Microprocessors and Microsystems xxx (2012) xxx–xxx

Contents lists available at SciVerse ScienceDirect

Microprocessors and Microsystems

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

TeMNOT: A test methodology for the non-intrusive online testing of FPGA with hardwired network on chip

Muhammad Aqeel Wahlah [a,*,1], Kees Goossens [b,2]

[a] Computer Engineering Department of Technical University of Delft, The Netherlands
[b] Electrical Engineering Faculty in Technical University of Eindhoven, The Netherlands

ARTICLE INFO

Article history:
Available online xxxx

Keywords:
Non-intrusive
Online test
FPGA
Hardwired NoC

ABSTRACT

Modern Field Programmable Gate Arrays (FPGAs) posses small feature sizes, and have gained popularity in mission-critical systems. However, FPGA can suffer from faults due to the small feature sizes and harsh external conditions that are faced by a mission-critical system. Therefore, the architecture of FPGA must be tested to ensure a reliable system performance. At the same time, due to the mission-critical nature of a system, the test process should be *non-intrusive*, i.e., applications and FPGA regions that are not being tested remain unaffected. An online test methodology is, therefore, required that not only verifies the reliability of FPGA architecture, but also does not degrade the performance of other, running FPGA applications.

In this paper, we propose an online test methodology that uses hardwired network on chip as test access mechanism, and conducts test on a region-wise basis. Importantly, the proposed test methodology exhibits a non-intrusive behaviour that means it does not affect the applications and FPGA regions, which are not being tested, in terms of configuration, programming, and execution. Our test methodology posses approx. 32 times lower fault detection latency as compared to existing schemes, respectively.

1. Introduction

Field Programmable Gate Arrays (FPGAs) have evolved from a simple programmable logic device to complex platform-based devices [1]. In recent years, FPGAs have emerged as target architectures to implement System-on-Chip (SOC) designs, e.g., in the fields of medicine [2], radio astronomy [3], and high performance computing [2,4], etc. The SOCs can execute multiple applications and in different usecases. FPGA architecture implements an SOC design by using the two physical planes: logic and configuration as shown in Fig. 1. The logic plane executes the desired application(s), whereas the configuration plane (re)configures the desired application on the logic plane. Conventionally, the logic plane consists of an array of different types of input output blocks and logic blocks, i.e., configurable logic blocks (CLBs), Block RAMs, and processor units (e.g., PowerPC) etc., as shown in Fig. 1A. The logic plane also contains an interconnection network, which consists of routing channels and switch-boxes, to connect the logic blocks. The configuration plane architecture [5], on the contrary, comprises *memory cells*[3] to store the configuration bits (or bitstream), and the *configuration circuit* to load the bitstream in the memory cells, as shown in Fig. 1B. Before proceeding, we define the terminology that will be used in the paper.

1.1. Terminology

A *Usecase* defines the set of applications that execute in parallel. Typically a *soft* Intellectual Property (IP) is mapped on FPGA reconfigurable computational blocks, e.g., CLBs. An IP is *hardwired* or *hard* when it is directly implemented in silicon, e.g., PowerPC. We define *(re)configuration* or *loading* as the installation of new functionality in FPGA by sending a bitstream to a reconfiguration region. A (soft or hard) IP is *programmed* after it is configured, if necessary, which entails changing the state of its registers when it is in functional mode. *Online testing* verifies the function of FPGA chip while the system on FPGA is operational.

1.2. Problem description

FPGA architectures, which belong to the deep sub-micron regime and use transistors as small as 40 nm [1], can suffer from faults. The chances of FPGA to become faulty increases, if an FPGA is used

* Corresponding author.
E-mail addresses: m.a.wahlah@tudelft.nl (M.A. Wahlah), k.g.w.goossens@tue.nl (K. Goossens).
[1] PhD Scholar in the Computer Engineering Department of Technical University of Delft, The Netherlands.
[2] Full Professor in Electrical Engineering Faculty in Technical University of Eindhoven, The Netherlands.

[3] For our paper, we refer to the class of FPGAS [1] that use static random access memory (SRAM) cells to implement the required (combinational or sequential) functionality.

0141-9331/$ - see front matter © 2012 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.micpro.2012.05.011

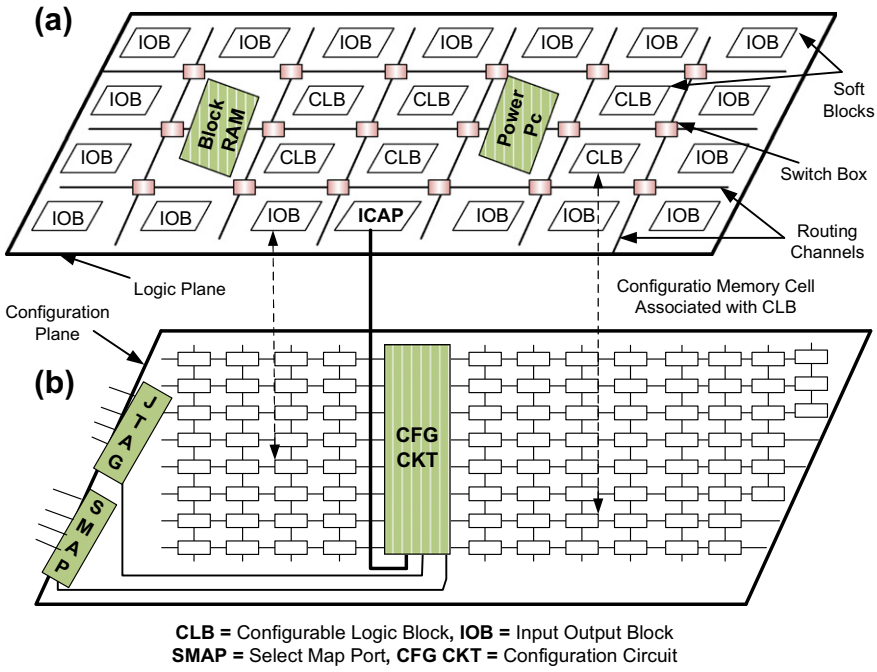

Fig. 1. Architecture of FPGA (a) logic and (b) configuration planes [5].

in a mission-critical system that is exposed to harsh external conditions (e.g., cosmic radiations) [6]. The radiations can flip the bits in the memory cells of the configuration plane. The flipped/wrong values of the memory cells are critical to handle, because these can be propagated to the configurable logic blocks (CLBs) and interconnection network of FPGA architecture [7]. This can cause the logic plane element to *permanently stuck-at* wrong values, until appropriate detection and correction measures are taken.

Online testing is enforced to detect faults in FPGAS [8–11]. Typically, online testing requires test access mechanism (e.g., boundary scan infrastructure (BSI) [12]) to verify the reliability of the target architecture. The online testing can be categorised as *functional* or *structural*. *Functional test* verifies FPGA architecture for the intended set of applications [8], whereas *structural test* verifies FPGA architecture irrespective of the intended set of applications [10]. Meanwhile, due to the mission-critical nature of an FPGA system, online test cannot be performed on the whole FPGA chip simultaneously [11]. Conventionally, it is performed after dividing FPGA into multiple (logical) regions, which can be (i) a single configurable logic block (CLB) [9], or (ii) a single/multiple columns of CLBs [11]. This means, at a certain point in time, some FPGA regions are under test (RUT) and some regions are not under test (RNUT). Consequently, the behaviour of an FPGA-based system can be viewed from two aspects, i.e., system behaviour in RUTs and system behaviour in RNUTs.

