Douglas Cunningham, Petra Hofstedt, Klaus Meer, Ingo Schmitt (Hrsg.): INFORMATIK 2015
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2015

Parity-based Soft Error Detection with Software-based Retry vs. Triplication-based Soft Error Correction - An Analytical Comparison on a Flash-based FPGA Architecture

Gökçe Aydos[1] Görschwin Fey[1 2]

Abstract: *Field-programmable gate arrays* (FPGAs) are often utilized in space avionics. To protect the FPGA logic against the ionizing radiation effects in space, redundancy in form of concurrent error detection can be used.

In this work, we present a comparative study of a parity-based error detection with software-based retry, and a triple modular redundancy technique on a known flash-based FPGA architecture (Microsemi ProASIC3). We compare critical path delay, circuit area overhead, multiple bit error probability and error correction time penalty. Our analysis shows that a solution based on parity-based error-detection can at least save about half of the resource overhead caused by triplication of the flip-flops if the target circuit can be functionally isolated from the rest of the circuit in the FPGA and if the software supports retransmission of access requests.

Keywords: fault tolerance, FPGA, ProASIC3

1 Introduction

Field-programmable gate arrays (FPGAs) are often utilized in space avionics due to their processing efficiency, reprogrammability, and extensible interface capabilities; providing flexibility for a range of mission requirements. The avionics must be protected from ionizing radiation in space. In the absence of a shield (e.g., magnetic field of the earth), a high energy particle can traverse through a digital circuit and induce significant amount of charge, which can eventually cause unexpected system responses like random signal glitches on the sensor data, but also catastrophic system failures like mission-loss due to a nonresponsive satellite [Pe11a]. Due to the lack of cost-efficient physical access to the space system, the avionics must implement intrinsic fault-tolerance mechanisms based on the mission requirements.

A local corruption of information stored in a node by a single energetic particle is called *single event upset* (SEU). If an SEU is latched by an FF (flip-flop), then it can result in a static bitflip. These errors are not permanent and can be corrected e.g., with a reset, thus they are also called *soft errors* [Pe11b]. Soft errors often happen in the sequential elements of a circuit, due to the latching-window, electrical- and logical-barriers of combinatorics [Li94].

[1] University of Bremen, Reliable Embedded Systems, 28359 Bremen, goekce@cs.uni-bremen.de
[2] German Aerospace Center, Institute of Space Systems, 28359 Bremen, goerschwin.fey@dlr.de

Error detection involves only the discovery of an error, while the error correction takes care of both detecting and recovering the correct information after an upset. This requires a sort of information redundancy in form of *space*, e.g., triplication and voting, or *time*, e.g., processing information three times by a single unit and comparing the results. Often, error correction in an FPGA is implemented with space redundancy, especially as *triple modular redundancy* (TMR). Error correction typically requires more resources compared to error detection, in form of redundancy. In presence of tight constraints, this overhead can turn into a hurdle for fulfilling the design timing closure and area requirements.

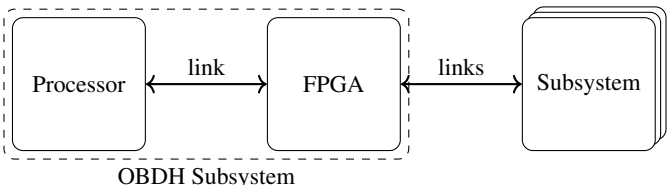

Fig. 1: Overview of an example data handling system. The processor runs the mission software and the FPGA implements interface protocol circuits required by various subsystems on-board of a satellite. The processor uses the FPGA for communicating with the subsystems.

Alternatively, a part of the space redundancy in the FPGA may be eliminated by implementing additional time redundancy, e.g., in software, if the FPGA acts as a co-unit beside an already radiation-hardened processor. An example architecture is depicted in Fig. 1, where the FPGA implements the communication protocol interfaces needed for communicating with the satellite subsystems and the processor runs the mission software. The processor uses the FPGA in a master-slave manner. The FPGA circuit only implements error detection and, in case of an error, the software instructs the FPGA to reprocess the last request. With this collaborative approach, error correction is achieved and the overhead of local error correction is eliminated. This technique will be referred as *error detection with software-based retry* (EDSR). In this paper, a *parity-based* error detection technique is used in the implementation of EDSR.

Parity-based codes and *triplication* are well-known *concurrent error detection* techniques (CED) [NZ98],[Gö08]. Also *error detection with retry* for achieving error correction was proposed, e.g., in [Ni99]. In recent years, one the one hand, partial hardening techniques were proposed due to the relatively high overhead of CED techniques, which selectively harden susceptible parts of the circuit [MT03]. On the other hand, software-based fault-tolerance techniques are also popular due to the flexibility and relatively loose constraints of software, e.g., regarding memory requirements, compared to hardware [Re02],[Go06]. Software- and hardware-based techniques have their tradeoffs, therefore these can also be used together [Re02].

This work applies parity-based error detection with software-based retry and triplication on an example data handling architecture based on a commercially-available flash-based FPGA and provides an analytical comparison of critical path overhead, area overhead, multiple bit error probability, and error correction time penalty. This FPGA is chosen because it is state-of-the-art for space missions (e.g., [Tr14]) and it is optionally available in a special integrated circuit package for space environment. Our contributions are (a) the

discussion of parity-based error detection in the context of the full system stack and (b) the analysis of TMR versus EDSR with respect to space-proven technology.

