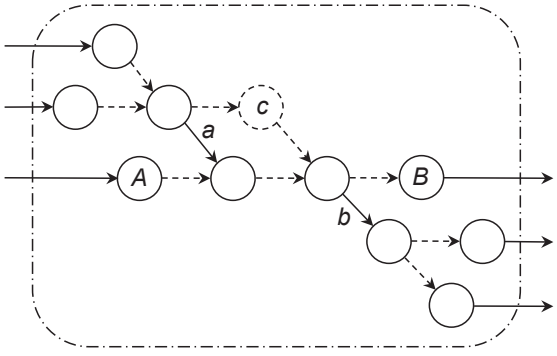

Figure 6. General acyclic DFG structure.

number of frames needed is not known until the configuration bitstream is produced, which is too late in the design flow to partition the circuit into components, replicate them and insert voter logic.

Current design tools such as PlanAhead, allow the layout of a design to be managed and currently must be used to implement a partially reconfigurable design. Reconfigurable modules need to have bounding boxes specified for them. However, using PlanAhead for laying out and partitioning a DFG becomes tedious and error-prone as the number of TMR components grows. It is therefore desirable to provide a tool for automatically partitioning a DFG and inserting the additional control logic. Several such tools exist [9], [15], [16] but it is not clear that they suit our methodology. The primary focus of XTMR [16] is to automatically insert voter logic into fully protected circuits. As it is not designed to achieve a specified maximum fault recovery time, it does not constrain the layout of the TMR modules, nor does it include automatic generation of the control needed to trigger and manage reconfiguration and to re-synchronize a reconfigured module. The BLTmr [9], [15] tool is focused on reducing the overheads of TMR voter insertion by defining a hierarchy of components that require TMR protection. Other methods focus on reducing the impact on critical path delay and the speed of circuits that have been given the TMR treatment. Our method focuses instead on maximizing reliability by guaranteeing a maximum fault recovery time and on availability by protecting all components using TMR.

Partitioning is only feasible when estimates of the area and latency of DFG nodes are known. High-level specifications can make use of information provided about IP blocks used. For example, the Xilinx CORE Generator provides information about the area and latency of its cores. Alternatively, the synthesized and technology-mapped netlists can be analyzed and partitioned. The analysis needs to explore the DFG in a breadth-first manner as illustrated in Fig. 7.

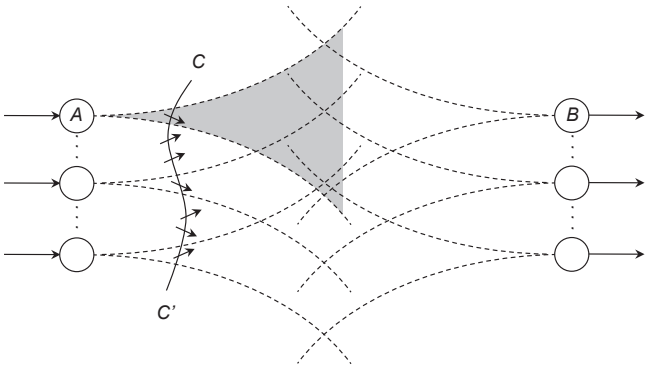

Figure 7. Partitioning a DFG.

The breadth-first DFG search commences with the input nodes. Each of these forms the source of a cone of logic that is affected by a change in the input. See, for example, the shaded cone associated with node A depicted in Fig. 7. Each output of the graph is also associated with a cone of nodes involved in its computation. The graph is searched by extending the wavefront CC' of nodes included in the current partition commencing with the inputs. The latency and area of the nodes included in the current partition are iteratively updated. When the size of either of these parameters causes the estimated maximum delay expression $2t_{D\_MAX}+t_R$ to be exceeded, the partition is finalized, replicated and the TMR voter and control logic is inserted. The graph exploration then continues with the outputs of the previous partition, a subset of which form the inputs of the next partition.

As explained in Section 4.2, all feedback edges need to be cut so as to eliminate cycles. Doing so ensures all feedback values are checked and the correct values are provided as input, even to newly reconfigured modules, whose state has not yet been re-established. If feedback edges are converted into an output-input pair, any arrangement of cycles, including nested cycles and interlinked cycles, as illustrated in Fig. 8 can be handled within our partitioning methodology. It is not clear that previously described tools are similarly guaranteed to produce correct TMR implementations.

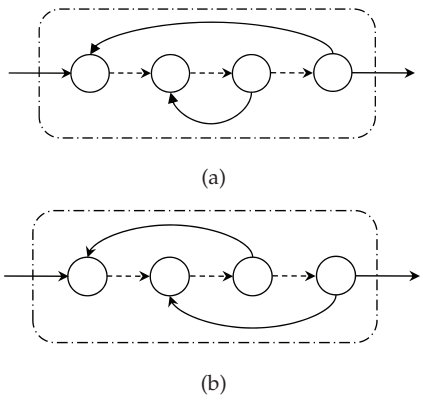

Figure 8. (a) Nested and (b) interlinked DFG cycles.

