Reconfiguration Network Design for SEU Recovery in FPGAs

Ediz Cetin
School of Electrical Engineering & Telecommunications
University of New South Wales, Sydney, Australia
e.cetin@unsw.edu.au

Oliver Diessel, Lingkan Gong, Victor Lai
School of Computer Science & Engineering
University of New South Wales, Sydney, Australia
{odiessel, lingkang, victorl}@cse.unsw.edu.au

***Abstract*— *Field-Programmable Gate Array* (FPGA) systems provide an ideal platform for meeting the computation requirements for future on-board processing. FPGAs, however, are 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 detail the design of a reconfiguration network that provides the infrastructure to enable SEU recovery in FPGAs. The reconfiguration network's structure and operation is detailed along with performance analysis using results from simulated and implemented designs. The results indicate that total error recovery time from SEUs is dominated by the reconfiguration delay, and that the communication delay of the reconfiguration network is relatively small.**

Keywords*— *Fault tolerance; radiation induced errors; reconfiguration network; reconfigurable hardware

I. Introduction

Burgeoning international interest in the development of space missions based on low-cost satellites stimulates the demand for new approaches to digital processing based on *Commercial Off-The-Shelf* (COTS) reconfigurable *Field-Programmable Gate Arrays* (FPGAs). The main challenge to deploying FPGA-based systems in space is dealing with radiation-induced errors that affect the configuration memory implementing the application circuits. Recovery from radiation-induced configuration errors involves error detection, reconfiguration, and state restoration. There are a number of approaches that have been reported in the literature dealing with *Single Event Upsets* (SEUs) [1] – [7]. While these approaches detail methods for recovering from SEUs, the infrastructure needed to support these approaches is cursorily mentioned. In this paper we detail the design, implementation and performance analysis of one such network to support SEU recovery in FPGAs. This network is designed as part of our research activity looking at rapidly recovering from SEUs in FPGAs by utilizing *Triple Modular Redundancy* (TMR), combined with dynamic partial reconfiguration to recover from configuration errors in a single module while the remaining modules of a TMR component continue to operate [7].

The paper is organized as follows: Section II details the reconfiguration network design with Section III detailing the voter design. Performance analysis is provided in Section IV with concluding remarks and future work given in Section V.

II. Reconfiguration Network

An FPGA-based reconfigurable system utilizing TMR for protection against radiation induced errors requires infrastructure for communicating reconfiguration requests which may arise from any voter in the system to the *Reconfiguration Controller* (RC). The *Reconfiguration Network* (RN) detailed in this paper is intended to interconnect every voter with a central RC that manages the reconfiguration of corrupted modules. Crucially, the network is required to cross clock domains to handle requests from voters operating at different clock frequencies (multirate systems) and must scale to manage requests from tens to hundreds of components.

A. Reconfiguration Network Structure

In our approach we propose employing a multirate token ring network between multiple voters and a single central *Internal Configuration Access Port* (ICAP) based RC. Voters continuously check whether the modules they are checking are in agreement or not. When a disagreement in the outputs persists, they initiate a reconfiguration request for the module in error, as it is likely that the configuration of the module has been corrupted by an SEU. As depicted in Fig.1, the RN connects all voters with the RC in a ring. The darkly shaded circle represents a voter that has the token and is therefore permitted to communicate with the RC. In the network, messages are sent in one direction only, clockwise in our design. The network uses 5 message types for communicating as detailed in Table I.

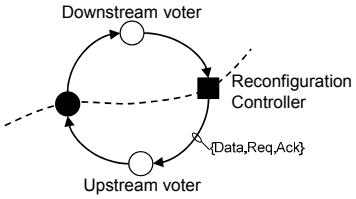


Fig. 1. Reconfiguration Network

Communications between sending and receiving nodes are multirate since they may be operating at different clock frequencies. In particular, the reconfiguration port, ICAP, must be operated at a maximum frequency of 100 MHz, which may be slower than many application circuits. Links therefore incorporate a request-acknowledge pair of handshake signals and employ a chain of 3 synchronization registers to minimize the chance of metastability occurring during the handshake [8]. In addition, a (7,4) Hamming code is used to provide limited

forward error correction for flit transfers across each link [9]. Using this method, 4-bit message packets are encoded with 3 parity bits to form 7-bit flits with the ability to detect and correct single bit errors in the received flit. A further level of error detection is employed at the physical level, whereby handshake signal transitions timeout after 127 local clock pulses and cause a "communications alarm" to be raised. Flits are transferred bit-serially across a single wire. A link is therefore comprised of the output port of the sender, three wires connecting it with the receiver (data, request and acknowledge), and the receiver's input port. Equivalently, a *Network Interface* (NI) is composed of an input and output port and a controller. These are described in detail in the following sections.