In the following sections, we firstly explain the TMR and EDSR techniques and the particular implementations which are compared. Then, a reference data processing system is presented, which is used as a testbench for comparing the two implementations. Afterwards, the analytical analysis and its results based on a commercially-available flash-based FPGA architecture are presented. Our analysis shows that the software-based error correction approach can cut down area overhead about 50 % compared to TMR at the expense of extra software runtime, if the target circuit can be functionally isolated from the rest of the FPGA circuit in case of an SEU.

2 Compared Hardening Techniques

In this section TMR and EDSR techniques are described more in detail including their system impacts.

2.1 Triple Modular Redundancy

In TMR, one module is triplicated and the outputs of the three modules are input to a voter, which outputs the majority value. A *module* in this sense can be anything from a whole system to a small functional block or simply a gate. TMR regarding FPGAs can be implemented at various abstraction levels, e.g., at circuit- or gate-level.

There are various TMR techniques based on the reliability requirements of a circuit [Be08]. One of them is the *Local TMR* (LTMR) and is applied on the gate level; a combinational net being registered by an FF is connected to two additional FFs and the outputs of the three FFs are connected to a majority voter. In this work, only SEUs on the FFs are considered. Consequently, the local TMR is used as the compared TMR technique.

TMR detects and corrects a single bit error on an FF locally using a majority voter, hence the TMR techniques can be automatically applied on top of a circuit. This makes TMR functionally transparent to the rest of the system, consequently the circuit mostly does not require a redesign before mapping to an FPGA.

2.2 Error Detection with Software-based Retry

Detection of an error also requires space or time redundancy, but often less redundancy resources than both detection and correction. If the resources on a device are scarce and costly, then implementing a local error correction scheme can become a hurdle. In this case, the error correction can be moved, e.g., to software, if the processing architecture renders it possible. Issuing a non-local error correction requires more recovery time than

a local correction, beginning from the detection until the correction of the error. Nevertheless, if the error rate of the system is low, then a non-local error correction can be practicable.

A well-known error detection technique is *parity-based error detection* (PBED), which adds a parity bit to every data word being stored, e.g., by XORing the data bits and storing the result along with the data word. [NZ98] Upon reading the data word, the parity is calculated again, compared to the stored parity value and in case of a mismatch, an error signal is asserted. Subsequently, an error handler can react and initiate a recovery scheme to correct the error.

After an error, a module must be recovered to an operational state. Often, this is done by resetting the module to its initial state. This in turn leads to a loss of the processing context that must be brought back, which involves periodically backing up the processing context, i.e., checkpointing. If the processing context does not contain any information which is needed for a long time, i.e., when a module regularly falls back to a defined state, then the overhead of checkpointing in the circuit may be eliminated by reissuing a processing request. Examples for such a module are a protocol converter or simply a module which exchanges data between two modules after reformatting data. These modules do not have to store an information for a long time and have a defined state after a chunk of data or a transaction is processed. The example FPGA circuit B presented in Fig. 2 falls also in this category, as it only exchanges data between two modules and moves to its initial state after a request is processed. If an SEU occurs during processing of a request, then the error handler can reset the processing module and flag an error to the processor that a processing request can be reissued, i.e., software-based retry. Alternatively, instead of flagging, the request can be reissued after a nonresponsive timeout. In this case, the time penalty caused by an SEU is negligible, if the FPGA SEU rates during a mission due to space radiation are low.

3 Reference Architecture

We compare the two hardening techniques using a reference model of an on-board data handling unit for a satellite [Tr14]. In the following, important parts of the system are described at the functional level.

3.1 Overview

The on-board data handling unit comprises of two main processing modules: a processor and an FPGA. The processor runs the mission software, which involves communicating with different subsystems on-board of the space system. The communication is done through the FPGA, which acts as an interface component and implements the various communication interfaces needed by the subsystems (e.g., RS232, CAN). Fig. 1 shows an overview of the architecture. We assume that the processor, the communication line between the processor and the FPGA, and the subsystems are sufficiently protected from any

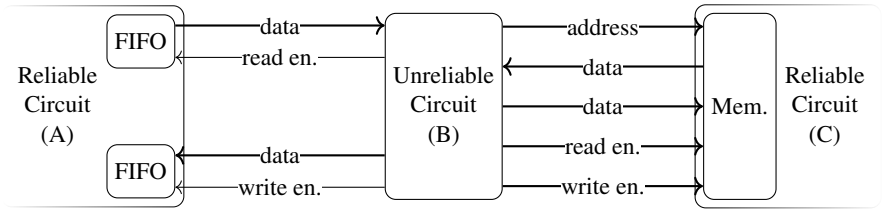

Fig. 2: Simplified model of the FPGA design architecture. The circuit in the middle processes the memory access requests from the left side and responds according to the result of the processed request. The reliable parts are assumed to be immune to SEUs. The unreliable circuit must be hardened on the design level.

single upsets through intrinsic redundancy in ICs and error-correcting code on communication links.

3.2 FPGA Design

From the processor point of view, the FPGA is a remote memory bus, where the implemented link interfaces are memory-mapped. The processor utilizes these interface modules by reading and writing the respective memory areas.