An online test scheme, however, has to overcome a number of challenges. First, an online test scheme should be non-intrusive to the applications in RNUTs, which means (a) programming and execution of SoC application that reside in RNUTs is not disrupted, and (b) (if required) bitstream of new SoC applications is configured in RNUTs. In other words, during the online test, the normal operation of RNUTs is not affected in terms of *configuration*, *programming*, and *execution*. Second, an online test scheme should have *high performance in terms of insignificant fault detection latency* (FDL). The FDL is the amount of time that an online test scheme takes to detect faults in a region under test, and ideally it should be 1 cycle. At the same time, an online test scheme should have *low cost in terms of insignificant spatial and temporal overheads*. Here, *spatial overhead* represents the additional test hardware [13], e.g., test pattern generators (TPGs) and output response analysers (ORAs), to perform the online verification of an FPGA region. The TPGs and ORAs are made up of FPGA reconfigurable resources, i.e., CLBs and interconnection wires, etc. [14]. The *temporal overhead* represents the amount of time that is required to configure the spatial overhead.

Our online test methodology meets the above mentioned challenges. The proposed methodology uses *hardwired network on chip as test access mechanism*, and conducts *test on a region-wise basis*. A region in our methodology is termed as test configuration functional region (TCFR), as we shall explain in Section 4. The proposed test methodology exhibits *a non-intrusive behaviour*, which means it allows the test process in parallel with the configuration, programming, and execution of applications in RNUTs. Moreover, our online test methodology performs *test when an application is invoked*, which ensures that application always execute on a reliable architecture. The nature of the *test is structural*, which ensures a high percentage of fault detection for the target FPGA architecture. In addition, the proposed scheme has *reduced spatiotemporal overheads*, because it does not make use of TPGs and ORAs for generating and analysing test sequences. Instead, the proposed scheme uses the connections through the hardwired NOC to access and analyse the architecture of a particular region in FPGA.

In the remainder of this paper, we explain the existing online test schemes in Section 2. Afterwards, we explain the basic idea of our test methodology in Section 3.1, and present the motivation for such a scheme in Section 3.2. We then move onto explain the target platform that is being tested (Section 4). It is followed by explaining the design flow and methodology to perform the online testing in Section 5. We present results and analysis in Section 6. Lastly, we conclude in Section 7.

2. Related work

In the literature, a number of online test schemes [13,8,17,9,15,10,11,18] have been proposed to detect faults in FPGA architecture. In the following discussion, we explain each of these individually.

Please cite this article in press as: M.A. Wahlah, K. Goossens, TeMNOT: A test methodology for the non-intrusive online testing of FPGA with hardwired network on chip, Microprocess. Microsyst. (2012), http://dx.doi.org/10.1016/j.micpro.2012.05.011

In [13], the authors introduce the concept of roving STARs, where a STAR is a self testing area that is comprised of TPG, ORA, and circuit under test. The online testing is structural which roves the STARs periodically to test every section (that can comprise multiple CLB columns) of the FPGA architecture. The TAM mechanism used by the scheme is the conventional boundary scan infrastructure [12].

The authors in [8] make use of built in self test (BIST) to detect operational faults in the system. The scheme in [8] applies the test sequence periodically to the circuit under test (CUT), and checks the CUT responses to detect the existence of operational faults. The online testing is functional, which can test multiple CLBs simultaneously. The authors, however, have not mentioned the TAM mechanism for the proposed method.

In [9,15], the authors apply the concept of active replication for the online testing of CLBs. This active replication method enables the relocation of each CLB functionality without halting the system, even if the CLB is part of an executing application. In [9] the authors have applied the structural testing, whereas in [15] the authors have performed functional testing of the target FPGA. However, in both the schemes the test is performed at a single CLB level by making use of the conventional boundary scan infrastructure [12].

The authors in [10] propose a new CLB architecture for FPGAs and associated online testing, and reconfiguration techniques that detect configuration faults in the CLBs of FPGAS. The test scheme is structural, which can detect single or multiple faults in an FPGA. The authors, however, have not mentioned the TAM mechanism for the proposed method.

The authors in [16] provide an error mitigation technique that is based on modular redundancy and time redundancy. It uses duplication with comparison (DWC) and concurrent error detection (CED). The test scheme is functional, i.e., application dependent, which requires suitable encoding and decoding functions for testing the CLBs. The authors, however, have not provided any details about the TAM.

The testing in [11], like the conventional online test schemes, is based on fault-scanning method. The scheme is applicable to bus-based FPGA architectures, and it assumes that certain parts of the FPGA are fault-free. The online testing is structural, which is performed at the granularity of multiple CLBs. The authors, however, have not provided any details about the TAM.

The authors in [18] claim to provide a faster online test scheme as compared to [13]. The scheme is based on roving tester (ROTE), which tests parts of the circuit by duplication and comparison manner. The testing is intended to detect possible circuit functions (applications) with a test granularity that can range from single CLB column to multiple CLB columns. The authors, however, do not specify the TAM for their approach.

From the above discussion, we can conclude that the existing schemes scan through the FPGA chip to find out the perspective faults [13,8,9,15,10,11,18]. In these schemes, a relatively small portion of FPGA chip is taken off-line, while allowing the rest of FPGA to continue its normal operation. The region to be tested is replicated on an already verified portion of the device, before being taken off-line and tested. Testing in these schemes is accomplished by roving the test functions, i.e., test pattern generator, output analyser, and region under test bitstream, across the entire FPGA. These schemes, as illustrated through Table 1, use a conventional boundary scan infrastructure (BSI) as their TAM [13,9,15]. The nature of online testing can be structural [13,9,15] or functional [8,17,9,18]. The schemes can have different levels of test granularity, ranging from a single CLB [15,9,16] to multiple CLB columns [13,11,18]. The main focus of the existing online test schemes has been to maximise the percentage of fault detection with the minimum fault detection latency. Additionally, none of these schemes, due to the limitations of the existing TAM architecture, can achieve the interleaved test and configuration operations for multiple applications, Table 1 (Column 4). This means, each one of these induces a level of intrusiveness, which could be a partial or a complete application, Table 1 (Column 5).

Table 1
Our work positioning w.r.t. existing online approaches. Abbreviations, NA: not available, Col: column, St: structural, Fu: functional, P/C: partial/complete.

Scheme	TAM	Test type	Test granularity	Test and load	Intrusiveness level
Abramovici [13]	BSI	St	CLB Col	No	(P/C) App.
Al-Asaad [8]	NA	Fu	CLB Col	No	(P/C) App.
Gericota [15]	BSI	St	CLB	No	(P/C) App.
Gericota [9]	BSI	Fu	CLB	No	P. App.
Lima [16]	NA	St	CLB	No	(P/C) App.
Reddy [10]	NA	St	CLB	No	(P/C) App.
Shnidman [11]	NA	St	CLB Col	No	(P/C) App.
Dutt [17]	NA	Fu	CLB Col	No	(P/C) App.
Verma [18]	NA	Fu	CLB Col	No	P. App.
Our online test methodology	HW-NoC	St	TCFR	Yes	None

3. Basic idea and motivation

In this section we explain the basic idea and motivation of our online test methodology.

