Proxy Circuits for Fault-Tolerant Primitive Interfacing in Reconfigurable Devices Targeting Extreme Environments

Adewale Adetomi
Ewireless Research Group, School of Engineering
University of Edinburgh
Edinburgh EH9 3JL, United Kingdom
a.adetomi@ed.ac.uk

Sangeet Saha
Embedded and Intelligent Systems Lab
University of Essex
Colchester CO4 3SQ, United Kingdom
sangeet.saha@essex.ac.uk

Klaus McDonald-Maier
Embedded and Intelligent Systems Lab
University of Essex
Colchester CO4 3SQ, United Kingdom
kdm@essex.ac.uk

Tughrul Arslan
Ewireless Research Group, School of Engineering
University of Edinburgh
Edinburgh EH9 3JL, United Kingdom
t.arslan@ed.ac.uk

***Abstract*—Continuous interface access to device-level primitives in reconfigurable devices in extreme environments is key to reliable operation. However, it is possible for a primitive's interface controller, which is static to be rendered non-operational by a permanent damage in the controller's circuitry. In order to mitigate this, this paper proposes the use of relocatable proxy circuits to provide remote interfacing capability to primitives from anywhere on a reconfigurable device. A demonstration with device register read controller shows that an improvement in fault-tolerance can be achieved.**

Keywords—reconfigurable device; reliable; primitive interfacing; extreme environment; network on chip

I. INTRODUCTION

Reconfigurable hardware devices like FPGAs have become more attractive for computing, thanks to their combination of sheer hardware performance and (re)programmability, the latter especially useful in applications in extreme environments (e.g., space and nuclear) where it allows key fault-tolerant features to be implemented (e.g., scrubbing and relocation [1]) for adaptation and recovery from errors.

To take advantage of ASIC-like performance and power consumption, reconfigurable devices often come with specialized functional modules or primitives built directly into the silicon. These fixed structures often provide specialized functionalities like analogue-to-digital conversion; and device-level functionalities like configuration memory access and clock management. To access these primitives, a user design implements a controller around the primitive's interface to gain access to control the core's functionalities through key read-write operations.

The subject of this paper is relates to the fault tolerance of such a controller. While a soft error in the primitive's controller has a transient effect and can be easily masked by Triple Modular Redundancy (TMR) [2] and fixed by scrubbing [3], the same cannot be said of a hard error, which renders the affected area permanently damaged. If this area falls within a critical section of the controller, it could cripple its key functionalities. A potential solution in this situation is circuit relocation, which is the movement of a task from one location to another on the device while maintaining its functionality and preserving its inter-task communication capability and timing properties.

In the event of a permanent chip damage, provoked by radiation, for instance, circuit relocation would allow the controller to be moved to a damage-free area on the chip in order to circumvent the permanent damage. However, because the controller requires access to the primitive, which is in a fixed location on the chip (e.g., the internal configuration access port in an FPGA [4]), a mechanism that would ensure a continuous access to the primitive in the case of an emergent hard error is required.

This paper presents the implementation of a fully functional and relocatable controller for device register read. Because a primitive is in a fixed location on the chip, a special technique is required to decouple a controller from the primitive it controls while preserving the cross-interface functionalities between the controller and the primitive. We introduce proxy circuits to abstract the original interface access between a controller and its corresponding primitive. To the best of our knowledge, this is the first time a mechanism like this will be reported in the open literatures.

Circuit relocation often generates the question of how to provide dynamic communication for relocated circuits and how to preserve timing, among others. This paper presents our mitigation approach to the circuit relocation challenges and describes the implementation of an example proxy circuit for device register access for diagnosis in an FPGA. We further discuss the benefits of our approach to fault tolerance and relate the implications on resource utilization and interface access latency.

II. Meeting the Requirements for Full Relocatability

For a circuit to be relocatable from one location to another on the FPGA, a set of stringent requirements must be met, namely: the matching of the source and destination regions in terms of resource type and relative positions, including the interconnect structure; the provision of dynamic communication between the relocated circuit and the other circuits on the FPGA; and the preservation (or rather, avoidance) of any static interconnections routes crisscrossing the target region. A practical implementation of a relocatable circuit must put these requirements into consideration. Here, we describe what mechanisms we are adopting to meet these requirements.