The FPGA model consists of three functional blocks, circuit A, B, and C as shown in Fig. 2. Circuit A serves the memory access requests from the processor to circuit B, which issues memory accesses on circuit C and finally returns the data to the processor using the FIFO interface of circuit A. Circuit C with a memory block inside resembles the memory-mapped interfaces. Reliable circuits A and C in this architecture are assumed to be sufficiently protected against SEUs (e.g., by TMR), whereas the unreliable one must be protected by a soft error hardening technique. The compared hardening techniques will be applied on the unreliable circuit.

The FIFOs and the memory need a single clock cycle for reading or writing a single word, which renders the masking of a single word access operation in a clock cycle (in case of an error) possible.

3.3 Communication Protocol

The communication protocol between the processor and the FPGA consists of two kinds of messages: *request* and *response*. The processor sends memory access requests for a specific address or address interval to the FPGA and the FPGA responds with the according response: In case of a read request, the response carries the data which is requested by the processor. If a write request is issued, the FPGA sends an acknowledge (ACK) response after the write request is complete. A not-acknowledge (NACK) response is sent, if a request cannot be successfully processed. Every request is answered with a response and a second request cannot be sent before the response to the first request has been received.

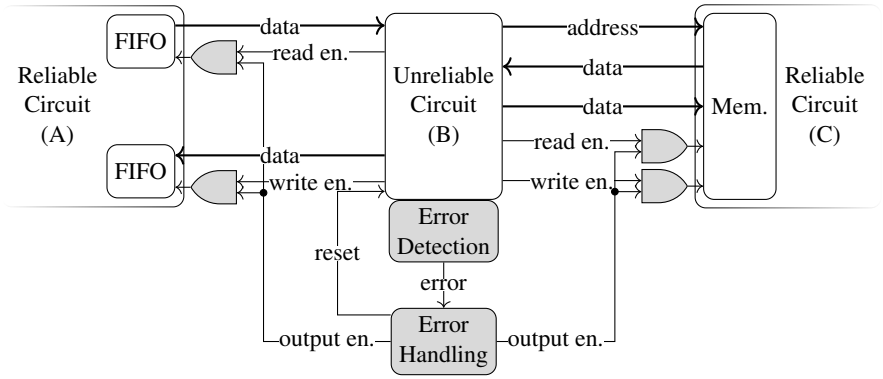

Fig. 3: Parity-based error detection applied on the unreliable circuit. Data redundancy is generated by the parity generation block. The error detection block checks the data integrity. The error handling block generates the recovery and masking signals. The AND gates isolate the unreliable circuit by masking all the control signals which can change the state of the neighbor circuits.

4 Implementation

This section explains how the particular TMR and EDSR using PBED techniques are implemented on the reference architecture.

4.1 Triple Modular Redundancy

In this work, we concentrate on the SEUs in the sequential parts of a circuit, therefore LTMR is implemented as the compared TMR technique. The implementation is straightforward and it does not require additional attention on the hardened circuit, because it can be applied on top of a logical circuit before it is placed and routed for the FPGA.

4.2 Error Detection with Software-based Retry

Fig. 3 shows PBED applied on the reference FPGA design. The *error detection* block continuously generates the data redundancy and checks the integrity of data. If an error is detected, the *error* signal is asserted and the *error handling* block immediately masks the control signals on either side of the unreliable circuit.

The FFs in the unreliable circuit are segmented to groups and for each group one parity FF is introduced. One single group with a parity FF is called a *cluster*. Fig. 4 shows the generic implementation of the error detection in a single cluster. The number of clusters is given by c_{cl} (c: count, cl: cluster). Each cluster contains $s_{\mathrm{cl}} - 1$ user FFs plus one parity FF (s: size). Even parity is generated by XORing the inputs to the user FFs by the $\mathrm{XOR}_{\mathrm{pg}}$. The integrity of the stored bits is checked by the $\mathrm{XOR}_{\mathrm{pc}}$ with s_{cl} inputs and the *cluster error* is generated by each cluster. Finally, c_{cl} cluster error signals are reduced to a single

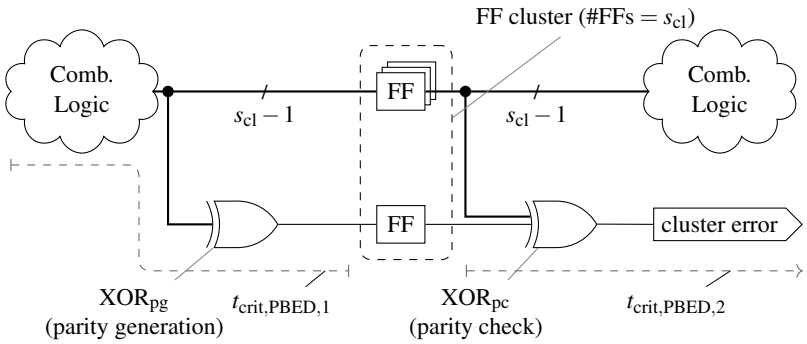

Fig. 4: Implementation of PBED part (a): generation of the cluster error signal.