3.1. Basic idea

In this paper, we propose the online test methodology that posses non-intrusive nature and:

1. uses hardwired NoC as test access mechanism (TAM),
2. tests undisrupted in parallel with other application(s) configuration, programming, and execution,
3. tests at application startup, i.e., before the configuration of an application,
4. uses structural test to find faults in the FPGA architecture,
5. posses reduced spatial and temporal overheads with respect to the conventional online test schemes.

The discussion on each of these points is as follows.

(1 and 2) HWNoC as TAM: We test FPGA after dividing into multiple test configuration functional regions (TCFRs).[4] The online testing is performed at the granularity of TCFRs. Our online test methodology uses HWNoC [19], as the test access mechanism, to verify the FPGA architecture. The architecture of HWNoC can transport four types of data, i.e., test, configuration, and functional (programming and execution) data. The data is transported by using the connections that are allocated at compile time, but are created and terminated at run time. Moreover, the HWNoC architecture can ensure non-intrusive data transportation, which means all types of data (test, configuration, programming, and execution) that flows through the same HWNoC do not produce interference with each other. This means while testing, it is possible to achieve (a) un-disrupted execution for already executing application(s), and (b) the operations of configuration, programming, and execution for new applications.

(3) Test at application startup: The test procedure is triggered at an application startup time. We assume that multiple applications can not occupy the same TCFR(s) simultaneously. However, an application can execute in multiple TCFRs. In our online test methodology, application TCFRs are tested in prior to configure an application, because the associated TCFRs are not tested during the execution of an application. This in turn avoids the disruption in

[4] The architecture of TCFR is explained in Section 4.2.

Please cite this article in press as: M.A. Wahlah, K. Goossens, TeMNOT: A test methodology for the non-intrusive online testing of FPGA with hardwired network on chip, Microprocess. Microsyst. (2012), http://dx.doi.org/10.1016/j.micpro.2012.05.011

the execution of application, which can be caused by the test procedure. On the negative side, an application configuration is delayed until an application TCFRs are not tested. Additionally, a fault that occurs during an application execution time can only be detected after an application finishes its execution.

(4) Structural test: Our online methodology performs the structural test and currently detects the *permanent stuck-at faults*. A logic block or wire is said to experience a *stuck-at fault* when its logic value always stays at 1 or 0 and cannot be reversed. The structural test provides us with the increased reusability, which can be exploited in multiple ways. (a) A single test suite (bitstreams and stimuli for an FPGA TCFR) irrespective of the intended set of applications that run on the TCFR. This mean the test bitstream for a single TCFR can be reused for multiple TCFRs. (b) A single test (over a certain period of time) for multiple time-multiplexed applications executing on the same TCFR(s). This means the result of a structural test can be reused for multiple applications that execute on same TCFR(s).

(5) Reduced spatiotemporal overheads: Our online test methodology uses the system manager, which uses the HWNoC connections to analyse the structural faults in a TCFR. It is important that the SM can execute on a programmable hardware processor, e.g., PowerPc. The *additional resources* are, therefore, in the form of test connections and the code to perform the analysis. In our methodology, no specialised test hardware (e.g., TPGs and ORAs) are imported to the FPGA logic plane. This in turn eliminates the spatiotemporal overheads that are required to place and configure the test hardware on the logic plane of FPGA.

In the next section we build the motivation behind using the above-mentioned parameters for our methodology.

3.2. Motivation

For the motivation purpose, we consider SoC with 2 applications as shown in Fig. 2A. Fig. 2B shows the placement of applications in the logical regions of FPGA, where IPs of both the applications communicate with each other by using a functional interconnect (e.g., bus or NoC). To build the motivation of our methodology, we examine the possible shortcomings that can arise while testing the SoC by using the conventional test schemes [9,18].

A region under test (RUT) in existing schemes, e.g., [18] can be considered as a set of wires and CLBs. The conventional schemes scan through FPGA to find out faults [9]. This means testing is performed by roving an RUT across the chip. RUT1 in Fig. 2C and RUT2 in Fig. 2D indicate two such roving instances. In Fig. 2D, *Tested1* indicates the region that was earlier tested by RUT1. Conventionally, the region under test is taken off-line, while allowing the rest of FPGA to continue its normal operation. However, in case an application (e.g., A1) is already executing on such a region (e.g., RUT2). Then, the state and functionality of that application (i.e., A1) is first replicated and saved in a region that is free and had passed the test (e.g., *Tested1*). Additionally, the FPGA interconnection is routed accordingly for intra-application execution. In Fig. 2E, time T2 indicates such a situation, when RUT2 testing is triggered. However, the process first stops A1 for saving the states and rerouting the interconnection wires as shown in Fig. 2E. The above discussion motivates us to *trigger the online test at an application startup time*. By doing so, we can *avoid intrusiveness* by ensuring that an application always executes at the pretested regions.

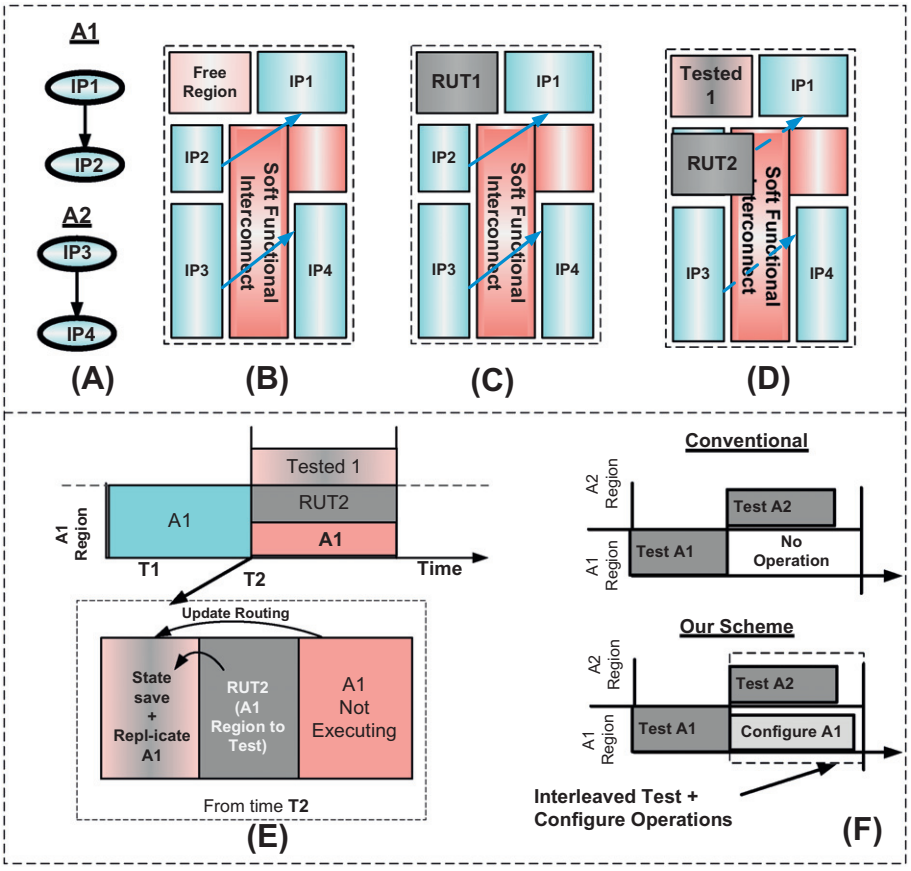

Fig. 2. (A) applications with task graphs, (B) applications on FPGA, (C) and (D) regions under test in FPGA, (E) test process details from spatial and temporal point of views, (F) abstract comparison of our methodology with conventional schemes.