5 Evaluation

Our evaluation was aimed at proving the ideas outlined in Sections 3 and 4. We focused our attention on implementing a variety of *Finite Impulse Response* (FIR) filter circuits and a *Block Adaptive Quantizer* (BAQ) circuit as used in the compression of *Synthetic Aperture Radar* (SAR) data that are to be transmitted from satellites to Earth. These circuits were chosen to enable us to assess the additional area, the decrease in clock frequency, and the reconfiguration delay that result when representative circuits from the signal processing domain are implemented using our resynchronization technique. Moreover, we aimed to explore the different approaches needed for mapping cyclic versus acyclic components.

Our implementations have targeted Xilinx Virtex-5 XC5VFX70T devices, as found on the ML507 prototyping

board from Xilinx. We have used the ISE 12.4 design environment for entering and synthesizing our designs and used the PlanAhead tool in order to floorplan the circuits we studied.

5.1 Finite Impulse Response Filter

We studied a 21-tap *Finite Impulse Response* (FIR) filter because it offered the opportunity to study both cyclic and acyclic implementations. An n-tap FIR filter can be implemented using any number of accumulators from 1 to n. Fig. 9 illustrates a cyclic (temporal) design using just a single accumulator. The Xilinx CORE generator, when provided with appropriate parameters, automatically generates the design illustrated in this figure. Note that this design feeds the accumulated sum of the delayed inputs (buffered in the Data Storage block) and scaled by the corresponding coefficients (stored in the Coefficient Storage block) back to an adder. For an n-tap filter with a single accumulator, a filter output value is produced n clock cycles after each sample is input. The contents of the Data Storage block are then shifted by one delay unit and a new input can be processed. Not illustrated in the figure is a finite state machine including a counter that controls the operation of the filter.

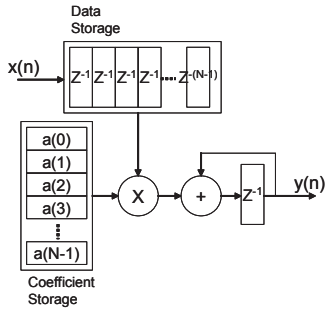

Figure 9. A single accumulator FIR engine block diagram as implemented by Xilinx CORE generator [34].

In our study we compared the implementation obtained using Xilinx CORE generator with a hand-coded implementation based on the same design. The reason we did this was that we do not yet have the tools to explore a circuit netlist in order to break feedback paths such as those found in the accumulator and the controller of the single accumulator FIR design. In contrast, by hand-coding the design we were able to route these signals out of the component and provide the feedback via an external connection. As discussed in Section 4.2, using our method it would not be possible to resynchronize a reconfigured copy of the single accumulator FIR design without extracting the feedback paths in this way. The results of mapping the generated and hand-coded designs to our FPGA are reported in Table 3 and discussed below.

An FIR filter can also be implemented as an acyclic (spatial) design, as illustrated in Fig. 10. Xilinx CORE generator can be directed to utilize as many accumulators in the transposed direct-form as there are taps in the filter. In this case, one sample input is consumed and one filter output is produced during each clock cycle. While such a design uses more resources, the lack of feedback paths and not needing a controller can significantly improve the clock frequency and throughput of the design. We again compared the implementation obtained using Xilinx CORE generator with a hand-coded implementation based on the same design in order to rule out the possibility that the CORE design has an internal feedback path that would affect resynchronization after reconfiguration. The results of mapping the generated and hand-coded designs to our FPGA are reported in Table 3 and compared with the results obtained for the single accumulator FIR designs.

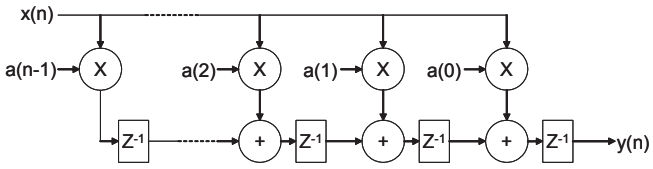

Figure 10. Transposed Direct-Form FIR engine block diagram as implemented by Xilinx CORE generator [34].

Table 3 reports the resource utilization, clock frequency and throughput (free and constrained placement), partial bitstream size, and component latency for a number of single accumulator and transposed FIR designs. All designs have 16-bit datapaths. Only those designs that are reconfigurable have a bitstream size or latency recorded for them. Since our triplicated designs were implemented using constrained placement of the module logic (to facilitate eventual reconfiguration), unconstrained placement frequency and throughput results were not recorded for these designs.

The design referred to as firMAC is the single accumulator design obtained using Xilinx COREgen. We refer to this architecture as MAC. Using a placement strategy that balances optimization for speed and area, the place and route tools used 157 *Look-up Tables* (LUTs), 163 *Flip-Flops*(FFs), and 1 embedded *Digital Signal Processing* (DSP) block. A free (unconstrained) placement resulted in an operating frequency of 333 MHz, and given 21 cycles/sample, this corresponds to a throughput of 15.9 Msamples/s. It should be noted that we were able to achieve a clock frequency of 459 MHz (38% improvement) by constraining the placement to a rectangular region just large enough to provide the necessary resources.