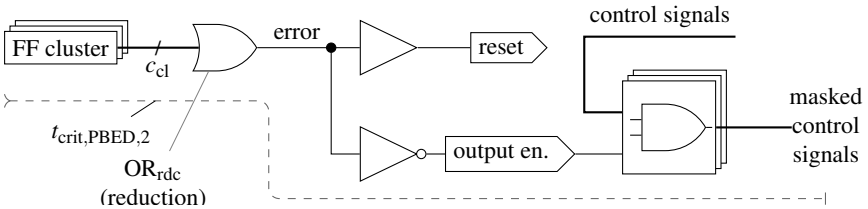

Fig. 5: Implementation of PBED part (b): generation of the global error signal and error handling.

error signal by an OR gate. The reduction of the cluster error signals and subsequent error handling is shown in Fig. 5.

The error handling is done by generating the *reset* and *output enable* signals combinationally using the error signal. The enable signal masks the control signals (i.e., FIFO and memory control signals) of the unreliable circuit. The reset signal recovers the circuit from a possibly erroneous state to its initial state. In the next cycle, the error flag is deasserted and the unreliable module begins data processing again.

If an incomplete or no response is received by the processor in the timeout window, then a recovery procedure is initiated. If an error happens during processing of a read request, then this request is repeated. If an error occurs in the middle of a write transaction, the software cannot know which part of the transaction was completed and the software can synchronize itself by reading these addresses again or simply retry the last transaction. If a write to a memory location triggers an operation (e.g., transmitting a command to a subsystem), then retrying retriggers the last operation, which can be undesirable and dangerous.

In case of such *action-triggering* memory locations, the software can issue single memory write operations only. This has the advantage that every atomic memory write operation is acknowledged separately and the software knows exactly which single memory operation did not succeed. This requirement can be loosened, if a memory area is written which does not trigger an action, i.e., the output of the target system does not change after the

transaction. An example is the transmit buffer of a communication interface module, where the transmit operation must be first triggered by setting a bit in a control register allowing to begin a data transfer to a subsystem. In this case, the processor would first try to write the transmit payload-data to the buffer with a single write request and in the subsequent request the transmission operation would be triggered using an atomic memory access.

5 Analytical Comparison of Needed Resources

In this section, we analytically determine and compare critical path delay, circuit area overhead, multiple bit error probability and discuss the error correction time penalty of the two shown techniques. The circuits are mapped to the Microsemi flash-based radiation-tolerant ProASIC3 FPGA (RT3PE) [Mic13] featuring three input LUTs. This is a known space-proven FPGA and utilized in state-of-the-art on-board-computing systems.

Many of the comparison parameters are dependent on the size of one cluster s_{cl} and the total cluster count in the unreliable circuit c_{cl}. The parameters are determined for $s_{\mathrm{cl}} \stackrel{!}{=} 3^x$ and $c_{\mathrm{cl}} \stackrel{!}{=} 3^y$, where $x, y \in \mathbb{N}$, which fits the RT3PE architecture with three input LUTs.

This selection of input parameters makes the most timing-efficient use of the FPGA area for a specific logic depth. With the increasing count of s_{cl} and c_{cl} more LUTs are needed for parity generation and the reduction of cluster error signals, respectively. With increasing number of LUTs on a critical path, longer delay is introduced on this path. However, the additional delay is only proportional to the logarithm of s_{cl} and c_{cl}. Consequently, the critical path of a benchmark design only changes for different values $x, y \in \mathbb{N}$, leading to such selection of s_{cl} and c_{cl} values. This behavior is visualized in Fig. 6 and explained in Subsection 5.1 more in detail.

In PBED, for each group of $s_{\mathrm{cl}} - 1$ FFs one parity bit is generated. Logic optimization (e.g., logic packing, retiming) and interconnect delays are not considered, which depend significantly on the resource utilization in an FPGA.

In the following, the nominal parameters (i.e., hardening not implemented) are labeled with the subscript $_{\mathrm{nom}}$ and the parameters of the circuits with LTMR and PBED with $_{\mathrm{LTMR}}$ and $_{\mathrm{PBED}}$, respectively. An overhead in a measurement parameter by the applied technique is labeled with the subscript $_{+}$.

5.1 Critical Path Delay

The critical path delay t_{crit} limits the maximum frequency of a design and increases with additional serial logic. In LTMR, every bit must be decoded by a majority voter (MAJ3) before it is propagated to the combinational logic, which causes an extra delay. The subscript $_{\mathrm{pd}}$ stands for propagation delay.

$$t_{\mathrm{crit}+,\mathrm{LTMR}} = t_{\mathrm{pd,MAJ3}} \tag{1}$$

In PBED, there are two critical path candidates (see Fig. 4 and 5):

1. The critical path of the nominal circuit plus the parity generation path ($t_{\text{crit,PBED,1}}$)
2. The error detection plus the error handling plus the isolation path ($t_{\text{crit,PBED,2}}$)

The first path delay can be calculated as follows: The parity has to be generated before the combinational signals are registered. The propagation delay of the XOR_{pg} block is called $t_{\text{pd,XOR}_{\text{pg}}}$.