Please cite this article in press as: M.A. Wahlah, K. Goossens, TeMNOT: A test methodology for the non-intrusive online testing of FPGA with hardwired network on chip, Microprocess. Microsyst. (2012), http://dx.doi.org/10.1016/j.micpro.2012.05.011

In the conventional test schemes, application IPs and associated functional interconnect both coexist in the same reconfigurable plane [9,18], as shown in Fig. 2B. Therefore, while testing an application (i.e., A1), an RUT can belong to IP plus functional interconnect (e.g., RUT2). In this situation, the other application (i.e., A2), whose region is not aimed for testing, can get disrupted. It is because of the unavailability of the *soft* functional interconnect for the A2 traffic, during the state saving and replication period of A1. The broken virtual connection in Fig. 2D illustrates this situation. Hence, it motivates us to have applications and the functional interconnect in *disjoint planes*, which is ensured by hardwiring the functional interconnect (i.e., HWNoC) in FPGA.

Additionally, the existing approaches [9,18] use boundary scan infrastructure (BSI) [12], which occupies the FPGA configuration circuit while testing. This means the inherent nature of the TAM in these schemes does not allow the configuration (of another application) in parallel with the ongoing test process. Therefore, restricting the parallel operations of test and configuration for multiple applications. For instance in Fig. 2F, the parallel operations of A1 testing and A2 configuration (on a pretested region) are not possible with the existing schemes, such as [9,18]. Hence, imposing a delay on either, test or configuration, process. This motivates us for *TAM which does not interfere* with the bitstream loading of an application.

Moreover, FPGA-based SOCs can comprise multiple time-multiplexed applications, where developing and exercising a test suit for each application is prohibitive due to the economic and time constraints. Therefore, it motivates us to perform the *structural test*, which not only posses application independent nature but also ensures maximum percentage of fault detection.

In the next Section 3, we explain the our FPGA, whose architecture is tested by following the principles that have been mentioned in the current section.

4. Target FPGA architecture

Our architecture comprises FPGA with a (i) hardwired communication plane [19], and multiple (ii) test configuration functional regions (TCFRs), as shown in Fig. 3. In further discussion we explain these individually.

4.1. Hardwired NoC architecture

The communication plane of our FPGA is a hardwired Network on Chip. Its consists of routers (R) and network interfaces (NIs). The routers in our design are hardwired and posses the architectures as detailed in [20], whereas a Network Interface is further split into NI kernels and NI shells. NI kernel is hard and its architecture is explained in [19,21]. On the other hand NI shell, which de (serialises) the incoming/outgoing data, is soft because IPs can vary in terms of width and number of ports and depth of FIFOs, etc.

There must be at least one IP that can programme the system. This can be a CPU that bootstraps the system by programming the HWNoC, or a hard (secure) boot module [22,23]. In our FPGA system, we make use of a *Control processor* to bootstrap the system, as shown in Fig. 3.

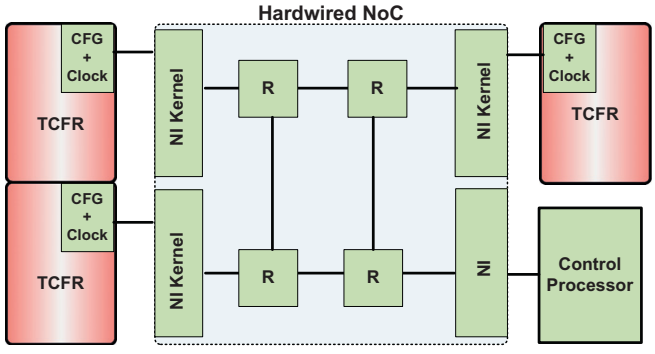

Fig. 3. Showing FPGA with HWNoC.

4.2. Test, configuration, and functional region architecture

The architecture of a *test configuration functional region* (TCFR) is shown in Fig. 4A. This shows that each TCFR, like the conventional FPGA logic plane, comprises an array of CLBs that are connected through interconnection network. There is a local *Clock manager* that is memory-mapped, i.e., programmable clock frequency for the required CLB can be generated by writing to the registers that are accessible through the HWNoC. Similarly, a memory-mapped *Reset controller* is present by using which the application IPs are enabled or disabled from processing the input data. Importantly, the TCFRs are not divided into multiple disjoint regions, and are connected to each other by using the BUS MACROS (not shown here to keep things simple). This means TCFRs are not isolated at functional level, which allows a soft IP to span in multiple TCFRs. Towards limitations, the logic plane of a TCFR currently does not posses special computational hardware, e.g., MAC and DSP units, etc.

Additionally, each TCFR has its own configuration architecture. The configuration architecture operates on the configuration bitstreams that are transported via the HWNoC. The configuration architecture has been extended (and modelled in SystemC) in the paper to perform testing of the FPGA architecture, as shown in Fig. 4B. For this purpose, it uses a port to write and read-back the bitstream. In addition it contains the elements, i.e., address decoder, dedicated registers, and a 1-word wide *data-bus* to write the incoming bitstream in the desired CLB location. It is important that in our architecture a column of 16 CLBs defines the minimum configuration region (MCR). We use HWNoC to send bitstream to test/configuration port, and for this purpose a real-time connection with fixed latency and guaranteed bandwidth is used. A header register is used to store the bitstream header, whereas the frame data register is used to store the bitstream frames, as shown in Fig. 4B. The frame data register is large enough to store the bitstream frame of 41 words [24]. The address decoder, which is connected to each MCR, enables the respective MCR to write the bitstream. The controller is a finite state machine, and performs the specific functions of; (i) receiving the bitstream from the NI shell, (ii) activating the demultiplexer to forward the received data either to the header register or to the frame data register, and (iii) shifting the data right in frame data register to load in the *data-bus*. The detailed explanation of controller function is explained in Section 5.3.

In the next Section 5, we elaborates the design flow and methodology to perform the online testing of FPGA architecture.

5. Design flow and methodology

In this section we explain our online test methodology to find out the faults in the logic elements of TCFRs, as shown in Fig. 5. Currently, we evaluate the reliability of logic elements in TCFRs. However, the technique proposed in [25] can be used for the verification of our HWNoC routers. For the convenience of the reader, we briefly describe the process of online testing of HWNOC in Section 5.3.4.

