Run-Time Reconfigurable Fault Tolerant Architecture for Soft-Core Processor NEO430

Karel Szurman and Zdenek Kotasek
Brno University of Technology, Faculty of Information Technology, IT4Innovations Centre of Excellence
Bozetechova 2, 612 66, Brno, Czech Republic
Email: {iszurman, kotasek}@fit.vutbr.cz

***Abstract*—Reconfigurable fault tolerant (FT) architecture can be implemented into a SRAM FPGA by using combination of Partial Dynamic Reconfiguration (PDR) and Triple Modular Redundancy (TMR). SRAM FPGAs are susceptible to Single Event Upsets (SEUs) which are the most common transient faults induced by the cosmic radiation. Therefore, SEU mitigation strategy is required when SRAM FPGAs are integrated into safety-critical systems. An essential requirement for these systems is often to remain fail-operational and thus, to perform implemented functionality after the occurrence of a fault. In this paper, we propose a run-time reconfigurable FT architecture based on coarse-grained TMR with triplicated soft-core processor NEO430 core, PDR for removing all transient SEU faults and the state synchronization allowing smooth state recovery from the inconsistent state when the reconfiguration of a failed processor instance was finished into the state where all three processors operate synchronously. The paper describes implemented FT architecture and run-time fault recovery strategy performing all necessary steps without additional blocking of the system functionality. The state synchronization for the soft-core processor NEO430 architecture is described in a further detail. Moreover, the paper presents developed PDR framework used for validation and testing of proposed fault recovery strategy.**

***Index Terms*—TMR, fault recovery, state synchronization, soft-core processor, partial dynamic reconfiguration, SEU mitigation**

I. Introduction

Safety-critical systems are mostly developed as fail-safe systems able to detect any failure and to react on the failure situation by bringing the system into a safe state (e.g. by stopping driving of an autonomous vehicle); or as fail-operational systems whose safe state cannot be easily identified. Fail-operational systems are able to compensate detected failures in order to allow the system to continue in its function or mission (e.g. typically redundant avionics) [8]. These systems frequently utilize some type of processor implementing control algorithms and FPGA performing accelerated computations, signal processing or integrating customized system on a chip design. Many safety-critical systems are also hard real-time systems which produce time-critical outputs used in safety-critical functions (e.g. to control the flight of an aircraft) [7]. Thus, in order to satisfy requirements for hard real-time and fail-operational operating, such systems must be implemented also as fault tolerant. Although SRAM FPGAs are susceptible to Single Event Upsets (SEUs), responsible for transient faults induced by random bit-flip errors in configuration memory cells or in user logic, these programmable circuits are suitable platform to be used in Fault Tolerant Systems (FTSs).

SEU mitigation strategy may combine hardware redundancy and Partial Dynamic Reconfiguration (PDR) in order to implement error detection, self-repair ability and fault recovery mechanism into the system. With respect to the compromise between the system reliability and the resource overhead, various hardware redundancy schemes can be used. The most used form is Triple Modular Redundancy (TMR) which can be applied on different granularity levels in the system design. Coarse-grained TMR and PDR are often combined in one reconfigurable architecture. The time between SEU occurrence and completion of fault recovery become crucial parameter because the reliability of the TMR with one failed replica is worse than the reliability of an unprotected system [13]. The fault recovery process can be generally divided into three phases: 1) fault detection, 2) fault removal by reconfiguration of a region containing replica identified as faulty, and 3) state synchronization bringing the reconfigured replica into the operating state consistent with other correctly operating replicas [5] [6] [10]. While the PDR can be performed run-time, the synchronization may impact availability of the system.

This paper describes proposed run-time reconfigurable FT architecture based on TMR with hardware-implemented state synchronization for soft-core processor NEO430 enabling reliable and non-blocking processor execution in the presence of SEU induced faults and during the fault recovery process.

II. Related work

Various partially reconfigurable architectures and fault recovery strategies have been investigated in recent years. Since TMR has large overhead, another possibility may be employing of less demanding method such as Duplication with Comparison (DWC). Authors in [15] presents an error detection and self-repairing method for partially reconfigurable systems based on fine-grained DWC and PDR at the level of individual frames in FPGA configuration memory. Fine-grained DWC enables high error detection and it has lesser overhead. However, this method has no longer ability for fault masking and thus, the continuous processor execution in the presence of faults is not possible as with TMR.