$$t_{\text{crit+,PBED,1}} = t_{\text{pd,XOR}_{\text{pg}}} \tag{2}$$

The error detection path consists of the XOR_{pc}, OR_{rdc}, a NOT gate, and an AND gate (Fig. 4 and 5). The NOT gate and the AND gate can be packed into one LUT, which is called *OR2A*:

$$t_{\text{crit,PBED,2}} = t_{\text{pd,XOR}_{\text{pc}}} + t_{\text{pd,OR}_{\text{rdc}}} + t_{\text{pd,OR2A}} \tag{3}$$

XOR_{pc}, XOR_{pg} and OR_{rdc} are trees of LUTs as shown in Fig. 6. The propagation delay of a *block* with an input size s_{input} is called $t_{\text{pd}}(block, s_{\text{input}})$ and can be calculated by determining the depth d of the tree and multiplying it with the propagation delay of the respective three input macro (e.g., OR3 for an OR block), as the interconnect delays are not considered.

$$\begin{aligned} t_{\text{pd}}(block, s_{\text{input}}) &= d_{block} \cdot t_{\text{pd},macro} \\ &= \lceil \log_3 s_{\text{input}} \rceil \cdot t_{\text{pd},macro} \end{aligned} \tag{4}$$

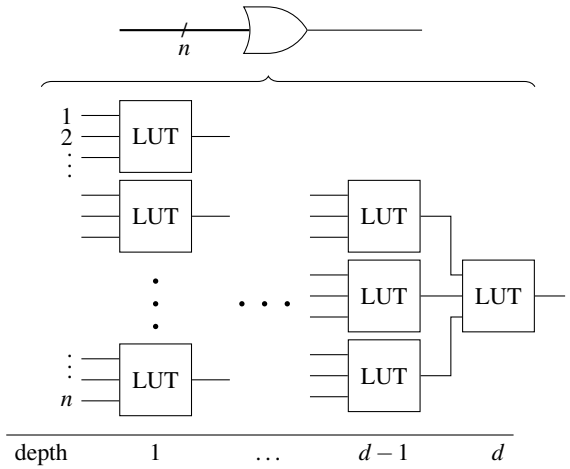

Fig. 6: The figure shows how a gate with n inputs is mapped to an FPGA architecture with three input LUTs. After mapping, a LUT tree with a depth of $d = \lceil \log_3 n \rceil$ is created. Note that if n is not a power of three, then not all the leafs of the tree exist.

With Eq. 4, the propagation delays of the three defined blocks can be calculated:

$$\begin{aligned} t_{\text{pd,XOR}_{\text{pg}}} &= \lceil \log_3(s_{\text{cl}} - 1) \rceil \cdot t_{\text{pd,XOR3}} \\ t_{\text{pd,XOR}_{\text{pc}}} &= \lceil \log_3 s_{\text{cl}} \rceil \cdot t_{\text{pd,XOR3}} \\ t_{\text{pd,OR}_{\text{rdc}}} &= \lceil \log_3 c_{\text{cl}} \rceil \cdot t_{\text{pd,OR3}} \\ &= \left\lceil \log_3 \left\lceil \frac{c_{\text{FF,nom}}}{s_{\text{cl}}} \right\rceil \right\rceil \cdot t_{\text{pd,OR3}} \end{aligned} \tag{5}$$

As $t_{\text{crit,PBED,2}}$ is generated in parallel to the nominal circuit (i.e., not serial like $t_{\text{crit,PBED,1}}$), $t_{\text{crit,PBED,2}}$ stays uncritical up to a certain depth of parity check and reduction blocks. Therefore the parameters s_{cl} and $c_{\text{FF,nom}}$ limit the maximum frequency of the design.

At a junction temperature of 125°Cand worst-case supply voltage 1.14 V, the $t_{\text{pd,MAJ3}}$, $t_{\text{pd,XOR3}}$, $t_{\text{pd,OR3}}$, $t_{\text{pd,OR2A}}$ are 1.14 ns, 1.42 ns, 1 ns and 1 ns[3] respectively [Mic13]. With these data the critical path caused by the FFs and combinational elements can be calculated for various s_{cl} and $c_{\text{FF,nom}}$ parameters.

			$t_{\text{crit+,1}}$ (ns)		$t_{\text{crit,2}}$ (ns)	$Area_+$		$Area_+ : c_{\text{FF,nom}}$	
(x,y)	s_{cl}	$c_{\text{FF,nom}}$	PBED	LTMR	PBED	PBED	LTMR	PBED	LTMR
(1,4)		162			7.84	248	486	153 %	
(1,5)	3	486	1.42	1.14	8.84	734	1458	151 %	300 %
(1,6)		1458			9.84	2192	4374	150 %	
(2,3)		216			8.26	251	648	114 %	
(2,4)	9	648	2.84	1.14	9.26	737	1944	114 %	300 %
(2,5)		1944			10.26	2195	5832	113 %	
(3,2)		234			8.68	260	702	111 %	
(3,3)	27	702	4.26	1.14	9.68	746	2106	106 %	300 %
(3,4)		2106			10.68	2204	6318	105 %	

Tab. 1: Comparison of PBED and LTMR regarding critical path and area overhead.