TABLE I. RECONFIGURATION NETWORK MESSAGE TYPES

Message Type	Length (flits)	Description
Token	1	Allows the voter to initiate a reconfiguration request and respond to messages from the RC
Reconfiguration Request (RR)	3	Initiated by a voter, comprises a header flit (*RRhdr*) and two module ID flits (*RRmod1*, *RRmod2*)
Acknowledge RR (AckR)	1	Response of the RC if the module ID of a received reconfiguration request is valid
NackR	1	Response of the RC if the module ID of a received reconfiguration request is invalid
Reconfiguration Done (Rdone)	1	Message from RC to voter initiating the reconfiguration request to indicate reconfiguration has been completed

B. Network Interfaces

The NIs comprise Voter and RC interfaces. A voter NI, V-NI, and its ports to the voter logic, as well as its connections to neighboring NIs, is shown in Fig. 2 and a high-level description of the controller FSM is depicted in Fig. 3.

The V-NI receives flits that arrive via its input port from an upstream sender. Each flit is parity checked and latched into the *checked flit* register before its type is determined. Unless the V-NI has received a *Token* flit already and held onto it because the voter has requested reconfiguration, all checked flits are simply passed on to the output port and sent downstream. However, if reconfiguration is requested by the voter, then the *Token* flit won't be passed on. In this case, the V-NI sends a *Reconfiguration Request* (RR) downstream to the NI of the RC. It does so by first sending an *RRhdr* flit. Next, the most significant 4 bits of the module ID of the module to be reconfigured are sent as a 7-bit flit using Hamming (7,4) coding. Finally, the least significant 4 bits are also encoded and sent. Thereafter, the V-NI will only respond to an *AckR* or *NackR* message. Receiving any other type of message will cause a communications alarm to be raised. Such an alarm indicates a fatal error that should halt the system or result in a system reset. If an *AckR* flit is received in response to the issued *RR*, then the only message the V- NI will next accept is an *Rdone* message. On the other hand, if a *NackR* flit was received in response to sending of the *RR*, then a second attempt to send the *RR* is made. If this is not responded to with an *AckR* message, then the alarm is raised. Assuming an *Rdone* message is finally received in response to a *RR*, the voter logic commences resynchronizing the newly reconfigured module in response to the *reconfig_done* signal. In response to receiving the *Rdone* message, the V-NI also sends a *Token* flit to the next downstream voter.

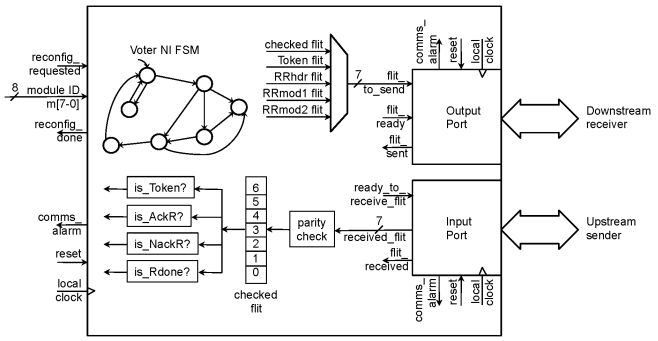


Fig. 2. Voter network interface

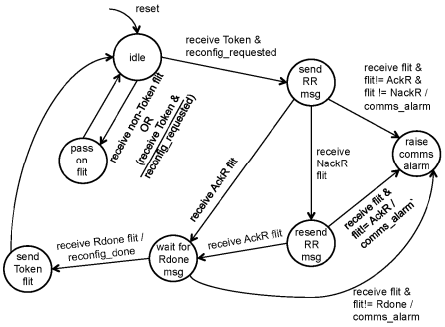


Fig. 3. High-level description of voter network interface controller

The NI of the RC, RC-NI, is depicted in Fig. 4 and a high-level state diagram of the FSM controlling the NI is depicted in Fig. 5. The structure of the RC-NI is very similar to that of the V-NI. However, the behavior, as implemented by the FSM, reflects the role of the RC as the target of reconfiguration requests initiated by voters. In that role, it provides feedback to the initiating voter in the form of an acknowledgement and confirmation that the reconfiguration has been completed.