A. Matching Target Location

It is easier to meet this requirement in FPGAs that have homogeneous resources and those that have heterogeneous resources arranged at regular column intervals. It is salient to state that with newer FPGAs like the 7 series and the UltraScale, the architecture of the chip area is more heterogeneous and irregular. This means it is more difficult to find matching locations on the horizontal plane. Vertical matching locations are easier to come by because there is homogeneity at the column level of all known Xilinx FPGAs. In general there is usually a matching location for any circuit on the FPGA, not considering other circuits that may be on the FPGA. However, in reality, other circuits could have taken up potential matching locations, in which case other techniques of relocation like the memoization-based method in [5] can be used, though with limitation in the data width and with possible timing issues (see Section II.D), but nonetheless, a promising solution to an impossible situation.

B. Provision of Dynamic Communication

Circuit relocation allows a circuit to be moved from its original location where it has established communication routes with other circuits on the FPGA. Current design tools do not natively support re-establishing these communication routes after relocation. Indeed, it is possible to use online routing to re-establish communication between a relocated circuit and the other circuits on the chip but this is computationally expensive, often requiring tens of thousands of clock cycles [6] and far from being trivial. Another possibility is to use the reconfiguration mechanism to transport data, an idea exploited in [7]. In this case, the I/Os of the original circuits are abstracted by a task interface logic, with on-chip memories used to store input and output data, which are then transferred through the configuration layer via the Internal Configuration Access Port (ICAP). However, this can be counterintuitive in a situation where the primitive being controlled is the ICAP itself in which case one would need the controller in order to gain access to the ICAP, but a relocatable controller is intended to be detachable and relocatable away from the ICAP. This would create a conundrum if the physically connected controller-ICAP combination were to be used to provide dynamic communication between the controller and the ICAP itself. Because the controller is relocated away from the ICAP, data needs to be transferred between the controller and the ICAP. However, according to [7], to transfer data using the ICAP, the controller needs to be attached to the ICAP.

In order to provide a dynamic communication that is applicable to a generic primitive, we employ the mechanism advanced in [8], where the clock buffers of an FPGA are adapted for dynamic communication in a scheme called Clock-Enabled Low-Overhead Communication (CELOC), with a corresponding Clock-Enabled Relocation-Aware Network on Chip (CERANoC) advanced to support online relocation. Fig. 1 shows the implemented star-shaped CERANoC network for this work. The clock nets of an FPGA run through dedicated wires in the silicon and as such, provide a side channel for routing data regardless of where circuits are or relocated to on the FPGA in runtime. Further details can be found in [8].

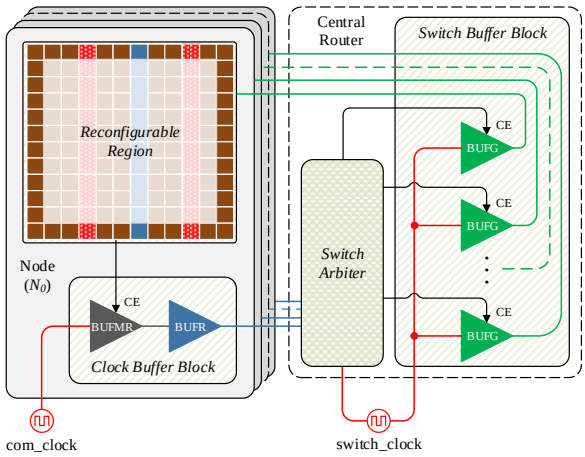

Fig. 1. A start-shaped CERANoC network using [8]

C. Preservation of Existing Interconnect Routes

The target location for a relocated circuit is not guaranteed of being free from interconnect routings that belong to a neighbouring circuit. If the relocated circuit is reconfigured in the location without regard to these pre-existing routings, it could override the routings and break the functionalities of the nearby circuits. A method of preservation of pre-existing routings would be to read back the interconnect configuration information of the target region and compare with those of the relocated circuit, but care has to be taken to ensure that these pre-existing routings are avoided if their path would conflict with the routing in the relocated circuit. A good allocation algorithm search should put this into consideration in finding a suitable place for relocation in the first place.

The task here is made more difficult because FPGA design tools generally allow circuits to use routing resources that are outside the partitions assigned to them, mostly because of the need to resolve routing congestion. A way of reducing routing congestion especially at the interface of circuits is to use bit-serial interconnections between circuits, as this has been shown to have reduced footprint and congestion factors [9].

The adaptation of clock buffers for communication calls for the adoption of bit-serial interconnect owing to the limited availability of clock buffers in a typical FPGA [8]. This use of a bit-serial communication implementation nevertheless is beneficial because it helps in easily meeting the requirement for the preservation of existing routes, while at the same time garnering the other benefits of bit-serial over bit-parallel interconnects, which include high speed and power savings as demonstrated in [9][10].