Table 1 shows the critical path delays $t_{\text{crit+,1}}$ and $t_{\text{crit,2}}$ for various values of the input parameter (x,y). The parameters s_{cl} and $c_{\text{FF,nom}}$ are determined using (x,y), where $s_{\text{cl}} = 3^x$, cluster count $c_{\text{cl}} = 3^y$ and nominal FF count $c_{\text{FF,nom}} = (s_{\text{cl}} - 1) \cdot c_{\text{cl}}$. With increasing depth of XOR_{pg}, $t_{\text{crit+,1}}$ grows for PBED, i.e., every time when s_{cl} reaches a higher power of 3. The additional path delay $t_{\text{crit+,1}}$ of LTMR is independent of the input parameters. For $s_{\text{cl}} = 3$ PBED and LTMR have a similar critical path overhead. PBED has additionally the $t_{\text{crit+,2}}$, which grows with increasing depth of XOR_{pc} and OR_{rdc} blocks.

[3] $t_{\text{pd,OR3}}$ and $t_{\text{pd,OR2A}}$ were available neither in the datasheet or macro library documentation. Therefore an estimated value of 1 ns is assumed for these two parameters.

5.2 Circuit Area Overhead

Assuming that the circuit area is proportional to the CLB count, we define the parameter *Area* as the CLB count. For comparison, we are interested in the area overhead $Area_+$, i.e., the CLB cost $c_{\text{CLB}+}$:

$$Area_+ = c_{\text{CLB}+} \tag{6}$$

In ProASIC3 architecture, every *configurable logic block* (CLB)[4] can be either configured as an FF or LUT. Then, the circuit area overhead can be calculated by adding the count of additionally introduced LUTs and FFs:

$$c_{\text{CLB}+} = c_{\text{LUT}+} + c_{\text{FF}+} \tag{7}$$

In the LTMR design, the FFs are triplicated, i.e., two additional FFs are added for each FF:

$$c_{\text{FF}+,\text{LTMR}} = 2 \cdot c_{\text{FF,nom}} \tag{8}$$

LTMR requires one LUT per FF as voter:

$$c_{\text{LUT}+,\text{LTMR}} = c_{\text{FF,nom}} \tag{9}$$

In total, the area overhead for LTMR is:

$$Area_{+,\text{LTMR}} = c_{\text{CLB}+,\text{LTMR}} = 3 \cdot c_{\text{FF,nom}} \tag{10}$$

In PBED, one parity register is needed for a single cluster:

$$c_{\text{FF}+,\text{PBED}} = c_{\text{cl}} \tag{11}$$

In PBED, LUTs are needed for the XOR_{pg}-, XOR_{pc}-, OR_{rdc}-blocks, and OR2A gates for masking the control signals:

$$\begin{aligned} c_{\text{LUT}+,\text{PBED}} =& c_{\text{cl}}\left(c_{\text{LUT,XOR}_{\text{pg}}} + c_{\text{LUT,XOR}_{\text{pc}}}\right) \\ &+ c_{\text{LUT,OR}_{\text{rdc}}} + c_{\text{LUT,OR2A}} \end{aligned} \tag{12}$$

As shown in Fig. 6, a block with n inputs $block_n$ creates a tree, so the needed maximum LUT count for a tree of depth d can be determined by the following formula, assuming that every new level of the tree introduces 3^{depth} LUTs at maximum:

$$\begin{aligned} c_{\text{LUT},block_n,\max} &= \sum_{i=0}^{d_{block_n}-1} 3^i \\ &= \frac{1}{2} \cdot \left(3^{d_{block_n}} - 1\right) \end{aligned} \tag{13}$$

[4] In ProASIC3 terminology, a CLB is called a *tile* or *VersaTile*

Using the formula for depth $d = \lceil \log_3 n \rceil$ (Fig. 6):

$$c_{\text{LUT},block_n,\max} = \frac{1}{2} \cdot (3^{\lceil \log_3 n \rceil} - 1) \tag{14}$$

If n is a power of 3 (e.g., in case of XOR_{pc} and OR_{rdc}), then the equation can be further simplified:

$$\begin{aligned} n \stackrel{!}{=} 3^x, x \in \mathbb{N} &\implies 3^{\lceil \log_3 n \rceil} = n \\ &\implies c_{\text{LUT},block_n} = \frac{1}{2} \cdot (n-1) \end{aligned} \tag{15}$$

If $n+1$ is a power of 3 (e.g., in case of XOR_{pg}), the same amount of LUTs are required. This is due to the fact that a block will in this case contain a single two-input LUT with the rest being three-input LUTs. A two- and a three-input LUT both occupy one CLB, thus the same area.

$$\begin{aligned} n+1 \stackrel{!}{=} 3^x, x \in \mathbb{N} &\implies 3^{\lceil \log_3 n \rceil} = n+1 \\ &\implies c_{\text{LUT},block_n} = \frac{1}{2} \cdot n \end{aligned} \tag{16}$$

With Eq. 15, $c_{\text{LUT,XOR}_{\text{pc}}}$ and $c_{\text{LUT,OR}_{\text{rdc}}}$; and with Eq. 16, $c_{\text{LUT,OR}_{\text{pg}}}$ can be determined. Additionally, there are four OR2As in the PBED implementation. Hence, the Eq. 12 can be rewritten to:

$$\begin{aligned} &c_{\text{LUT+,PBED}} = \\ &= c_{\text{cl}}\Big(\frac{1}{2} \cdot (s_{\text{cl}} - 1) + \frac{1}{2} \cdot (s_{\text{cl}} - 1)\Big) + \frac{1}{2}(c_{\text{cl}} - 1) + 4 \\ &= c_{\text{cl}}(s_{\text{cl}} - 1) + \frac{1}{2}(c_{\text{cl}} - 1) + 4 \end{aligned} \tag{17}$$