Please cite this article in press as: M.A. Wahlah, K. Goossens, TeMNOT: A test methodology for the non-intrusive online testing of FPGA with hardwired network on chip, Microprocess. Microsyst. (2012), http://dx.doi.org/10.1016/j.micpro.2012.05.011

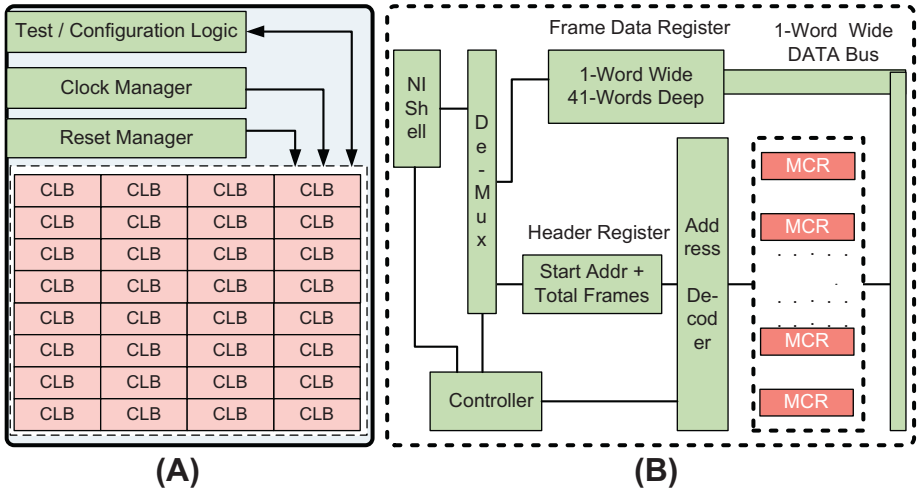

Fig. 4. (A) TCFR detailed architecture, and (B) TCFR architecture to write the (test) bitstreams.

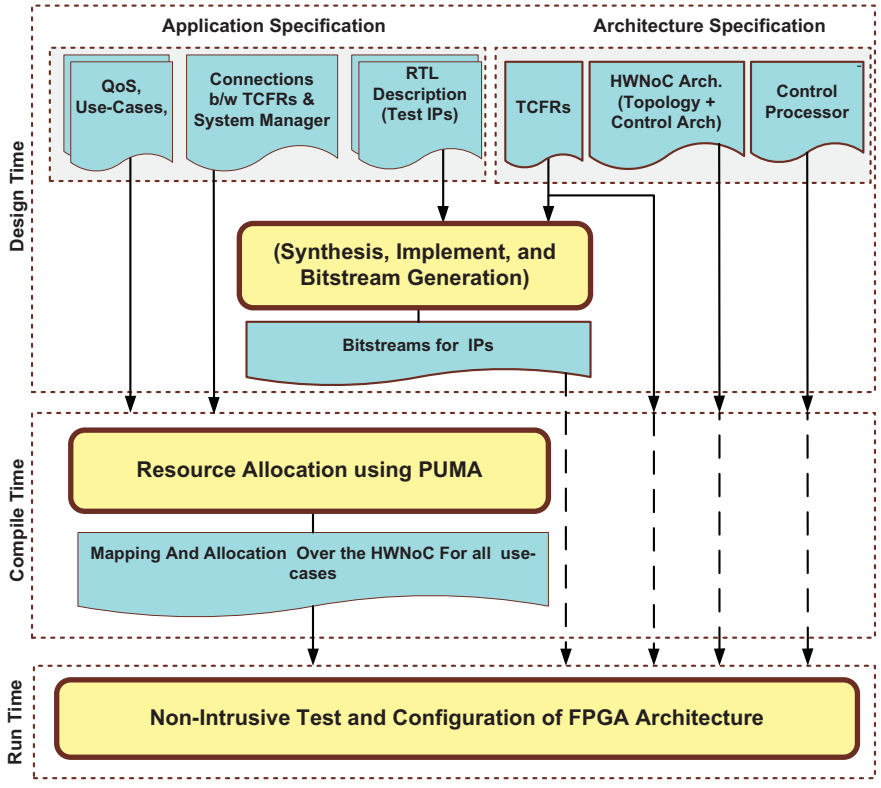

Fig. 5. Interaction between the design, compile, and run time flows.

To meet the objectives of Section 3.2, our online test methodology takes into account design, compile, and run time phases that are explained in the following sections.

5.1. Design time phase

At design time, we provide the (FPGA) architecture and application specifications, as shown in Fig. 5. From the *architecture point of view*, the specifications are (i) the dimensions and architecture of hardwired NOC and test configuration functional regions (TCFRs), and (ii) the Control processor, which is a programmable hardwired IP. From the *application point of view*, the specifications include (i) the hardware description of two test IPs, and (ii) the system application.

Initially, the hardware description of the *test IPs* is fed to the synthesis, implement, and generate bitstream tools. The bitstream generation tool, as a result, outputs the bitstreams of the target TCFR architecture. Currently, our design flow supports relocatable bitstreams that can be transported to (any) required TCFR to locate the stuck-at faults. The *system application* performs the online testing of the target FPGA architecture and, as well as, the configuration of user-application(s) in the desired FPGA region(s). The system

Please cite this article in press as: M.A. Wahlah, K. Goossens, TeMNOT: A test methodology for the non-intrusive online testing of FPGA with hardwired network on chip, Microprocess. Microsyst. (2012), http://dx.doi.org/10.1016/j.micpro.2012.05.011

application can be characterised as $SA = (IP, C)$. Here IP stands for the TCFRs, and the system manager (SM) that is mapped to the programmable *Control processor*. On the other hand C represents the connections from the SM to TCFRs. The connections are used to transport the bitstream to the target TCFR architecture. The resources of each connection are reserved during the compile time, as explained in the next Section 5.2.

5.2. Compile time phase

The inputs of this phase include (i) HWNoC architecture, (ii) the system application connections with required Quality-of-Service (QoS) constraints, and (iii) the usecases of the system application. The inputs are fed to the resource allocation tool named PUMA [26], as shown in Fig. 5. PUMA goes through the system application and maps the test ports of a TCFR to the respective NI. This is followed by allocating the connection between the SM and a TCFR. The process is repeated until the connections between the SM and all the TCFRs of FPGA architecture are not allocated.

It is important that while testing, the behaviour of the existing applications (in terms of configuration, programming, and execution) should not be affected in the regions that are not under test. This requires a virtual platform for our online test methodology, so that it can exercise a non-intrusive behaviour to the other operations. To ascertain such a virtual platform, the resources (paths, QoS and flow control) should be reserved for the system application in all the possible usecases (U_t) of the SoC. Here $U_t = \{u_0, u_1, \ldots, u_n\}$, constitutes jointly exhaustive and mutually exclusive subsets whose union completes SoC usecases. The system application connections are, then, allocated in all the usecases such that the reserved resources produce no contention. For this purpose, PUMA makes use of the approach as suggested in [27,28].