In comparison, our hand-coded design with internal feedback signals, named gfirMAC, was found to operate at 126 MHz, despite the smaller utilization, when the place and route tools used a balanced optimization strategy and the placement was unconstrained. Constraining the placement improved the maximum clock frequency

Table 3
Results of mapping 21-tap single accumulator and transposed direct-form FIR filter designs to Xilinx XC5VFX70T device.

Design Name	Design Entry	Arch	Utilization			Frequency (MHz)/ Throughput (Msps)		Bitstream Size (W)	Latency (CC)
			LUT	FF	DSP	Free	Constrained		
firMAC	coregen	MAC	157	163	1	333/15.9	459/21.9		
gfirMAC	HDL	MAC	55	10	1	126/6.0	163/7.8		
gfirMAC-FB	HDL	MAC	57	10	1	126/6.0	160/7.6		
gfir3MAC-FB	HDL	MAC	242	24	3		131/6.2		
gfir3MAC-FB-dr	HDL	MAC	911	200	3		107/5.0	4,100	441
firTF	coregen	TF	39	90	21	206/206	342/342		
fir3TF	HDL + coregen	TF	129	249	63		242/242		
fir3TF-dr	HDL + coregen	TF	528	425	63		155/155	6,396	21
gfirTF	HDL	TF	14	18	20	447/447	500/500		
gfir3TF	HDL	TF	75	54	60		252/252		
gfir3TF-dr	HDL	TF	498	230	60		172/172	6,396	21

by 29%, and extracting the feedback signals for design gfirMAC-FB, did not affect this design's performance. We found that the freely placed, hand-coded design suffered a 62% performance penalty with respect to the freely placed CORE design. The greater amount of resources used by the CORE design, particularly of flip-flops, suggests some of the better performance may be due to internal critical path optimisation.

We used gfirMAC-FB as our component module in a conventional TMR reference system named gfir3MAC-FB. In this implementation, 3 copies of gfirMAC-FB were constrained to rectangular regions of the FPGA large enough to contain each copy and suitably placed for reconfiguration, the voter logic was also added, but the proxy pins[2] needed to be able to reconfigure the regions and the reconfiguration controller were not implemented. This design can detect and recover from transient errors but cannot be reconfigured to eliminate a permanent error. As can be seen, the LUT usage increased almost 5-fold, the FF utilization increased 2.5X, and 3 DSP blocks were used instead of one. Compared to the constrained gfirMAC-FB design, a slowdown to 131 MHz (18% decrease) was observed. The critical path in this implementation was found to be from the counter of the filter's controller via the external feedback port to the voter and back into the filter to select the next filter coefficient. Adding the reconfiguration controller and proxy pins (module interfaces) to create the reconfigurable design named gfir3MAC-FB-dr increased the number of LUTs used another 4 times, the number of FFs used 8X, and slowed the design down to 107 MHz (a decrease of 33% from the constrained single gfirMAC-FB component, but only 18% slowdown relative to the conventional TMR design). The critical path was found to be the same as before but with the delay of the proxy pins on leaving and re-entering the filters added to the critical path delay.

The resulting partial bitstream for each triplicated module was found to be 4,100 32-bit words in length. This corresponds to a reconfiguration delay of 41 μs when the ICAP is clocked at its maximum clock frequency of 100 MHz. The latency of the component, corresponding to the maximum time needed for a TMR module to be resynchronized, is 441 clock cycles since it will take 21 clock cycles per sample input and the Data Storage buffer won't be refilled and synchronized with its TMR partners until the 21st sample is input after the reconfigured module is restarted. For this design, the time to first detect an error does not take more than 21 clock cycles. For this circuit, the resynchronization delay therefore comprises approximately 10% of the reconfiguration delay.

In summary, the overheads of triplicating the cyclic, single accumulator FIR design and adding the capacity to reconfigure and resynchronize a single module at a time resulted in a slowdown of 33% relative to a standalone, constrained implementation of just one module. Just triplicating the design, slowed it down by 18%. Furthermore, we observed a 20-fold increase in each of the resources needed, except that we only used 3X as many DSP blocks. Relative to a constrained CORE generated design, the increase in resource usage was more modest, but the performance penalty was 77%.

Our results for the transposed direct-form designs differ from those of the single accumulator designs in the following ways. The Xilinx CORE generated design, named firTF, had a considerably smaller area than

2. A proxy pin is the means for attaching a reconfigurable wire to the static part of a design. Each wire connecting a reconfigurable module uses one proxy pin, which consumes one LUT.

firMAC except that it used 21 DSP blocks instead of just 1. An unconstrained placement using a balanced optimization strategy achieved a maximum clock frequency of 206 MHz. Constraining the design resulted in a frequency of 342 MHz (66% improvement). The conventional TMR implementation obtained by triplicating this design but not implementing the proxy pins or reconfiguration controller needed to be able to reconfigure the individual modules (design fir3TF) increased the utilization 2–3X, and slowed the design down to a clock frequency of 242 MHz (29% reduction from the constrained firTF design). Enabling reconfigurability by adding the proxy pins and reconfiguration controller (fir3TF-dr) resulted in a further 2–4X increase in resource utilization and slowed the design down to 155 MHz (55% reduction from the constrained firTF design; 36% reduction from the conventional TMR design).