Finally, with Eq. 7, 11 and 17, total area cost for PBED equals to:

$$\begin{aligned} \mathit{Area}_{+,\text{PBED}} &= c_{\text{cl}} + c_{\text{cl}}(s_{\text{cl}} - 1) + \frac{1}{2}(c_{\text{cl}} - 1) + 4 \\ &= c_{\text{cl}}(s_{\text{cl}} + \frac{1}{2}) + 3.5 \end{aligned} \tag{18}$$

$c_{\text{FF,nom}}$ is a main input parameter, therefore it is better to rewrite c_{cl} using $c_{\text{FF,nom}}$:

$$\mathit{Area}_{+,\text{PBED}} = \frac{c_{\text{FF,nom}}}{s_{\text{cl}} - 1}(s_{\text{cl}} + \frac{1}{2}) + 3.5 \tag{19}$$

Table 1 shows the area overhead Area_+ and area overhead caused per FF $\mathit{Area}_+ : c_{\text{FF,nom}}$[5] for various values of s_{cl} and $c_{\text{FF,nom}}$ parameters. PBED area overhead is approximately 59 % of the LTMR area overhead for $s_{\text{cl}} = 3$ and it decreases with increasing s_{cl} and c_{cl}. The LTMR area overhead is independent of the input parameters.

[5] Area overhead Area_+ is related to $c_{\text{FF,nom}}$ instead of the whole design including combinatorics, because the area overhead is only dependent on $c_{\text{FF,nom}}$ and the combinatorics LUT count is arbitrary.

5.3 Multiple bit error probability

We apply the definition of a cluster also on LTMR and define an LTMR cluster as the group of three FFs after triplication, i.e., $s_{\text{cluster,LTMR}} = 3$. LTMR and PBED techniques both are immune against one bit flip in a clock cycle, but not against multiple bit errors, assuming that every FF in LTMR is updated in every clock cycle[6]. In this subsection, we will compare the LTMR and PBED regarding multiple bit error probability, i.e., the probability that an error cannot be detected on the circuit.

If a single particle travels through the circuit, then it can cause single or multiple bit errors dependent on the amount of energy transferred to the circuit and the size of the IC structures. In this analysis, we assume that the CLBs are far enough from each other to consider bitflips as independent events. Therefore, define p as the bit flip probability of a single FF in a clock cycle and assume that p for individual FFs are statistically independent. Then, the multiple bit error (MBE) probability in a single cluster $p_{\text{MBE,cl}}$ can be calculated by:

$$\begin{aligned} p_{\text{MBE,cl}} &= \sum_{i=2}^{s_{\text{cl}}} \binom{s_{\text{cl}}}{i} p^i (1-p)^{s_{\text{cl}}-i} \\ &= 1 - \sum_{i=0}^{1} \binom{s_{\text{cl}}}{i} p^i (1-p)^{s_{\text{cl}}-i} \\ &= 1 - (1-p)^{s_{\text{cl}}} - s_{\text{cl}} \cdot p(1-p)^{s_{\text{cl}}-1} \end{aligned} \tag{20}$$

The last equation assumes that all kinds of multiple bitflips cannot be detected by PBED. In fact, PBED can detect all odd number of bitflips in a cluster, but the probability of multiple bit flips in a cluster greater than 2 is negligible. With $p_{\text{MBE,cl}}$, the multiple error probability of the whole circuit p_{MBE} can be calculated in a similar manner like in the previous equation:

$$\begin{aligned} p_{\text{MBE}} &= \sum_{i=1}^{c_{\text{cl}}} \binom{c_{\text{cl}}}{i} p_{\text{MBE,cl}}^i (1-p_{\text{MBE,cl}})^{c_{\text{cl}}-i} \\ &= 1 - (1-p_{\text{MBE,cl}})^{c_{\text{cl}}} \\ &= 1 - ((1-p)^{s_{\text{cl}}} + s_{\text{cl}} \cdot p(1-p)^{s_{\text{cl}}-1})^{c_{\text{cl}}} \end{aligned} \tag{21}$$

Assuming one year mission in L2 orbit under 1/cm² shielding, a programmed circuit with 5000 FFs on an RTPE3000L FPGA has four SEUs [BSV11]. If this design runs at 20 MHz, then p can be calculated by:

$$\begin{aligned} p &= 4/5000/365/24/60/60/(20 \times 10^6) \\ &\approx 1.27 \times 10^{-18} \end{aligned} \tag{22}$$

Table 2 shows a comparison of multiple bit error probabilities for various s_{cl} $c_{\text{FF,nom}}$ parameters. For $s_{\text{cl}} = 3$, p_{MBE} is approximately the same for PBED and LTMR. Generally,

[6] Otherwise, the bitflips can accumulate and lead to uncorrectable errors.

when $c_{\mathrm{FF,nom}}$ increases, p_{MBE} also increases, but for $s_{\mathrm{cl}} > 3$, PBED is more susceptible to multiple bit errors.