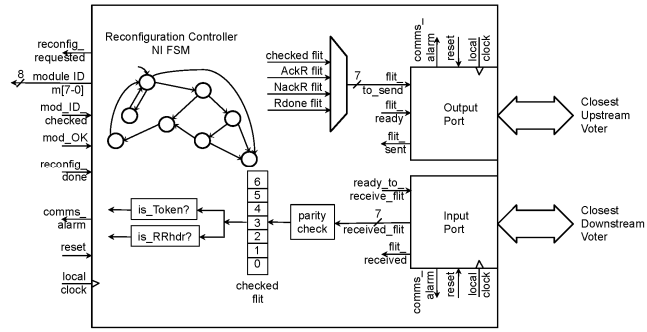


Fig. 4. Reconfiguration Controller network interface

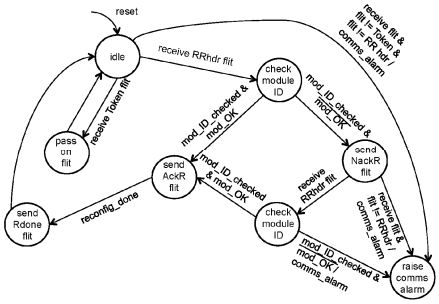
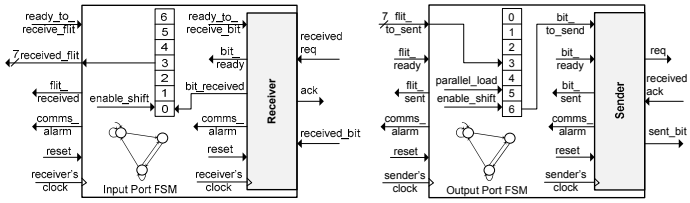


Fig. 5. High-level description of reconfiguration controller network interface controller

The RC-NI raises a communications alarm if any flit other than a *Token* or *RRhdr* is received in the idle state. Tokens are passed on upstream (refer Fig.1). Receipt of an *RRhdr* flit informs the NI that the two following flits should be *RRmod1* and *RRmod2*. These are parity checked and then output to the RC as an 8-bit module ID and output signal *reconfig_requested* is asserted. If the RC logic is able to access its module directory with the given ID, it asserts *mod_ID_checked* and *mod_OK*. In response, the RC-NI sends an *AckR* message to the initiating voter and follows that with an *Rdone* message when the reconfiguration is complete. If the checked module ID was found to be invalid, the RC does not assert the *mod_OK* signal. The RC-NI will then send a *NackR* message to the initiating voter for it to resend its *RR* and module ID. If the resent message is still problematic, the communications alarm is raised, otherwise normal processing continues until an *Rdone* message is sent and the RC-NI returns to the idle state.

C. Communication Ports and Transceivers

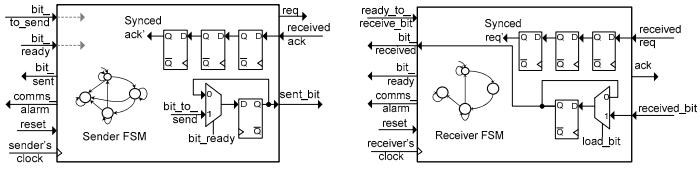
Each NI includes an input and an output port which enable multirate communication with its neighboring interfaces on the token ring. Respectively, these ports receive and send the 7-bit flits that make up a message. While the NI processes incoming flits and determines what type of flit to send on, the ports receive or send each flit bit-serially. The designs of the input and output ports are shown in Fig. 6.



(a) (b)

Fig. 6. NI (a) Input and (b) Output port design

The sender and receiver components are illustrated in Fig. 7. These components feature 3-stage synchronizers [8] to facilitate multirate operation and wait for up to 127 local clock cycles for the *ack'* and *req'* handshake signals to be toggled. A comms alarm is raised to reset the system upon timeouts.



(a) (b)

Fig. 7. (a) Sender component (b) Receiver component

III. VOTER DESIGN

The structure and behavior of the voter logic were outlined in [7]. In this section, the design of the voter logic is elaborated to include the details of its integration with the NI. Fig. 8 shows the voter block logic. The voter fulfills 4 roles:

- *Error detection* – the logic needs to detect errors and distinguish between "transient" and "permanent" errors that require reconfiguration.
- *Reconfiguration trigger* – reconfiguration requests need to be triggered together with the ID of the module to be reconfigured. This module also needs to be isolated from the design so that spurious outputs do not affect ongoing error detection.
- *Ongoing error detection* – a voter alarm needs to be triggered if the remaining functioning modules disagree after a reconfiguration has been triggered.
- *Resynchronization* – after the reconfiguration has finished, the time at which to recommence checking of the outputs needs to be controlled.