For reconfigurable TMR, the state synchronization is important part of the fault recovery. According to [5], finding an adequate synchronization strategy implies balancing a trade-off between the speed of the synchronization, extra hardware overhead utilized by synchronization logic and the impact on

978-1-7281-0072-2/19/$31.00 ©2019 IEEE

the critical path of the system. For processors which repeat the same execution periodically or do not require continuous availability, the simplest way for the state synchronization is global reset. However, the main drawback of this method is the loss of global state which can depend on previous program execution. The need for state maintenance can be overcome by implementation of backward fault recovery method or proper state synchronization strategy.

In general, a processor state is given by the content of internal program and data memories; a register file and all internal registers implementing processor logic which are not directly visible for programmers. The largest amount of data which needs to be synchronized is the content of internal memories. With respect to huge resource overhead, the use of a shared memory accesible from all three processor instances is only practice solution, as it is indicated by [5], [6], [12] and [14]. Nevertheless, the critical part of the processor state synchronization is the maitenance of all internal registers located in a CPU architecture by copying of their state from correctly working reference CPUs to the reconfigured one. This requires implementation of the synchronization method directly in hardware to enable access to all registers and to minimize the synchronization time.

Considering only processor registers which are accessible by programmers, the approach described in [12] could be used. The authors proposed fault recovery strategy for 32-bit RISC soft-core processor Plasma protected by TMR. A SEU fault recovery is done by combination of the read-back scrubbing and the processor synchronization. Moreover, possible permanent fault recovery is performed by evacuating the corrupted module into a spare region in FPGA configuration memory. The processor state synchronization is needed in both cases. It is performed by storing and restoring the processor context through shared BRAM memory.

For internal architectural registers which cannot be easily accessed and synchronized by previously mentioned method, the synchronization requires usage of hardware. In [11], we proposed and evaluated serial and parallel hardware synchronization for the reconfigurable fault tolerant CAN bus control system. The serial synchronization was based on the principle of data shifting through internal registers to next unit in the oriented circle topology. The second synchronization approach used two parallel buses for data transfers between registers in reference and synchronized circuits. The principle of the synchronization was based on sequential addressing of each register through an address bus and enabling write or read signals for circuits which are active during the synchronization process. Reference circuit transfers the content of its addressed registers to the data bus byte after byte while the synchronized circuit reads these data from the bus and stores them into its internal registers.

In [3], an extensive fault injection campaign and an analysis of SEU effect on soft-core processor LEON3 shown that the most of faults that led to the processor's malfunction were caused by the set of critical registers (mainly by program counter) and ALU operands. Moreover, it was found that only a third of injected faults propagates to the CPU interfaces. Authors in [4] also identified the register file as the most critical CPU structure during their soft-error vulnerability assessment for commercial ARM Cortex-R5 processor, which is extensively used in real-time safety-critical applications. We have not investigated reliability of the soft-core microcontroller NEO430 CPU so far. However, these results confirm the need for CPU core protection and fast fault recovery.

III. Run-time Reconfigurable FT Architecture

Our PDR design includes soft-core microcontroller NEO430 with the CPU core protected by reconfigurable TMR architecture. In the TMR architecture, the same input signals are shared between all CPU instances and their output signals are brought into the majority voters. Each TMR voter is enhanced by additional error detection logic for identification of a failed CPU instance. The design architecture is shown in Fig. 1.

The FPGA design floorplan is divided into the static and the dynamic area. The static area includes Generic Partial Dynamic Reconfiguration Controller (GPDRC), Internal Reconfiguration Access Port (ICAP) core, flash controller for communication with a parallel flash containing stored partial bitstreams, microcontroller memories and all peripherals shared between triplicated CPUs, TMR majority voters and the PDR framework for validation of fault recovery experiments. In the dynamic area, the replicated CPU instances are placed into corresponding Partial Reconfiguration Modules (PRMs).

The GPDRC [9] is previously developed reconfiguration controller based on ICAP interface. It supports detection and correction of single appearing transient faults caused by SEUs and the mitigation of permanent faults by reducing FT architecture scheme and changing its layout in utilized PRMs.