D. Dynamic Timing Closure

When a circuit is moved away from the original location, for which its static timing closure has been ensured, certainty has to be provided that the circuit will continue meeting timing in the new location. To ease timing closure, it is recommended that the inputs of a relocatable circuit are registered [4]. Once a relocatable circuit meets timing within its bounded partition at design time, for the purpose of relocation, it can be expected to maintain its timing closure within its own physical bound regardless of wherever it is relocated on-chip in runtime so far a strictly matching target location is used. Timing issues are related more to the interface between the static logic and the relocatable circuit. As the circuit is moved about, its physical distance from the static interface changes and thus its interface timing changes. This timing change has to be taken into consideration.

In [11], a worst-case interface timing analysis is used, where every possible location for a relocation circuit is analysed offline and the worst-case timing used to characterise the circuit for online relocation. With our use of the clock buffers for communication, the interface's static timing is not violated because the clock network is on a dedicated network different from the general interconnect. With the communication interface well characterised – the latency is known deterministically for a given clock frequency [8] and from the point of view of the relocatable circuit, any delay is incurred as communication latency rather than setup/hold time delay. All static/hold time delays related to the communication interface are already covered by the network on chip.

E. Provision of Clock Network at the Target Location

The design tools do not route clock signals to regions of the device that are not used by user designs. As such, if a circuit is relocated in runtime, a suitable clock network must be provided at the target location. The clock networks of the FPGA are routed through switch matrices. These routing resources can be controlled online by activating specific bits in the configuration memory. This approach is exploited in [12] to provide online clock routing for relocatable circuits. For instance, the authors manipulate regional clock buffers in runtime to provide different clock frequencies to tasks and to also switch away from failed clock networks.

To simplify the process of clock routing, the clock tree routing information that pertains to each relocation region can be extracted from the static design and added to the relocatable circuit at design time as done in [13]. Alternatively, dummy primitives (e.g., flip-flops) can be placed into the floorplan and connected to global clock lines at compile time, forcing the place and route tool to route the required clock network to predetermined regions that will be used for relocation [14].

III. Proxy Circuits for ICAP Access

In order to demonstrate the proposed concept, a device register read circuit is implemented. Fig. 2 is a diagrammatic representation of a static non-relocatable ICAP controller for device register read (RGR) access, which is a capability that can be used to diagnose an FPGA. For instance, if a configuration error occurs, the status (STAT) register of the device can be queried to know the source of error. This controller is comprised of a Finite State Machine (FSM) and a Command Sequence ROM (CSROM). It should be noted that after issuing the RGR command and prior to desynchronising the interface, a readback event occurs where the 32-bit register value is presented at the output of the ICAP deterministically three clock cycles after asserting the ICAP's enable.

To read a register, a series of commands must be issued to the configuration interface [15] (see Fig. 3). The preamble commands are used to synchronise the interface before reading a register, and then the postamble commands are used to desynchronise it. These commands are kept inside CSROM, implemented from distributed memory. The FSM sends the commands to the ICAP.

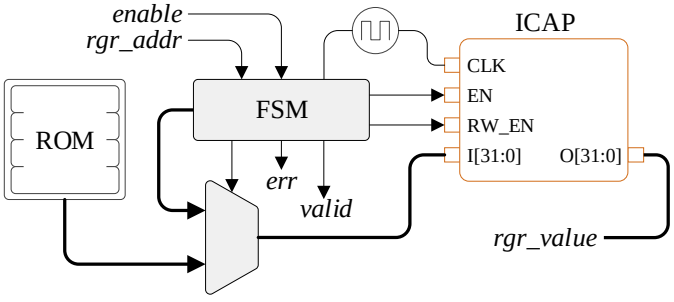

Fig. 2. A static non-relocatable ICAP controller with direct connections between the controller and the ICAP interface