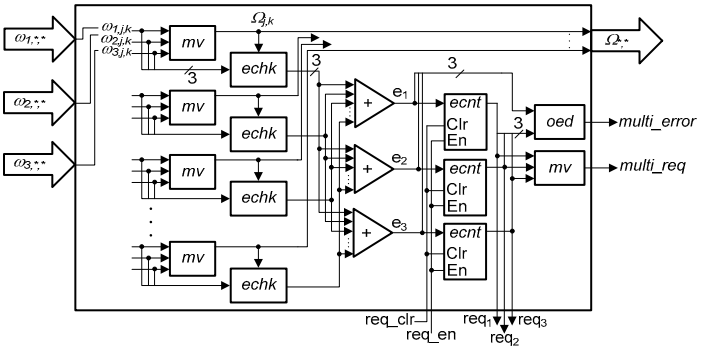


Fig. 8. Voter Block Logic

Error-Detection – We refer to the *k*-th bit of output *j* for module *i* using the symbol $\omega_{i,j,k}$. We use a *majority voter* (*mv*) to form the checked output $\Omega_{j,k} = \omega_{1,j,k} \oplus \omega_{2,j,k} \oplus \omega_{3,j,k}$. The corresponding output of each module can then be checked by reference to the checked output i.e. $\forall i,\ \omega_{i,j,k} \oplus \Omega_{j,k} = e_{i,j,k}$, an error signal for module *i*, output *j,* bit *k*. An *error check* (*echk*) block for each bit *k* of each output *j* produces these signals.

Reconfiguration Trigger – All error signals $e_{i,j,k}$ for module *i* need to be OR-ed together (e_i) to detect the presence of an error in any output of module *i* during the current clock cycle. If the module is acyclic, an error occurring on successive cycles suggests the presence of a permanent error that can only be cleared via reconfiguration. We propose using a 2-bit saturating up/down *error counter* (*ecnt*) to count the number of successive errors for each module. This counter counts up to 3, and stays in state 3 until cleared by signal *req_clr*. This counter also does not underflow below value 0. Between these count values, each module error causes the counter to count up and each non-error causes the counter to count down. While the counter value is in state 3, a reconfiguration request signal req_i is generated for module *i*. A majority voter is also used to check the req_i signals to confirm that multiple reconfiguration requests have not been raised – this situation is flagged by signal *multi_req*. The reconfiguration trigger is enabled (*req_en*) when there is no outstanding reconfiguration request and when none of the modules being checked are being resynchronized. When any of the reconfiguration request signals req_i are raised, the *reconfig_requested* input of the V-NI is asserted and the corresponding *module_ID* is selected and provided to the NI.

Ongoing Error-Detection – After reconfiguration has been triggered, and until resynchronization has completed, the two remaining functioning modules need to be checked differently. Without loss of generality, if module 1 is being reconfigured, differences between the outputs of modules 2 and 3 need to be checked. Note that the majority voters output the correct values even when the outputs from module 1 are incorrect. Hence, the outputs of the module undergoing reconfiguration are not

explicitly isolated from the voter logic. We can also therefore use the OR-ed error signals of modules 2 and 3 (e_2 and e_3) to detect whether either of the remaining modules has also suffered an error. In particular, if $req_1 \cdot (e_2 + e_3)$, as computed by the *ongoing error detector* (*oed*), is found to be true, a multiple error (*multi_error*) condition is triggered that raises a voter alarm. Such a condition indicates multiple errors have occurred within the assumed minimum error period, and the outputs of the system cannot be relied upon to be correct. This condition is checked until resynchronization has completed.

Resynchronization – Arrival of the *reconfig_done* signal from the V-NI causes a resynchronization countdown to commence (loads *resynch_count* into a down counter). The countdown (enabled by signal *resynch_period*) proceeds for a number of clock cycles corresponding to the resynchronization period determined by the designer of the modules for this component. When the countdown has finished, the error counters are cleared and normal error checking recommences.

IV. Performance Analysis

The performance of the reconfiguration network is evaluated by using two representative satellite based signal processing circuits, a 16-bit, 21-tap single accumulator *Finite Impulse Response* (FIR) filter and an 8-to-3-bit, 256-sample *Block Adaptive Quantizer* (BAQ) [10] circuits interconnected with the RN. The set-up is depicted in Fig. 9.