We also tried implementing a hand-coded transposed filter design (gfirTF) and found our implementation could be clocked at 447 MHz. We suspect the improvement was at least partially due to hardcoding the coefficients into the design that then allowed the individual bits to be tied to VCC or ground. Implementing the conventional TMR design (gfir3TF) slowed the design down to 252 MHz (a reduction of 50%). The critical path was found to be from the filter output to the error detector of the voter logic. Adding the proxy pins and reconfiguration controller to obtain the dynamically reconfigurable gfir3TF-dr design, reduced the clock frequency further to 172 MHz (66% reduction from the single constrained design; 32% reduction in speed from the conventional TMR design). The critical path was now found to be from the voter to the reconfiguration controller (partial bitstream address tables). Both reconfigurable triplicated TF designs (fir3TF-dr and gfir3TF-dr) used similarly sized reconfigurable regions with partial bitstream sizes of 6,396 words. These result in a reconfiguration delay of 64 μs. The resynchronization delay for this design is the same as the error detection delay, which is 21 clock cycles – corresponding to the number of cycles needed for a problem near the input port of a module to flow through to the output port.

In summary, we found that applying our method to a CORE generated acyclic transposed FIR design resulted in a 55% slowdown and a 3–4X increase in resource utilization. A hand-coded acyclic design was slowed down 66% and resource utilization was increased by up to 34-fold (LUTs). Adding reconfigurability to conventional TMR implementations of a transposed 21-tap FIR filter contributed between 32% and 36% slowdown and 4–7X increase in resource usage.

Some additional points should be noted. The device we targeted is a medium-sized Virtex-5 device; it has 44,800 LUTs+FFs and 128 DSP blocks in total. The Virtex-5 device family ranges in size from 12,480 LUTs+FFs and 24 DSP blocks to 207,360 LUTs+FFs and as many as 4,200 DSP blocks. The increases in resources used by our reconfigurable designs appear large since the designs being triplicated and reconfigured are relatively small, but in comparison to large devices, the resource utilizations are comparatively small. Furthermore, our hand-coded designs were unoptimized "first cut" designs. We found it would have been possible to improve the performance of our designs quite substantially. For example, we found we would have been able to improve the performance of gfirMAC-FB by adding pipeline registers to the output and to the feedback signal used in selecting the next coefficient. This change to the base design would have resulted in a constrained frequency of 173 MHz and a gfir3MAC-FB-dr frequency of 154MHz (44% improvement over the first cut, and just 11% worse than the single instance of the pipelined design). This improvement would have been possible for a modest increase in size — just 3 additional LUTs and 39 additional FFs for the pipelined reconfigurable version compared to the "first cut" we reported upon — for just one additional cycle of latency. Lastly, it should be noted that current practical satellite-based systems (such as those used to gather *Synthetic Aperture Radar* (SAR) images typically require throughputs in the range of 50-200 Msample/s. A common technique to deal with high data rates is to use a polyphase approach to filtering in which the ADC output is shared between several filter instances, each of which has inputs at a lower sample rate. We therefore conclude that the performance we observed for the circuits we examined is acceptable.

5.2 Block Adaptive Quantizer

Spaceborne SAR is used for the remote sensing of the Earth's surface, and is capable of producing high-resolution maps over large areas in short periods of time — typically a 25 km wide strip that is extended at 7 km every second in the direction of the satellite's flight path. This allows large areas to be imaged in a short period of time, but also presents considerable challenges with respect to storing and communicating the SAR data to the ground via a limited capacity channel.

The most commonly used technique for on-board SAR raw data compression is the Block Adaptive Quantization (BAQ) algorithm [35] – [38]. The BAQ algorithm is based on the observation that SAR data can be modelled as a normally distributed random variable with a slowly varying standard deviation [35]. With the BAQ approach, SAR data (both the In-phase and Quadrature channels) are divided into blocks of size N and statistics of the blocks e.g. standard deviation (σ), are estimated. The standard deviation of each block is then used to adapt the Max-Lloyd quantizer [36] threshold values block-by-block to optimally quantize the raw SAR data, which is originally m-bits wide, using fewer n-bits — typically 2, 3 or 4-bits. The quantized signal and the standard deviation of each block are combined and transmitted to the ground station for data-decoding. A block diagram of the BAQ algorithm is depicted in Figure 11.

The block size, N, is a trade-off between the statistics of the incoming signal and the variation of the signal

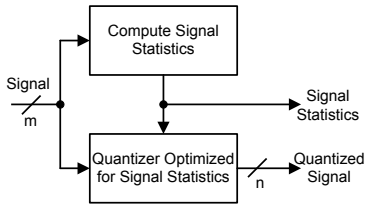

Figure 11. BAQ Algorithm.

power. For our design, N was chosen to be 256 samples. Furthermore, m is 8-bits wide with n was set to 3-bits. Figure 12 depicts the architecture used to implement the BAQ algorithm.