To allocate a connection in between the SM and a TCFR, the QoS constraints must be fulfilled across the required path. It is important that the QoS constraints are specified in terms of throughput demand that can be converted into discrete number of time-slots. In other words, a connection from the SM to a particular TCFR is said to be allocated, if it finds the required number of time-slots across the path. However, the time-slots across the path-links should be assigned in a pipelined fashion to meet the required throughput demands as suggested by [28]. For instance, if a path consists of two path-links L0 and L1, where each path-link has a slot table T of size N, then the time-slots across both the links should be positioned such that:

$$IF\ slot_n \in T_{L0}\ THEN\ (slot_{n+1}\ \%|N|) \in T_{L1} \quad (1)$$

The above equation can be implemented by constructing an aggregated slot table, which represents the complete path slot table. We explain the concept by making use of an example.

Example 1. We refer to Fig. 6, which shows 3 nodes of our FPGA architecture, as explained earlier in Section 4. Fig. 6A shows the slot tables associated with each link, where each slot table comprises 5 time-slots. The dark coloured slots represent the allocated slots to the user applications (not shown here for the simplicity). The SM, if communicates with an IP on TCFR3, and the SM uses the path as shown in Fig. 6B. Then, the *empty slots across the path* can be obtained by aggregating the slot tables of all the path-links. For this purpose, we used the approach of [28] that is based on merge and shift process as shown in Fig. 6C. As a starting point, the inputs of an empty slot table and the required path are provided, as shown in Fig. 6C. Afterwards, the slot tables of the links are accordingly shifted and merged as shown with Steps 1–8 in Fig. 6C.

Initially, the zeroth path-link T_{NI0R0} is shifted left by 0 (i.e., not shifted) as shown in Step 1 of Fig. 6C. Afterwards, the merge operation is performed by merging the output of Step 1 to the T_{agg} which is empty at the time of input (Step 2). In our situation, merging of two empty slots tables, i.e., T_{agg} and T_{NI0R0} results in an empty slot table, as shown in Fig. 6C. Then, the slot table of the first path-link T_{R0R1} is selected, which as shown in Fig. 6A, has two occupied slots (i.e., slot 0 and slot 4). Step 3 in Fig. 6C shows that the shift left of T_{R0R1} by 1 slot, moves *slot 4 to slot 3* and *slot 0 to slot 4*. In Step 4, the slot tables of T_{agg} (obtained from Step 2) and T_{NI0R0} (obtained from Step 3) are merged. In Step 5, the second path-link T_{R1R3} is shifted left by 2 slots. In Step 6, the results of Steps 4 and 5 are merged. In Step 7 the third path link T_{R3NI3}, which is the last path-link, is shifted left by 3 slots. In Step 8, the aggregated table is obtained by merging the results of Steps 6 and 7. As a result of the shift and merge process, the final OUTPUT table contains the empty slots that if are allocated to the zeroth path-link (i.e., T_{NI0R0}), then the Eq. (1) is satisfied.

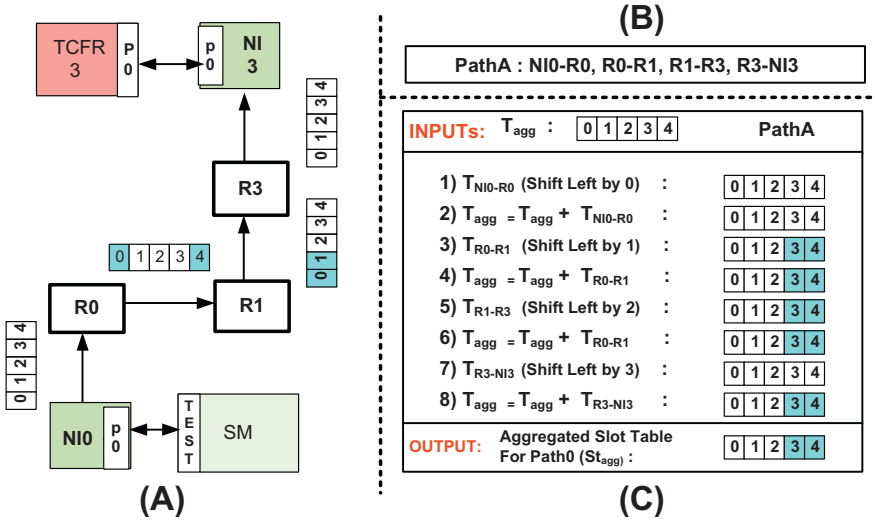

Fig. 6. Slot allocation across the path-links.

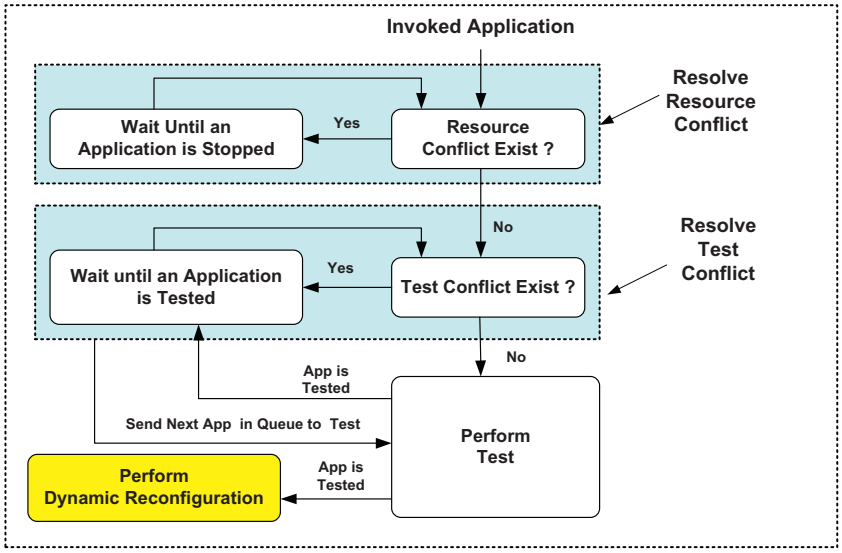

Fig. 7. Run time decision factors before to perform a test.

5.3. Run time phase

The run time flow of our test methodology is three-phased as shown in Fig. 7, and is explained as below.

Algorithm 1. Evaluate Resource Conflict

Require: $ExistingAps, InvokedAp, AllUCs, AllTCFRs$
Ensure: $Resource\ Conflict\ status\ for\ InvokedAp$
1: $ApswithUCs = getUCs(ExistingAps, AllUCs)$;
2: $ApswithTCFRs = getTCFRs(ExistingAps, AllTCFRs)$;
3: $CFconflict = getTCFRConflict(InvokedAp, ApswithTCFRs)$;
4: $ResourceConflict = true$
5: **if** $!CFconflict$ **then**
6: $UCconflict = getUConflict(InvokedAp, ApswithUCs)$;
7: **if** $!UCconflict$ **then**
8: $ResourceConflict = false$;
9: **end if**
10: **end if**
11: **return** $ResourceConflict$

5.3.1. Invoke application

An application that is invoked at run-time can be user-controlled or application-controlled. In case of user-controlled situation, the user decides to trigger a new application or change the parameters of the existing application. In case of application-controller situation, a currently executing application can trigger another application. This is possible when both the applications are the time-multiplexed parts of a larger application. However, when an application is invoked a number of conflicts need to be addressed as shown in Fig. 7.