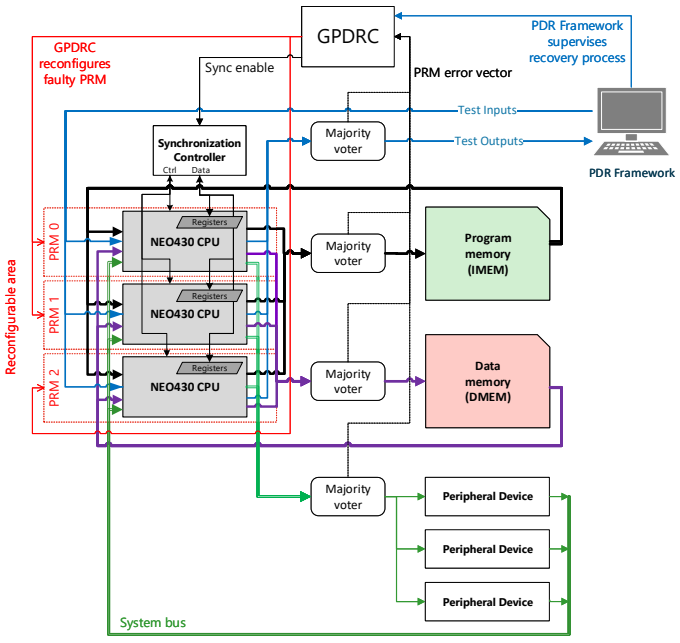

Fig. 1. Reconfigurable FT architecture

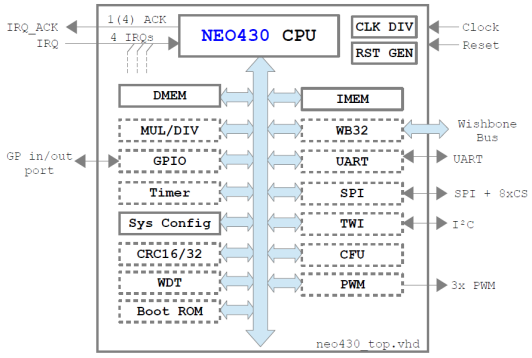

Fig. 2. Soft-core microcontroller NEO430 architecture [1]

The NEO430 microcontroller has customizable 16-bit CPU compatible with TI MSP430 [1] [2]. The CPU, program memory (IMEM) and data memory (DMEM), and all peripherals are interconnected through a system bus. The architecture is is shown in Fig. 2. An instruction execution is conducted by performing several micro operations, thus the CPU needs several consecutive cycles to complete a single instruction. The execution of micro operations is controlled by the control arbiter (CA) which generates control signals for the data path. The data path includes the register file, the Arithmetic Logic Unit (ALU) and the address generator (AG). When the NEO430 CPU is reconfigured, the internal registers within these components have to be addressed during state synchronization. The register file incorporates sixteen 16-bit registers. Program Counter (PC/R0), Stack Pointer (SP/R1), Status Register (SR/R2) and Constant Generator Register (CG/R3) have dedicated functions. R4 to R15 are general purpose registers.

Fault recovery implemented in the hardware is shown in Fig. 3. In the experiments, the reconfiguration of the PRM corresponding to the faulty CPU is started based on the PRM error vector generated by TMR voters which can detect a mismatch on the compared CPU outputs. A test application executed by triplicated CPUs periodically checks the digital input indicating the Sync enable request for performing the state synchronization. The synchronization request is generated from GPDRC after the reconfiguration of failed CPU instance is finished.

The state diagram in Fig. 4 shows processors execution during the normal state and the fault recovery process. After the reconfiguration, repaired CPU is restarted. During its startup, a program reads the digital inputs and checks if request for synchronization is active. Since the request was activated by GPDRC, the CPU switched into the SLEEP mode as well. When the program executed by operating CPUs is in a state suitable for synchronization, it will indicate readiness for the hardware synchronization through processor digital output to a synchronization controller. This is a special circuit responsible for parallel addressing of all synchronized registers and their copying from the correctly working CPUs to the recovered one. Afterwards, operating CPUs go into the SLEEP mode.