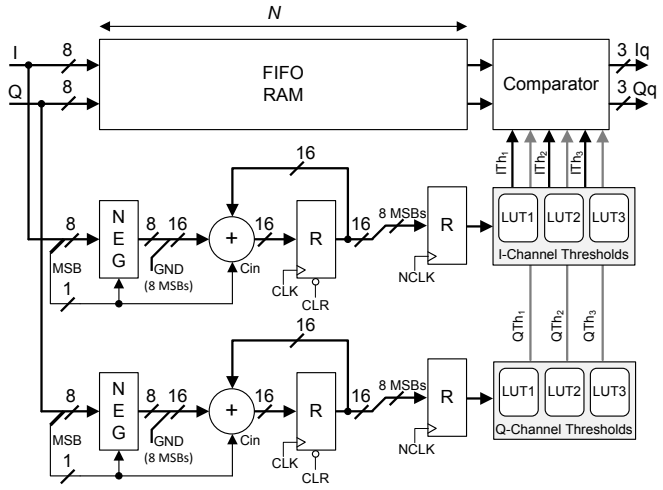

Figure 12. BAQ Architecture.

During the implementation phase, the standard deviation calculation was replaced with an average magnitude calculation, $|\overline{I}|$, $|\overline{Q}|$, to eliminate the need for a square root operation [37], [38]. Hence, in the implemented design, the quantization threshold values, *ITh/QTh*, are determined by calculating the average magnitude of a block of N raw SAR data samples and using this to address threshold Look-Up Tables (LUTs). Subsequently, a comparator maps the raw SAR data to 3-bit outputs, *Iq/Qq*, by comparing them to these thresholds. For n = 3-bits, the threshold values are $0, 0.5006\sigma, 1.050\sigma$ and 1.748σ, where σ is the block standard deviation [36]. The design consists of only 3 LUTs per I/Q-channel since the first threshold is zero. The design takes 256 cycles ($N = 256$) to calculate $|\overline{I}|$ and $|\overline{Q}|$ values for a particular block and a further 256 cycles are needed to realise the quantization. These two operations can be overlapped for successive blocks of N samples to obtain a continuous output stream.

The results of the BAQ implementations are recorded in Table 4. The first set of 3 designs (first 3 rows) used FPGA LUT resources to implement the threshold memory whereas the remaining ones used Block RAMs to do so. The FIFO-RAM, which buffers 256 samples ($N = 256$), was implemented using distributed RAM (LUTs) in all cases. All the designs reported upon in Table 4 were hand-coded using Verilog.

The single LUT-based BAQ design, baqLUT, used 386 LUTs and 145 FFs for its implementation. The maximum clock frequency for an unconstrained placement was found to be 251 MHz, whereas a constrained placement yielded a slightly higher clock frequency of 264 MHz. The critical path was found to be in the combinational logic that compares the FIFO RAM outputs with the threshold values to obtain the quantized results. Since the BAQ designs produce one quantized I/Q pair of data each clock cycle, their throughputs are the same as their operating frequencies. It should be noted that all the designs were optimized for area with a view to reducing the size of the partial bitstreams needed for the partially reconfigurable implementations. Additionally, it should be noted that the reconfigurable modules of the LUT-based designs were placed into reconfigurable regions that span two, non-contiguous bounding boxes in order to minimize the partial bitstream sizes. This was a consequence of the relatively high proportion of LUTs used as shift registers and distributed RAM, and that these are only available in half of the slices (CLBs) of the target device.

The conventional TMR implementation, baq3LUT, increased the resource utilization by a factor of 3X, and slowed the design down to 152 MHz (a 42% reduction from the constrained baqLUT design). The critical path for this design was found to start at the BAQ control logic and loop back into the BAQ to update the threshold values via the external feedback port and the voter logic. Adding the configuration controller and proxy pins to create the reconfigurable design baq3LUT-dr used about 50% additional resources and reduced the maximum clock frequency to 125MHz (51% reduction from the constrained baqLUT design; 15% reduction from the conventional TMR design). The critical path for this design was found to be the same as for the TMR design, baq3LUT, except that the newly inserted proxy pins further reduced the performance.

As the LUT-based BAQ modules were optimized for area, the generated partial bitstreams were only 5,904 words large. This was found to be significantly smaller than for the BRAM-based BAQ modules, which were 12,382 words large (see the baqBRAM-dr design). However, because the placement of the TMR and partially reconfigurable designs were split into two separate regions per module, the more dispersed (sparse) placement resulted in a significant increase in critical path length for feedback wires that had to extend to the voter logic and back to the BAQ threshold unit. A more judicious placement of the logic and careful buffering of feedback signals may be required to reduce the performance penalty we observed.

The BRAM-based single BAQ design, baqBRAM, used

Table 4
Results of mapping LUT- and BRAM-based BAQ designs to Xilinx XC5VFX70T device.