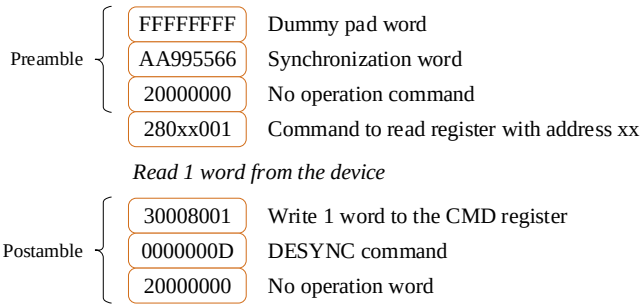

Fig. 3. Command template for reading device registers

In order to allow the controller to be relocatable, the entire controller is broken into multiple modules that can communicate via the unique CELOC-based communication mechanism. We call these modules proxy modules as they are able to interact as if they were directly connected even though

they can be far apart on the FPGA. In the two proxy circuits, the interconnection of the clock buffers allows a global start-shaped communication. There are various clock buffer combinations that can be used [8]. However, to allow the majority of the chip area to be available for other circuits to be implemented, the outer left and right edges of the clock regions are preferred for placing (or relocating) the proxy circuits. This implies that a clock buffer connection of [BUFMR → BUFR → BUFG] (necessitated by allowable clock-to-clock connections [16]) has to be used to access the clock network-based communication network.

A. *ICAP's Proxy Circuit*

This circuit abstracts the interconnections between the ICAP controller and the ICAP (see Fig. 4). It receives input signals from the controller and streams them over the CERANoC network to the controller's proxy attached to the ICAP. The ICAP proxy module serializes the parallel data (*en*, *rw_en*, and *data_in*) and encodes it for transmission through the clock buffers. The register read commands are transmitted over several clock cycles. The returned register value is received via the clock network (the data riding on *data_clock_in*).

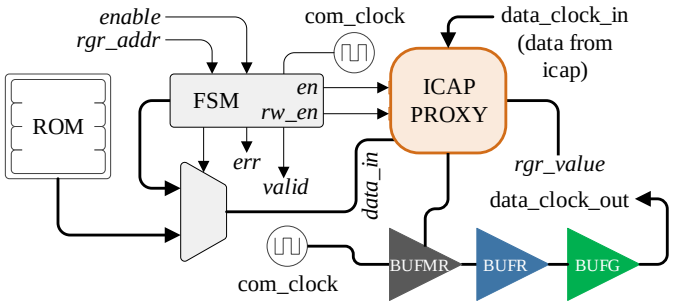

Fig. 4. The ICAP interface's proxy circuit

B. *Controller's Proxy Circuit*

The controller's proxy receives signals sent from the ICAP's proxy and applies them to the ICAP (see Fig. 5). Since the ICAP registers input data on the rising edge of the clock when *EN* is asserted [15], it is important to ensure that the application of inputs to the ICAP is synchronised to the rate of data transmission over the network. To achieve this, the *EN* port is gated and controlled by an internal signal that ensures that *EN* takes the value set by the remote controller only when a complete packet has been received.

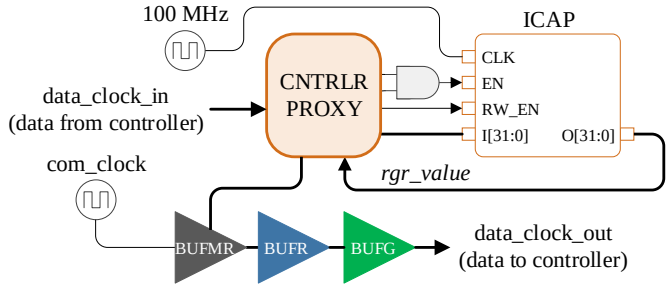

Fig. 5. The ICAP controller's proxy circuit

IV. Result and Evaluation

The demonstration is carried out on an Artix-7 FPGA. The proxy circuits are flushed left and right against the outer edges of the clock regions to allow other circuits on the FPGA to have maximum available area, and thereby minimising internal fragmentation. For comparison, a static version of the controller is also implemented. Tables I and II present the comparative numbers for resource overhead and device read latency.

The obvious limitation of the introduction of proxy circuits is the increase in resource overhead and the impact on read latency owing to the time taken to transmit the data signals over the dynamic communication network and the special synchronisation adopted in the controller's proxy circuit. However, there is a relocation (and by extension, reliability) improvement of at least, (N-1)×, where N is the number of clock regions in the device. In the Artix-7 device used for demonstration, this translates to 500% improvement in reliability.