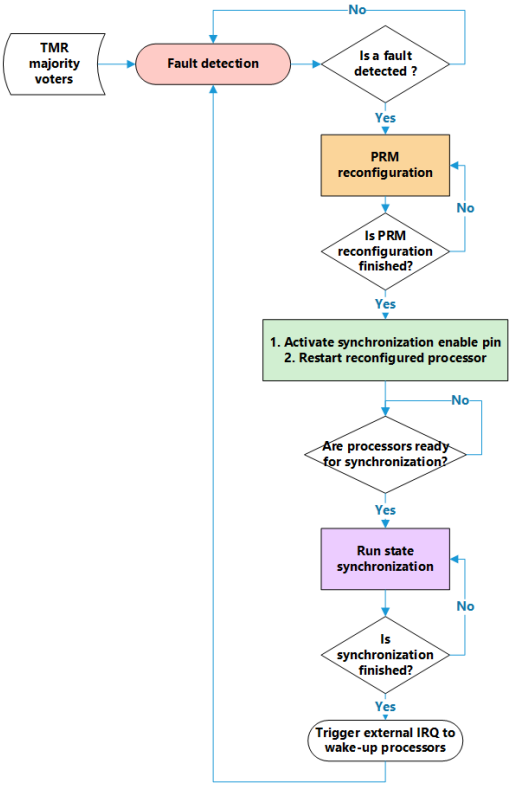

Fig. 3. Processor fault recovery implemented in hardware

In this state, CPUs are in an idle state IFETCH 0 waiting for an external IRQ generated by the synchronization controller which will activate normal operating mode. In parallel with the previous step, the synchronization controller performs synchronization procedure of all architectural registers (PC, SP, SR, CG, GP R4-R15) and all processor internal registers (MAR, IR, ALU SRC and DST) by their copying from correctly working CPUs into the faulty one, while all CPUs are idle in the SLEEP mode. After the hardware synchronization phase is finished, the external IRQ signal is triggered to bring all CPUs from the SLEEP mode to the operating mode. All three CPUs continue in the program execution from the synchronized state as all internal registers required for correct program execution and architectural registers were synchronized.

IV. Experiments and Validation

For experimental validation of the proposed solution for runtime fault recovery, we implemented PDR Framework which allow following validation activities: (1) to communicate with a test application executed on the protected processor, (2) to monitor and measure duration of various phases during the recovery process and (3) to validate the design functionality. The PDR Framework is based on another NEO430 microcontroller, called Debug Core, which is placed in the static FPGA area. The Debug Core is interconnected with protected processor through Wishbone bus in point-to-point topology as it is shown

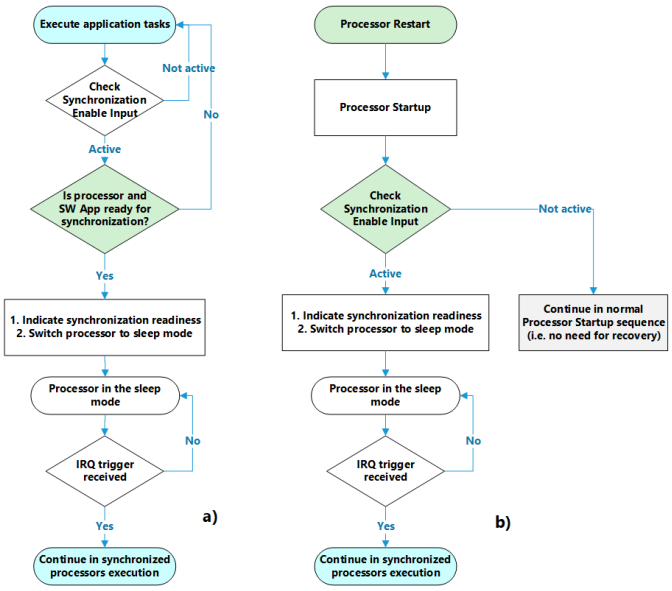

Fig. 4. Processor operating during normal state (a) and the fault recovery (b)

in Fig. 5. The Debug Core implements a monitoring application which can be controlled via serial command line interface. This interface allows to read/write data from/to internal status and control registers within the PDR Framework which are memory-mapped into area accessible through Wishbone bus communication. Moreover, the PDR Framework also supports setting of inputs which then passed to the test application or receiving the test outputs useful for verification of correct program execution of the protected processor during fault recovery simulation.

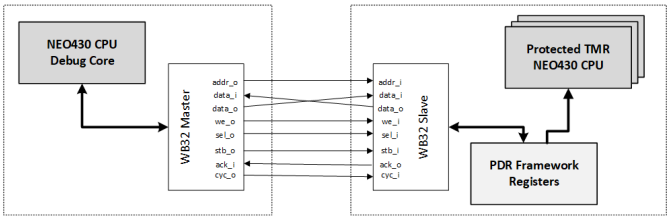

Fig. 5. Wishbone interconnection between Debug and protected CPU cores

V. Conclusions and Future Research

Three main concerns for processor state synchronization in proposed architecture exists. The implemented state synchronization strategy for reconfigurable TMR NEO430 CPU with respect to the microcontroller architecture and real-time execution of software tasks was designed with following considerations: (1) the synchronization of an application data context stored in internal program and data memories is implicitly done by concurrent writing output data through majority voters and consequent reading data back, (2) the CPU core peripherals are shared by the same way as memories, (3) we were focused only on synchronization of internal CPU architecture.