Design Name	Design Entry	Utilization		Frequency (MHz)/ Throughput (Msps)		Bitstream Size (W)	Latency (CC)
		LUT	FF	Free	Constrained		
baqLUT	HDL	386	145	251/251	264/264		
baq3LUT	HDL	1,192	435		152/152		
baq3LUT-dr	HDL	1,700	611		129/129	5,904	512
baqBRAM	HDL	287	63	174/174	178/178		
baq3BRAM	HDL	889	189		149/149		
baq3BRAM-dr	HDL	1,397	365		143/143	12,382	512

287 LUTs and 63 FFs. The clock frequencies for the unconstrained and constrained placements were found to be 174 MHz and 178 MHz respectively. The conventional TMR implementation (baq3BRAM) again increased the utilization 3X, and slowed the design down to 149 MHz (16% reduction from the constrained baqBRAM design). Adding the configuration controller and proxy pins to create the reconfigurable design, baq3BRAM-dr, used about 60% more resource and reduced the maximum frequency to 143MHz (20% reduction from the constrained baqBRAM design; 5% reduction from the conventional TMR design). As the BRAM-based BAQ designs were constrained to contiguous areas on chip, the slowdown from adding the voter and proxy pins was not as high as for the LUT-based design. The resulting partial bitstreams were 12,382 words large and took 124 μs to configure at the reconfiguration clock frequency of 100 MHz. It should be noted that configuring a BRAM column involves configuring both the BRAM interconnection logic and the BRAM content, and the generated bitstreams are therefore much larger than for the LUT-based BAQ design.

The time to resynchronize a module after reconfiguration is 512 clock cycles at the operating frequency since it may be necessary to process 2 complete sample blocks to resynchronize the sample buffer and produce valid outputs.

5.3 Summary

Figure 13 presents a bar chart of the resource usage and clock frequency of the single module and dynamically reconfigurable (DR) triplicated implementations compared with the area and speed of the conventional TMR implementations of the hand-coded circuits we studied.

In general, the results confirm that triplicating a single module design to create a conventional TMR implementation results in a tripling of the resource usage and some slowdown in clock frequency. For the circuits we studied we found a further increase in LUT usage that ranged as high as 6.6X when the TMR implementation was modified to support dynamic reconfiguration and rapid resynchronization. This increase arises from a relatively constant overhead due to the reconfiguration controller

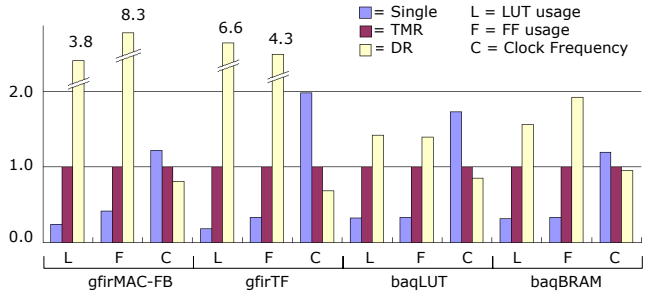

Figure 13. Area and speed of studied circuits relative to a conventional TMR implementation.

($\approx$ 270 LUTs for the FIR designs and 233 LUTs for the BAQ designs) and additional LUTs used due to the proxy pins for each wire that connects a reconfigurable module to the static part of the circuit. This overhead is therefore more significant for smaller circuits with high IO requirements e.g. the Finite Impulse Response circuits we studied. The increase in FF usage was simply due to the constant overhead of the reconfiguration controller (176 FFs), which is also relatively more significant when the circuit being triplicated has few flip-flops, but in overall terms is quite small.

The decreases in clock frequency observed when the conventional TMR implementations were adapted for dynamic reconfiguration and rapid resynchronization are due to increases in critical path lengths. In some cases, these increases are due to the additional delays of proxy pins on module outputs or feedback edges (gfirMAC-FB & baqLUT), and in other cases they are due to paths inside the reconfiguration controller we used (gfirTF & baqBRAM). Since the ICAP port can only be clocked at 100 MHz, and is therefore operated using a separate clock net, this part of the implementation should be excluded when considering the impact of our method on the speed of the application circuit. However, in the circuits we studied, we found that the slowdown due to the extended path delays of edges crossing the boundaries of reconfigurable regions was of a similar magnitude to that caused by the reconfiguration controller itself.

We found that further study is necessary to more precisely gauge the maximum possible performance of circuits implemented using our method and to identify the exact causes of slowdown.

6 Concluding Remarks

In this paper, we have presented a general technique for ensuring high reliability and high availability of FPGA-based systems subject to radiation-induced SEUs. Our method is based on the popular TMR technique and exploits partial reconfiguration at run time to restore the corrupted implementation of a TMR module. We have developed our method so that the system can fully recover from any single fault within a specified maximum fault recovery time. This includes the time to detect, reconfigure and resynchronize the state of a faulty module. We have shown that our method can be applied to any digital circuit. We have particularly focused our attention on correctly protecting and being able to correctly restore the state of circuit components containing feedback paths including nested and interlinked cycles.

Performance evaluations of applying our proposed approach to a selection of representative signal-processing circuits demonstrate that overheads associated with using the proposed approach are acceptable for earth observation applications. While the relative overheads in area can be substantial for small circuits, as circuit sizes grow, and compared with common FPGA device sizes, these overheads are small. The decrease in circuit speed appears to be significantly influenced by circuit design and floorplan. Further investigation will be undertaken to try to identify the common causes of critical path delay and how best to mitigate these, for example, by buffering feedback edges and automating the layout of critical regions.