First, when an application is invoked, it can produce resource conflict with the existing applications. The resource conflict can be in the logic plane (i.e., TCFRs) or in the communication plane (i.e., Hardwired NoC). The resources of HWNoC are allocated, at compile time, such that no contention is produced among the applications that belong to the same usecase. However, the invoked application can have a different usecase than the executing applications. A usecase transition can cause a resource conflict between the invoked and executing applications. In this situation, *the challenge is to resolve the conflicts* such that the testing of the invoked application is non-intrusiveness to other running applications.

Second, an ongoing test process of an application might fully occupy bitstream loading resources. This can restrict another application(s) to configure, though the other application(s) are already tested. In other words, the configuration process of the invoked application with pre-tested region is delayed until the test process is finished. In this situation, *the challenge is to achieve the configuration operation in parallel with the testing process for two different applications*. We discuss these issues in the next two Sections.

5.3.2. Resolve conflicts

After an application is invoked, it goes through the conflict resolution procedure before being tested. The procedure consists of resolving (if any) the resource and test conflicts between the invoked and existing applications. We explain these in the following discussion individually.

Algorithm 2. Evaluate TCFR Conflict

```
CFRconflict = false;
CFRMatch = false;
InvApCFRs = getApCFRs(InvAp, AllCFRs);
a = ExistingAps.begin();
while (!CFRconflict AND a != ExistingAps.end()) do
    Application* ExistingAp = *a;
    ExistingApCFRs = getApCFRs(ExistingAp, AllCFRs);
    ip = InvApCFRs.begin();
    while (!CFRMatch AND ip != InvApCFRs.end()) do
        InvCFR = *ip;
        CFRMatch = CheckCFRPresence(ExistingApCFRs, InvCFR);
        if (CFRMatch) then
            CFRconflict = false;
        end
        ++ip;
    end
    ++a;
end
return CFRConflict;
```

Resolve resource conflict: The Algorithm 1 explains our approach to resolve the possible resource conflict between the invoked and existing applications. Initially, the TCFRs and usecases are obtained for the existing applications, lines (1 and 2) of Algorithm 1. Afterwards, the TCFR conflict and usecase conflict are evaluated

Please cite this article in press as: M.A. Wahlah, K. Goossens, TeMNOT: A test methodology for the non-intrusive online testing of FPGA with hardwired network on chip, Microprocess. Microsyst. (2012), http://dx.doi.org/10.1016/j.micpro.2012.05.011

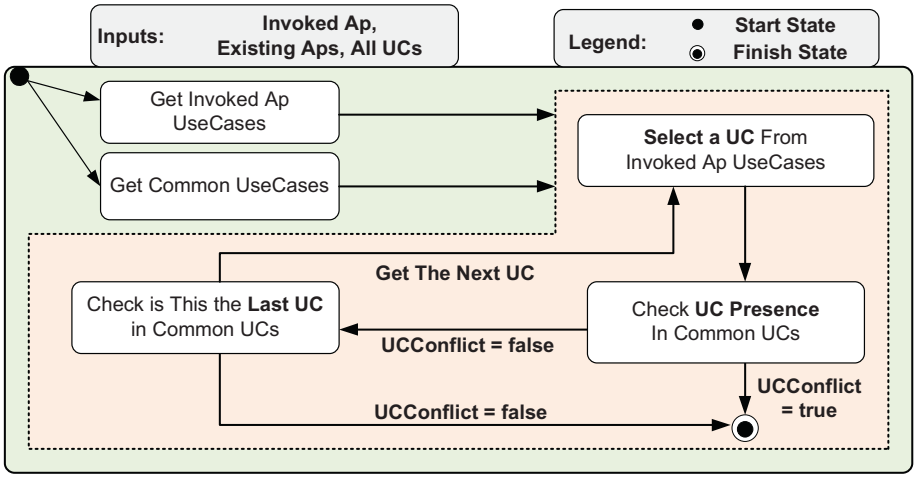

Fig. 8. Evaluate UseCase conflict.

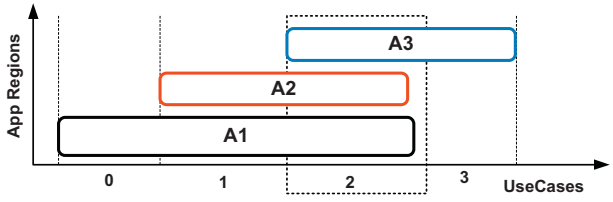

Fig. 9. Extract common UseCase example.

between the invoked application and existing applications, lines (3–10) of Algorithm 1. It is important that the conflict at usecase level is evaluated, only and only if, the TCFR conflict is resolved. In case of any resource conflict (TCFR or usecase), the calling function waits until an application is stopped, as shown in Fig. 7. Afterwards, the possible resource conflict is evaluated against an updated set of existing applications. Once, the resource conflict issue is resolved, the process moves onto the resolve *the test conflict* as shown in Fig. 7. However, we first explain the procedure to evaluate TCFR and usecase conflicts.

Algorithm 2 explains our approach to resolve a possible *TCFR conflict* between the invoked application and existing applications. Initially, the TCFRs of the invoked application are obtained. The inner while loop runs for each existing application, and compares its TCFRs against the TCFRs of the invoked application. In case of a TCFR match, i.e., the invoked application TCFR is found in use by an existing application, a TCFR conflict is reported and the process is terminated. However, in case of finding no TCFR match, then the outer while selects the next existing application. The outer while loop runs until a TCFR conflict is announced, or all the existing applications are examined for a possible TCFR conflict. The algorithm returns a TCFR conflict value to the calling function.

The flow diagram as shown in Fig. 8 explains the process to evaluate a possible *usecase conflict* between the invoked application and existing applications. It starts with extracting usecases for the invoked application. The Algorithm 3 explains our approach to extract the common usecase set. It shows that the outer while loop runs through all the usecases, one by one, whereas the inner while loop determines presence of the current usecase in all the existing applications. For the existing applications, the set of common usecases is extracted. For the convenience of the reader, we present an example system, where three applications A1, A2, and A3 can execute in different usecases, as shown in Fig. 9. When all the three applications are executing, then the *usecase 2* is the common usecase, as highlighted with rectangle in Fig. 9.

Algorithm 3. Extract Common UseCases

```
CommonUCs;
UCMatchInAllApps = true;
u != allUCs.begin();
while u != allUCs.end() do
    curUC = *u;
    a = ExistingAps.begin();
    while (!UCMatch AND a != ExistingAps.end()) do
        Application* ExistingAp = *a;
        ExistingApUCs = getApUCs(ExistingAp, AllUCs);
        UCMatch = CheckUCPresence(ExistingApUCs, curUC);
        if !UCMatch then
            UCMatchInAllApps = false;
        end
        ++a;
    end
    if UCMatchInAllApps then
        addUC(CommonUCs, curUC);
    end
    ++u;
end
return CommonUCs;
```

After obtaining the common usecases in existing applications, the usecases of the invoked applications are evaluated for presence against the set of common usecases, as shown with dotted box in Fig. 9. On finding a match between the invoked application usecase and the common usecase set, the flow terminates. This shows that the invoked application can execute in parallel with the existing applications without producing a resource conflict in the hardwired NoC.