			p_{MBE}	
(x,y)	s_{cl}	$c_{\mathrm{FF,nom}}$	PBED	LTMR
(1,4)		162	7.82 E-34	7.82 E-34
(1,5)	3	486	2.35 E-33	2.35 E-33
(1,6)		1458	7.04 E-33	7.04 E-33
(2,3)		216	1.25 E-32	1.04 E-33
(2,4)	9	648	3.75 E-32	3.13 E-33
(2,5)		1944	1.13 E-31	9.38 E-33
(3,2)		234	1.32 E-31	1.13 E-33
(3,3)	27	702	3.96 E-31	3.39 E-33
(3,4)		2106	1.19 E-30	1.02 E-32

Tab. 2: Comparison of PBED and LTMR regarding multiple bit error probability of one cluster $p_{\mathrm{MBE,cl}}$ and whole circuit p_{MBE}

Overall, $s_{\mathrm{cl}} = 3$ is a reasonable choice for saving significant amount of FPGA resources, and for having as little impact on the critical path as possible. If the maximum frequency is not important, then higher s_{cl} results in less area overhead up to approximately 35 % of the LTMR area overhead.

5.4 Error Correction Time Penalty

In LTMR, the error is corrected in the same clock cycle, but EDSR requires that the error is corrected by the software by repeating the memory access request, which in turn causes additional processing delays. Consequently, the total time penalty is proportional to the error rate during the mission. For example, assuming the same error rate from the Section 5.3 makes the time penalty per year insignificant.

6 Conclusion

The FPGAs used in space applications must be protected against radiation induced errors, which is often done by redundancy. TMR is often used on FPGA designs, which can correct the induced errors locally. If the logic resources are scarce and SEU rate on the FPGA during a mission is low, then the error correction functionality can be shifted to the software, leaving an FPGA circuit only with error detection. We have shown an example architecture which implements PBED with software-based retry and analytically compared the needed resources for the Microsemi ProASIC3 architecture. The results show that significant part of the area overhead caused by the LTMR can be saved by implementing PBED on an FPGA circuit and correcting the errors with time redundancy, i.e., repeating

the instructions to the FPGA in case of an error. The disadvantage of PBED shows up for greater s_{cl} values when the critical path for parity generation grows. Nevertheless, if the maximum frequency is not the first priority in design, then significant area can be saved at a cost of higher p_{MBE}. In certain applications this may be the key to adopt sufficient functionality in a single FPGA component.

Acknowledgment
This work has been supported by the University of Bremen's Graduate School SyDe, funded by the German Excellence Initiative.

References

[Be08] Berg, Melanie: Design for Radiation Effects. Presentation from Military and Aerospace Programmable Logic Devices (MAPLD) Workshop, 2008.

[BSV11] Battezzati, Niccoláo; Sterpone, Luca; Violante, Massimo: Reconfigurable Field Programmable Gate Arrays for Mission-Critical Applications. Springer, chapter 7, 2011.

[Go06] Goloubeva, Olga; Rebaudengo, Maurizio; Reorda, Matteo Sonza; Violante, Massimo: Software-implemented hardware fault tolerance. Springer, 2006.

[Gö08] Gössel, Michael; Ocheretny, Vitaly; Sogomonyan, Egor; Marienfeld, Daniel: New methods of concurrent checking, volume 42 of Frontiers In Electronic Testing. Springer Netherlands, 2008.

[Li94] Liden, P.; Dahlgren, P.; Johansson, R.; Karlsson, J.: On latching probability of particle induced transients in combinational networks. In: 24th International Symposium on Fault-Tolerant Computing (FTCS). pp. 340–349, June 1994.

[Mic13] Microsemi. Radiation-Tolerant ProASIC3 Low Power Spaceflight Flash FPGAs Datasheet, November 2013.

[MT03] Mohanram, K.; Touba, N.A.: Cost-effective approach for reducing soft error failure rate in logic circuits. In: International Test Conference (ITC). volume 1, pp. 893–901, Sept 2003.

[Ni99] Nicolaidis, M.: Time redundancy based soft-error tolerance to rescue nanometer technologies. In: 17th IEEE VLSI Test Symposium. pp. 86–94, 1999.

[NZ98] Nicolaidis, M.; Zorian, Y.: On-Line Testing for VLSI - A Compendium of Approaches. Journal of Electronic Testing Theory and Applications (JETTA), 12:7–20, February 1998.

[Pe11a] Petersen, Edward: Single Event Effects in Aerospace. John Wiley & Sons, chapter 1, 2011.

[Pe11b] Petersen, Edward: Single Event Effects in Aerospace. John Wiley & Sons, chapter 2, 2011.

[Re02] Rebaudengo, M.; Reorda, M.S.; Violante, M.; Nicolescu, B.; Velazco, R.: Coping with SEUs/SETs in microprocessors by means of low-cost solutions: a comparison study. IEEE Transactions on Nuclear Science, 49(3):1491–1495, Jun 2002.

[Tr14] Treudler, Carl Johann; Schröder, Jan-Carsten; Greif, Fabian; Stohlmann, Kai; Aydos, Gökçe; Fey, Görschwin: Scalability of a Base Level Design for an On-Board-Computer for Scientific Missions. In: Proceedings of the Data Systems in Aerospace (DASIA) Conference. 2014.