Further study is needed to assess the feasibility, cost and overheads of partitioning circuits based on estimates of the area and latency of operations that have been represented in this paper by abstract DFG nodes. One approach we are pursuing is to modify the open source VPR tool [39] to assess the method with benchmark circuits. Our results indicate that partitioning choices have a marked impact on the clock frequency, reconfiguration, and resynchronization delays of the circuits being protected. We aim to investigate this aspect with a view to optimizing a methodology based on our technique.

References

[1] OECD, *The Space Economy at a Glance 2011*, Jul 2011, ISBN:9789264084643.

[2] R. Trautner, "ESA's roadmap for next generation payload data processors," in *Data Systems in Aerospace Conference (DASIA)*, May 2011.

[3] European Space Agency (ESA), *Next Generation Processor for On-Board Payload Data Processing Application*, Oct 2007, EDP/2007.35/RT.

[4] P. J. Pingree, "Advancing NASA's on-board processing capabilities with reconfigurable fpga technologies," in *Aerospace Technologies Advancements*, T. T. Arif, Ed. InTech, Jan 2010, ch. 5, pp. 69–86.

[5] N. Rollins, M. Wirthlin, M. Caffrey, and P. Graham, "Evaluating TMR techniques in the presence of single event upsets," in *Military and Aerospace Programmable Logic Devices Conference (MAPLD)*, Sep 2003.

[6] C. Carmichael, *Triple Module Redundancy Design Techniques for Virtex FPGAs*, Jul 2006, XAPP197 (v1.0.1).

[7] A. E. Bezerra, F. Vargas, and P. M. Gough, "Improving reconfigurable systems reliability by combining periodical test and redundancy techniques: A case study," *Journal of Electronic Testing*, vol. 17, no. 2, pp. 163–174, 2001.

[8] L. Sterpone and M. Violante, "A new reliability-oriented place and route algorithm for SRAM-based FPGAs," *IEEE Transactions on Computers*, vol. 55, no. 6, pp. 732–744, Jun 2006.

[9] B. Pratt, M. Caffrey, F. J. Carroll, P. Graham, K. Morgan, and M. Wirthlin, "Fine-Grain SEU Mitigation for FPGAs using partial TMR," *IEEE Transactions on Nuclear Science*, vol. 55, no. 4, pp. 2274–2280, Aug 2008.

[10] L. F. Kastensmidt, G. Neuberger, F. R. Hentschke, L. Carro, and R. Reis, "Designing fault-tolerant techniques for SRAM-based FPGAs," *IEEE Design and Test of Computers*, vol. 21, no. 6, pp. 552–562, Dec 2004.

[11] K. P. Samudrala, J. Ramos, and S. Katkoori, "Selective Triple Modular Redundancy (STMR) based Single-Event Upset (SEU) tolerant synthesis for FPGAs," *IEEE Transactions on Nuclear Science*, vol. 51, no. 5, pp. 2957–2969, Oct 2004.

[12] V. Chandrasekhar, N. S. Mahammad, V. Muralidaran, and V. Kamakoti, "Reduced Triple Modular Redundancy for tolerating SEUs in SRAM-based FPGAs," in *Military and Aerospace Programmable Logic Devices Conference (MAPLD)*, 2005.

[13] K. Morgan, M. Caffrey, P. Graham, E. Johnson, B. Pratt, and M. Wirthlin, "SEU-induced persistent error propagation in FPGAs," *IEEE Transactions on Nuclear Science*, vol. 52, no. 6, pp. 2438–2445, Dec 2005.

[14] L. F. Kastensmidt, L. Sterpone, L. Carro, and R. M. S, "On the optimal design of Triple Modular Redundancy logic for SRAM-based FPGAs," in *Design, Automation and Test in Europe*, vol. 2, Mar 2005, pp. 1290–1295.

[15] E. Johnson and M. Wirthlin, "Voter insertion algorithms for FPGA designs using Triple Modular Redundancy," in *18th annual ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA'10)*, 2010, pp. 249–258.

[16] (2009) Xilinx TMRTool product brief.

[17] N. Gaitanis, "The design of totally self-checking TMR fault-tolerant systems," *IEEE Transactions on Computers*, vol. 37, no. 11, pp. 1450–1454, Nov 1988.

[18] S. D'Angelo, C. Metra, S. Pastore, A. Pogutz, and R. G.Sechi, "Fault-tolerant voting mechanism and recovery scheme for TMR FPGA-based systems," in *IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems*, Nov 1998, pp. 233–240.

[19] S. D'Angelo, C. Metra, and R. G.Sechi, "Transient and permanent fault diagnosis for FPGA-based TMR systems," in *IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT'99)*, Nov 1999, pp. 330–338.

[20] C. Bolchini, A. Miele, and D. M. Santambrogio, "TMR and partial dynamic reconfiguration to mitigate SEU faults in FPGAs," in *IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT'07)*, Sep 2007, pp. 87–95.

[21] M. Straka, J. Kastil, and Z. Kotasek, "Modern fault tolerant architectures based on partial dynamic reconfiguration in FPGAs," in *IEEE 13th International Symposium on Design and Diagnostics of Electronic Circuits and Systems (DDECS)*, Apr 2010, pp. 173–176.

[22] ——, "Fault tolerant structure for SRAM-based FPGA via Partial Dynamic Reconfiguration," in *13th Euromicro Conference on Digital System Design: Architectures, Methods and Tools (DSD)*, Sep 2010, pp. 365–372.