TABLE I. Resource Utilization Comparison

Module	**No Proxies**		**With Proxies**	
	Flip Flops	***LUTs***	***Flip Flops***	***LUTs***
Controller	73	105	73	105
ICAP's Proxy	-	-	255	87
Controller's Proxy	-	-	256	92
Total	73	105	584	284
Total (Slices)	**27**		**73 (2.88×)**	

TABLE II. Register Read Latency Comparison

Communication Clock's Frequency (MHz)	**Read Latency (µs)**	
	No Proxies	***With Proxies***
100.00	0.26	15.6
150.00	0.26	10.4
171.43	0.26	9.10

V. Summary

This paper has presented the use of proxy circuits to improve reliability in primitive interfacing in reconfigurable devices operating in extreme environments. A relocatable device register read controller has been demonstrated, with improved reliability, but incurring higher resource overhead and latency compared to a static implementation. However, relocatability is of immense benefit in critical applications, and the costs in terms of speed and resources, are easily offset when one considers that a hard error in a static primitive interface controller could cripple the controller, with knock-on effects on other dependent system functionalities.

VI. Acknowledgement

This work is supported in part by the UK Engineering and Physical Sciences Research Council (EPSRC) through grants EP/R02572X/1 and EP/P017487/1.

REFERENCES

[1] A. Adetomi, G. Enemali, and T. Arslan, 'A fault-tolerant ICAP controller with a selective-area soft error mitigation engine', in *2017 NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, 2017, pp. 192–199.

[2] R. E. Lyons and W. Vanderkulk, 'The Use of Triple-Modular Redundancy to Improve Computer Reliability', *IBM J. Res. Dev.*, vol. 6, no. 2, pp. 200–209, Apr. 1962.

[3] B. Bridgford, C. Carmichael, and C. W. Tseng, 'Single Event Upset Mitigation Selection Guide'. 2008.

[4] Xilinx Inc., 'Vivado Design Suite User Guide, Partial Reconfiguration - UG909 (v2018.1)'. Xilinx Inc., 2018.

[5] G. Enemali, A. Adetomi, G. Seetharaman, and T. Arslan, 'A Functionality-Based Runtime Relocation System for Circuits on Heterogeneous FPGAs', *IEEE Trans. Circuits Syst. II Express Briefs*, vol. 65, no. 5, pp. 612–616, May 2018.

[6] A. DeHon, R. Huang, and J. Wawrzynek, 'Hardware-assisted fast routing', in *Proceedings. 10th Annual IEEE Symposium on Field-Programmable Custom Computing Machines*, 2002, pp. 205–215.

[7] X. Iturbe, K. Benkrid, T. Arslan, R. Torrego, and I. Martinez, 'Methods and Mechanisms for Hardware Multitasking: Executing and Synchronizing Fully Relocatable Hardware Tasks in Xilinx FPGAs', in *2011 International Conference on Field Programmable Logic and Applications (FPL)*, 2011, pp. 295–300.

[8] A. Adetomi, G. Enemali, and T. Arslan, 'Enabling Dynamic Communication for Runtime Circuit Relocation', *IEEE Trans. Very Large Scale Integr. VLSI Syst.*, pp. 1–14, 2019.

[9] N. Kapre, 'On Bit-Serial NoCs for FPGAs', in *2017 IEEE 25th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2017, pp. 32–39.

[10] A. Morgenshtein, I. Cidon, A. Kolodny, and R. Ginosar, 'Comparative analysis of serial vs parallel links in NoC', in *2004 International Symposium on System-on-Chip*, 2004, pp. 185–188.

[11] A. Lalevée, P. H. Horrein, M. Arzel, M. Hübner, and S. Vaton, 'AutoReloc: Automated Design Flow for Bitstream Relocation on Xilinx FPGAs', in *2016 Euromicro Conference on Digital System Design (DSD)*, 2016, pp. 14–21.

[12] X. Iturbe, K. Benkrid, R. Torrego, A. Ebrahim, and T. Arslan, 'Online clock routing in Xilinx FPGAs for high-performance and reliability', in *2012 NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, 2012, pp. 85–91.

[13] A. A. Sohanghpurwala, P. Athanas, T. Frangieh, and A. Wood, 'OpenPR: An Open-Source Partial-Reconfiguration Toolkit for Xilinx FPGAs', in *2011 IEEE International Symposium on Parallel and Distributed Processing Workshops and Phd Forum*, 2011, pp. 228–235.

[14] C. Beckhoff, D. Koch, and J. Torresen, 'Go Ahead: A Partial Reconfiguration Framework', in *2012 IEEE 20th International Symposium on Field-Programmable Custom Computing Machines*, 2012, pp. 37–44.

[15] Xilinx Inc., '7 Series FPGAs Configuration, User Guide - UG470 (v1.13.1)'. Xilinx Inc., 2018.

[16] Xilinx Inc., '7 Series FPGAs Clocking Resources - User Guide UG472 (v1.11.2)'. Xilinx Inc., 2015.