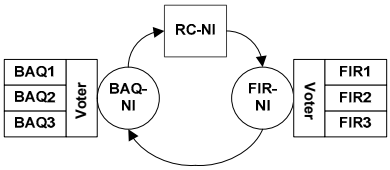


Fig. 9. System layout

The circuits were hand coded in Verilog and targeted at a Xilinx Virtex-5 XC5VFX70T speed grade -2 FPGA. We also tested our implementation on an ML507 board with a slower -1 grade FPGA by halving the clock frequencies of individual modules. Results of the experiments are shown in Table II.

The first 3 rows of the table record the clock frequencies used for the 3 "nodes" of the networked system. The next 3 rows list the simulated and actual transmission delays for a 7-bit flit on each link of the ring network. Then the estimated and actual delays to recover from a permanent FIR module error are presented. In the third section of the table we first report the time taken to detect an error and to initiate a reconfiguration request after waiting for the token to arrive. Next, the time to transmit and receive a 3-flit reconfiguration request at the RC is given. Messages are wormhole routed, and so this time corresponds to the delay for the first flit to arrive plus the delay of transmitting the remaining flits over the slowest link in the ring. The time to check the validity of the request (just 3 clock cycles) and the time to retrieve and write the module's bitstream to the ICAP port is then given. Note that the implemented controller is not optimized, while the simulated controller assumes a bitstream word can be retrieved and written each clock cycle. As can be seen, the total error recovery time is dominated by the reconfiguration delay, and that the communications delay of the RN is relatively small.

TABLE II. Reconfiguration Network Performance

Parameter	Simulation	Implementation
FIR node clock frequency	400 MHz	200 MHz
BAQ node clock frequency	333 MHz	170 MHz
RC node clock frequency	100 MHz	62.5 MHz
FIR-BAQ flit transmission time	278 ns	535 ns
BAQ-RC flit transmission time	541 ns	1,206 ns
RC-FIR flit transmission time	591 ns	1,104 ns
Initiate reconfiguration request time	1,195 ns	3,340 ns
Transmit reconfiguration request msg.	1,901 ns	4,153 ns
Retrieve bitstream & reconfigure	42,030 ns	1,918,864 ns
Transmit reconfiguration done msg.	591 ns	1,104 ns
Re-synchronization delay	1,311 ns	2,625 ns
TOTAL FIR error recovery time	47 µs	1,930 µs
TOTAL communications time	3.7 µs	8.6 µs

V. Concluding Remarks

In this paper we have detailed the design, implementation and initial performance analysis of a reconfiguration network specifically designed for supporting SEU recovery in FPGAs using TMR. To keep the resources used by this network to a minimum, and to ensure high performance, the RN is based on a token ring architecture that connects all voters with the RC in a daisy-chained manner. This architecture ensures a constant port size irrespective of the number of components and ensures few resources are allocated to a supporting function that is not heavily utilized. The generic and extendible nature of the RN makes it suitable for other FPGA SEU recovery schemes.

To date we have evaluated the performance of the reconfiguration network using 3 nodes on the network. We intend stress testing the design and assessing the role of layout in the performance of the network. We are also planning to study the fault tolerance of the network architecture.

References

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

[2] C. Carmichael, "Triple module redundancy design techniques for Virtex FPGAs," *Xilinx App. Note XAPP197*, vol. 1, 2001.

[3] A. Sari and M. Psarakis, "Scrubbing-based SEU mitigation approach for systems-on-programmable-chips," in *Field- Programmable Technology (FPT), 2011 International Conference on*. IEEE, 2011, pp. 1–8.

[4] 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.

[5] 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.

[6] 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.

[7] E. Cetin, O. Diessel, L. Gong and V. Lai, "Towards Bounded Error Recovery Time in FPGA-Based TMR Circuits Using Dynamic Partial Reconfiguration", *International Conference on Field-Programmable Logic* (FPL), September 2013.

[8] Altera White Paper, Understanding Metastability in FPGAs, July 2009.

[9] R.W. Hamming, Error Detecting and Error Correcting Codes, The Bell System Technical Journal 50(2), April 1950.

[10] U. Benz, K. Strodl, and A. Moreira, "A comparison of several algorithms for SAR raw data compression," *IEEE Transactions on Geoscience and Remote Sensing*, 33(5), pp. 1266 – 1276, Sep 1995.