Microprocessors and Microsystems 48 (2017) 62–68

Contents lists available at ScienceDirect

Microprocessors and Microsystems

journal homepage: www.elsevier.com/locate/micpro

Empirical results on parity-based soft error detection with software-based retry

Gökçe Aydos[a,*], Goerschwin Fey[a,b]

[a] University of Bremen, Bremen, Germany
[b] German Aerospace Center, Bremen, Germany

ARTICLE INFO

Article history:
Received 28 February 2016
Revised 21 June 2016
Accepted 16 September 2016
Available online 17 September 2016

Keywords:
Fault-tolerance
FPGA
LTMR
Parity
Error-detection
Retry

ABSTRACT

Local triple modular redundancy (LTMR) is often the first choice to harden the FFs of a flash-based FPGA application against radiation-induced bitflips in space, but LTMR leads to an area overhead of roughly 300%. To cope with this significant overhead, we propose an error detection based approach. In this work, we compare parity-based error detection with software-based retry, and LTMR on a reference architecture regarding maximum frequency, area overhead and processing time. Our results show that our solution based on parity-based error-detection saves from 29% up to 36% of the area overhead caused by LTMR.

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), high energy particles can traverse through a digital circuit and cause errors. These errors can be caused by permanently damaging the semiconductor structure or induce significant amount of charge leading to a transient voltage pulse on a net, which can eventually lead to hard or soft errors, respectively.

The most common functional transient radiation effects that happen on the gate level, which can cause a soft error, are the *single event-transient* (SET) and *-upset* (SEU). An SET can be seen as a transient voltage pulse on a circuit net. If such a change happens on a data net and then latched by a FF, this transient can lead to an upset of the FF-bit and thus to an SEU. SEUs are not permanent and can be corrected e.g., with a reset.

Fault-tolerance against SEUs can be implemented at various levels of a circuit, e.g., process- or design-level. Hardening a circuit at the design-level is referred as *radiation hardening by design* (RHBD) [1] and involves the wise use of available design elements by the designer. RHBD is preferred if the designer has merely access to a commercially available *integrated circuits* (ICs) and IC manufacture processes, respectively.

The right RHBD techniques depend on the underlying FPGA architecture. Currently, three memory architectures exist on the market dependent on how the programmable logic is configured. These are SRAM-, flash- and antifuse-based architectures. On SRAM-based FPGAs, the circuit programming information, i.e., configuration, is stored on SRAM. SRAM has a high SEU sensitivity, therefore, compared to antifuse- and flash-based configurations, the configuration of SRAM-based parts must additionally be protected.

In this work, we assume a flash-based FPGA architecture. In flash-based FPGAs, SEUs mainly happen in the flip-flops (FFs) of an FPGA application. The FPGA configuration bits do not have to be protected, because flash memory has a negligible soft error rate due to SEUs.

The state-of-the-art solution against single bitflips for flash-based FPGAs is the *local triple modular redundancy* (LTMR), i.e., triplicating the application FFs and voting their outputs. Unfortunately, triplication has a significant area overhead. 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

* Corresponding author.
E-mail addresses: goekce@cs.uni-bremen.de (G. Aydos), goerschwin.fey@dlr.de (G. Fey).

http://dx.doi.org/10.1016/j.micpro.2016.09.009
0141-9331/

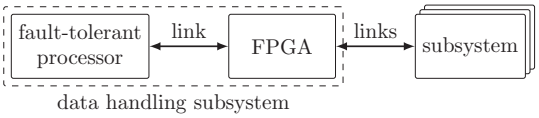

Fig. 1. Overview of the reference data handling architecture. Processor communicates with the subsystems through the FPGA.

communicating with the satellite subsystems and the processor runs the mission software.

The FPGA circuit which has to be hardened, only implements error detection. In case of an error, this circuit is functionally isolated, then recovered and the software finally instructs the circuit to reprocess the last request. With this collaborative approach, error correction is achieved and the overhead of local error correction is eliminated in the FPGA. This technique will be referred as *error detection with software-based retry* (EDSR). In this paper, *parity-based error detection* (PBED) is used in EDSR.