In this paper, we proposed run-time reconfigurable FT architecture for the soft-core processor NEO430 and the state synchronization strategy allowing smooth transition to the state when all triplicated processors operate synchronously after the failed processor instance was reconfigured. Furthermore, we described the recovery process validation method based on use of PDR Framework. Our future research will be focused on the design and evaluation of the state synchronization methods implemented in the hardware.

Acknowledgment

This work was supported by The Ministry of Education, Youth and Sports from the National Programme of Sustainability (NPU II) project IT4Innovations excellence in science – LQ1602 and the BUT project FIT-S-17-3994.

References

[1] S. Nolting, “The NEO430 Processor”, github.com/stnolting/neo430.
[2] Texas Instruments Inc, “MSP430x1xx Family – User’s Guide”, SLAU49F, 2006.
[3] R. Travessini, P. R. C. Villa, F. L. Vargas, E. A. Bezerra, “Processor core profiling for SEU effect analysis”, IEEE 19th Latin-American Test Symposium (LATS), Sao Paulo, 2018, pp. 1-6.
[4] X. Iturbe, B. Venu, E. Ozer, “Soft error vulnerability assessment of the real-time safety-related ARM Cortex-R5 CPU”, IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT), Storrs, CT, 2016, pp. 91-96.
[5] U. Kretzschmar, J. Gomez-Cornejo, A. Astarloa, U. Bidarte, and J. Del Ser, “Synchronization Of Faulty Processors In Coarse-Grained TMR Protected Partially Reconfigurable FPGA Designs”, Reliability Engineering & System Safety, 2016, vol. 151, pp. 1–9.
[6] A. Morillo, A. Astarloa, J. Lazaro, U. Bidarte, J. Jimenez, “Known-blocking synchronization method for reliable processor using tmr & dpr in sram fpgas”, in VII Southern Conference on Programmable Logic (SPL), Cordoba, Arg., April 2011, pp. 57–62.
[7] C. Spitzer, U. Ferrell, “Ferrell, T.: Digital Avionics Handbook”, CRC Press, Boca Raton, FL, USA, 2015, ISBN 978-1-4398-6861-4.
[8] H. Kopetz, “Real-Time Systems, Design Principles for Distributed Embedded Applications”, Kluwer Academic Publishers, USA, 2002, ISBN 0-792-39894-7.
[9] L. Miculka, Z. Kotasek, “Generic Partial Dynamic Reconfiguration Controller for Transient and Permanent Fault Mitigation in Fault Tolerant Systems Implemented Into FPGA”, in 17th IEEE Symposium on Design and Diagnostics of Electronic Circuits and Systems, Warszawa, PL, 2014, ISBN 978-0-7695-5074-9, pp. 171–174.
[10] E. Cetin, O. Diessel, L. Gong, V. Lai, “Towards bounded error recovery time in FPGA-based TMR circuits using dynamic partial reconfiguration”, International Conference on Field programmable Logic and Applications, Porto, 2013, pp. 1-4.
[11] K. Szurman, L. Miculka, and Z. Kotasek, “Towards a State Synchronization Methodology for Recovery Process after Partial Reconfiguration of Fault Tolerant Systems”, in 9th IEEE International Conference on Computer Engineering and Systems, Cairo, 2014, pp. 231-236, ISBN 978-1-4799-6593-9.
[12] M. Fujino, H. Tanaka, Y. Ichinomiya, M. Amagasaki, M. Kuga, M. Iida, T. Sueyoshi, “Fault Recovery Technique for TMR Softcore Processor System Using Partial Reconfiguration”, Springer, ICA3PP, 2012, vol. 7439, pp. 392–404, ISBN 978-3-642-33078-0.
[13] M. Schütz, A. Steininger, F. Huemer, J. Lechner, “State Recovery for Coarse-Grain TMR Designs in FPGAs Using Partial Reconfiguration”, IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT), Chicago, IL, USA, 2018, pp. 1-6.
[14] A. Morillo, A. Astarloa, J. Lázaro, U. Bidarte, J. Jimenez, “Reliable microprocessors for FPGAs: State of the art and trends”, International Conference on Applied Electronics, Pilsen, 2010, pp. 1-6.
[15] M. S. Reorda, L. Sterpone, A. Ullah, “An Error-Detection and Self-Repairing Method for Dynamically and Partially Reconfigurable Systems”, in IEEE Transactions on Computers, vol. 66, no. 6, pp. 1022-1033, 1 June 2017.