[23] C. Pilotto, J. R. Azambuja, and L. F. Kastensmidt, "Synchronizing triple modular redundant designs in dynamic partial reconfiguration applications," in *ACM 21st annual symposium on Integrated circuits and system design (SBCCI'08)*, 2008, pp. 199–204.

[24] C. Carmichael, M. Caffrey, and A. Salazar, *Correcting single-event upsets through Virtex partial configuration*, Jun 2000, XAPP216 (vl.0).

[25] R. Hentschke, F. Marques, L. F. Kastensmidt, L. Carro, A. Susin, and R. Reis, "Analyzing area and performance penalty of protecting different digital modules with Hamming code and Triple Modular Redundancy," in *ACM 15th annual symposium on Integrated circuits and system design (SBCCI'02)*, 2002, pp. 95–100.

[26] O. Ruano, P. Reviriego, and A. J. Maestro, "A new EDAC technique against soft errors based on pulse detectors," in *IEEE International Symposium on Industrial Electronics*, Jun 2008, pp. 2293–2298.
[27] "Increasing design functionality with partial and dynamic reconfiguration in 28-nm FPGAs," White Paper, Altera, jul 2010.
[28] Xilinx Inc., *Virtex-4 FPGA User Guide*, dec 2008, UG070.
[29] Xilinx Virtex device datasheets. [Online]. Available: http://www.xilinx.com/support
[30] D. J. Engel, K. Morgan, M. Wirthlin, and P. Graham, "Predicting on-orbit static single event upset rates in Xilinx Virtex FPGAs," Los Alamos National Laboratory, Tech. Rep. LA-UR-06-8178, Nov 2006.
[31] H. Quinn, P. G. J. Krone, M. Caffrey, and S. Rezgui, "Radiation-induced multi-bit upsets in SRAM-based FPGAs," *IEEE Transactions on Nuclear Science*, vol. 52, no. 6, pp. 2455–2461, Dec 2005.
[32] H. Quinn, K. Morgan, P. Graham, J. Krone, M. Caffrey, and K. Lundgreen, "Domain crossing errors: Limitations on single device Triple-Modular Redundancy circuits in Xilinx FPGAs," *IEEE Transactions on Nuclear Science*, vol. 54, no. 6, pp. 2037–2043, dec 2007.
[33] T. Ban and L. de Barros Naviner, "A simple fault-tolerant digital voter circuit in TMR nanoarchitectures," in *8th IEEE International NEWCAS Conference (NEWCAS)*, June 2010, pp. 269 –272.
[34] Xilinx Inc., *LogiCORE IP FIR Compiler v5.0*, Apr 2010, DS534.
[35] U. Benz, K. Strodl, and A. Moreira, "A comparison of several algorithms for SAR raw data compression," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 33, no. 5, pp. 1266 – 1276, Sep 1995.
[36] J. Max, "Quantizing for minimum distortion," *IRE Transactions on Information Theory*, vol. 6, no. 1, pp. 7 – 12, March 1960.
[37] R. Kwok and W. Johnson, "Block adaptive quantization of Magellan SAR data," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 27, no. 4, pp. 375 –383, Jul 1989.
[38] G. Kuduvalli, M. Dutkiewicz, and I. Cumming, "Synthetic aperture radar signal data compression using block adaptive quantization," in *SFC Space Earth Science Data Compression Conference*, 1994.
[39] V. Betz and J. Rose, "VPR: a new packing, placement and routing tool for FPGA research," in *International Workshop on Field Programmable Logic and Applications*, 1997, pp. 213–222.

Ediz Cetin is a Senior Research Associate in the School of Surveying & Geospatial Engineering, at the University of New South Wales, Sydney, Australia. He received his BEng.(Hons) and PhD degrees from the University of Westminster, London, United Kingdom. His research interests are in Global Navigation Satellite Systems (GNSS),Software Defined Radio, Blind signal processing and design & low-power implementation of digital circuits. He has authored/co-authored over 50 technical publications and holds two patents in the areas of Communications and GNSS receivers.

Oliver Diessel is a Senior Lecturer in the School of Computer Science & Engineering, at the University of New South Wales, Sydney, Australia. He gained B.E., B.Math., and Ph.D. degrees from the University of Newcastle, Australia. His research interests encompass the design and application of dynamically reconfigurable systems and technology, including modelling, design methods, and run-time support for such computer systems. He has co-authored over 60 publications and is an active participant in the leading forums in this area.

Lingkan Gong received the B.Sc. degree in Applied Physics and the M.Eng. degree in Computer Science and Engineering from Tong Ji University, China and East China Institute of Computer Technology, in 2005 and 2008, respectively. He is currently a Ph.D. candidate in the School of Computer Science and Engineering at the University of New South Wales. His research interests include dynamically reconfigurable systems and the functional verification of such systems.

Victor Lai received the B.Eng degree in Computer Science from University of New South Wales in 2008. He is currently a Masters of Engineering candidate at the University of New South Wales. His research interests include dynamically reconfigurable systems and how such reconfigurability can be better utilized at runtime.