Parity-based codes and *triplication* are well-known *concurrent error detection* techniques (CED) [2,3]. Also *error detection with retry* for achieving error correction was proposed, e.g., in [4]. 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 [5]. 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 [6,7]. Software- and hardware-based techniques have their tradeoffs, therefore these can also be used together [6].

This work applies parity-based EDSR on an example data handling architecture based on a commercially-available flash-based FPGA and provides an experimental comparison to LTMR. Up to now, there is no detailed comparison based on a state-of-the-art (e.g., [8,9]) flash-based FPGA. Due to the limited resources of space-proven flash-based FPGAs, area savings can be the key for fitting the application onto the FPGA. Our contributions are

- EDSR in the context of the full system stack including the discussion of requirements for the application
- fault tolerance analysis of transaction-based processing, which is an important part of EDSR
- empirical comparison of LTMR versus EDSR for circuit area overhead, maximum circuit frequency, and overall system latency due to error correction on a representative system in space-proven technology

In the following sections, we firstly present the reference data processing system, which is used as an example implementation for our approach. In Section 3, we explain LTMR and EDSR and the implementations which are compared. In Section 4, we generalize the processing approach shown in Section 3 and discuss its fault tolerance. Section 5 presents synthesis results based on a known flash-based FPGA. We end the paper with a brief conclusion.

2. Reference architecture

We use a reference model of an on-board data handling unit (OBDH) for satellites [9] for our analysis. Using this example architecture we will explain how EDSR is implemented in particular, because LTMR is mostly architecture-independent. First, we describe an overview of the system, then the FPGA design, and finally the communication protocol between the processor and the FPGA.

2.1. Overview

Fig. 1 shows an overview of the architecture. OBDH 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). We assume that the processor, the communication line between the processor and the FPGA, and the subsystems are sufficiently protected against soft errors.

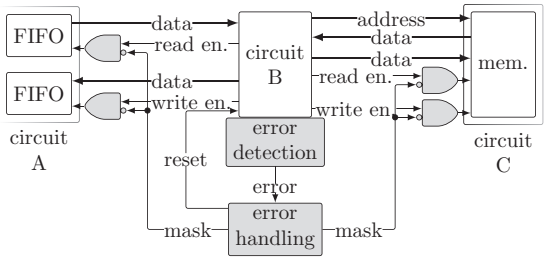

Fig. 2. Excerpt from the FPGA design. Circuit B is hardened by PBED using the gray components. Other circuits are immune to soft errors.

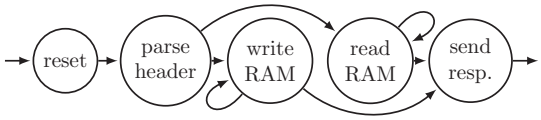

Fig. 3. Simplified state diagram of circuit B, which parses the remote memory packets sent by the mission software (i.e., the processor).

2.2. 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 simplified FPGA model consists of three functional blocks: sequential circuits 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. In Fig. 3, circuit B is shown more in detail. Circuit C with a memory block inside resembles the memory-mapped interfaces. The memories transfer one word per cycle. Circuit A and C including the FIFOs and RAM are assumed to be sufficiently protected against soft errors (e.g., by LTMR and error correcting and detecting code). Circuit B must be hardened by design.

The FIFOs and the memory need a single clock cycle for reading or writing a single word, which enables the masking a single word access operation in the same clock cycle.

2.3. Communication protocol

The communication protocol between the processor and the FPGA is visualized in Fig. 4. The protocol consists of two kinds of messages: *request* and *response*, which both make up a single transaction. The processor sends memory access requests for a specific address or address interval to the FPGA and the FPGA (more precisely, circuit B answers with the according response: A read request is responded with read data and a write request is acknowledged after the write operation. Every request is acknowledged with a response and a second request cannot be sent before the response to the first request has been received. If the FPGA does not respond after a timeout, e.g., due to a soft error, the last request is repeated.