Resolve est conflict: The test process is a time-consuming process. It is, therefore, possible that while online testing of an application, multiple applications have been invoked and become candidate for testing. In this situation, once the ongoing test finishes, multiple application can contend for the test process. This in turn can introduce conflict for testing. However, we resolve the issue by putting the applications in a queue. The test process for the application is started, only and only if, the test has been performed for (if any) earlier invoked applications, as shown in Fig. 7.

Please cite this article in press as: M.A. Wahlah, K. Goossens, TeMNOT: A test methodology for the non-intrusive online testing of FPGA with hardwired network on chip, Microprocess. Microsyst. (2012), http://dx.doi.org/10.1016/j.micpro.2012.05.011

5.3.3. Perform TCFR test

At run time, the online testing of an application TCFRs is performed. However, before explaining the process we discuss the nature of faults that our methodology targets.

Stuck-at fault description: A test configuration functional region, as detailed earlier in Section 4.2, consists of multiple CLB units (in our case it is 512 CLBs). The online test methodology targets stuck-at faults that can occur in the combinational part a CLB (i.e., LUTs), and configuration memory of a CLB. However, the configuration memory of a CLB that is going to be tested for stuck-at faults stands for the: (i) combinational part of a CLB, (ii) sequential part of CLB, and (iii) associated interconnection network.[5]

In a faulty FPGA, the configuration memory cell can stuck-at a particular value, i.e., 0 or 1. To find out stuck-at faults in the configuration memory, we make use of two complementary test bitstreams. One test bitstream, configures each LUT as exclusive-OR (XOR) gate. The other test bitstream configures the LUTs as exclusive-NOR (XNOR) gate.

Similarly, the functional input(s) of an LUT can get short. The output of LUT can then permanently point (i.e., stuck-at) to the same configuration memory address irrespective of the input patterns applied to an LUT. To figure out a defective input port to configuration memory address mapping, we provide an exhaustive set of $2^4 = 16$ test vectors of 4-bit each at the functional inputs of an LUT. This way we can verify that the output of LUT is as per desired or not, i.e., the output is not stuck-at a particular configuration memory address.

A 5 phase (programme, write, execute functional cycle, read, and evaluate) test process is carried out for each test bitstream, (i.e., XOR and XNOR configurations), see Fig. 10. The test process starts with programming a test connection for the required TCFR. This is followed by writing the test bitstream to that TCFR. Once writing is finished, a functional cycle is executed to find out prospective defects in the functional ports of CLBs. Afterwards, a read-trigger is sent to the TCFR to read-back the test bitstream. To find out the faults in the configuration memory, the read-back bitstream is then evaluated against the originally written test bitstream. In the following discussion, we explain each of the phases.

Program: Fig. 11 illustrates the steps involved in establishing a connection from the system manager to a TCFR. (1) It starts with programming the required tables (path, slot, and channel tables) of the system manager NI, so as to reach and programme a TCFR NI. (2) After reaching a TCFR NI the system manager programmes the TCFR NI tables by sending path, slot, and flow control information. The programming of TCFR NI ensures a contention free path in between a TCFR and the system manager. (3) A response channel, which is used for test bitstream read-back, is then opened from a TCFR to the system manager. (4) This is followed by opening request and response channels, which are used to write and read-back test bitstreams in between the SM and TCFR.

Write: The system manager after programming a connection retrieves the test bitstream from an off-chip memory. The bitstream is a combination of packet headers and frame data to configure a complete TCFR. The test bitstream is sent in the frame-wise manner over an established test connection. The process is repeated until the last set of the test frames is sent to the destination TCFR.[6] The destination TCFR, after finding the bitstream header, extracts information about the start frame address and the total number of incoming bitstream frames. The start frame address is input to the address decoder (in Fig. 4) to activate the respective MCR, Algorithm 4 (lines 5–9). Onwards, the controller forwards the test bitstream frames in the input frame register (IFR), Algorithm 4 (line 11).

In our design, the frame length is 41 words and hence the size of the IFR. Additionally, the data-bus is one word wide to save the area, and is used for word-by-word streaming of the test data to the respective MCRs. Hence, each time the test data is pushed into the respective MCR, the earlier positioned words in a MCR are shifted to the next locations serially. This forms a scan chain, which spans the whole MCR, Algorithm 4 (lines 15–26). Meanwhile, the IFR data is shifted rightwards by an amount of the sent words, Algorithm 4 (lines 30–36). The process repeats until the full frame is loaded into the respective MCR, Algorithm 4 (lines 12–43). The online test process loops over the required number of bitstream frames to load the test IP over the respective TCFR.

Algorithm 4. Writing the Test Bitstream at a TCFR, frame = single

```
1  // INPUT: bitstream
2   TCFRMCRs = 32, MCRFrms = 23, FrmBytes = 164, WrdByt
3  IFRsize=41, FrmWrds = FrmBytes / WrdBytes.
4  //Each Time On Receiving the Test Bitstream
5  IF (Header)
6  |     InHeader = getFrame(bitstream)
7  |     FrmCnt = getFrameCount(InHeader)
8  |     StFrmAddr = getStartFrameAddr(InHeader)
9  |     CurMCR = setAddressDecoder(StFrmAddr, MCRFrms)
10 ELSE
11     IFR = getFrame(bitstream)   //IFR = In Frame Reg
12 For(i =0; i < FrmWrds; i++)    // Loop over Frame Wo
13 |// In the following loop, Shift-right the MCR-words
14 |// bytes) before pushing a word from the DataBUS to
15 |// for instance if MCR=1, j=1, then TCFR bytes Indea
16 |  IF(LoadWrd > 0)
17 |  |   For(j = LoadWrdNo; j>0; j--)
18 |  |   Indx = CurMCR x MCRFrms x FrmBytes + j x WrdBy
19 |  |   TCFR[Indx-1 + WrdBytes] = TCFR[Indx-1] //Shift-
20 |  |   TCFR[Indx-2 + WrdBytes] = TCFR[Indx-2]
21 |  |   TCFR[Indx-3 + WrdBytes] = TCFR[Indx-3]
22 |  |   TCFR[Indx-4 + WrdBytes] = TCFR[Indx-4]
23 |  | ENDFor
24 |  ENDIF
25 |
26 | DATABUS(InFrmRg,RegAddr,CurMCR,TCFR) //Push Word-to
27 | LoadWrd++
28 |
29 |// Shift-Right i.e. 40th -> 41st Register in IFR etc
30 |// Shifts Required: IFRsize - (Loaded Words In MCR)
31 |  IF(LoadWrdNo < FrmWrds)
32 |  | For (i=0; i < (IFRsize-LoadWrdNo); i++)
33 |  | Indx = (IFRsize x WrdBytes) - (i x WrdBytes)
34 |  | IFR[Indx-1] = IFR[Indx- 1 - WrdBytes] //Shift-B
35 |  | IFR[Indx-2] = IFR[Indx- 2 - WrdBytes]
36 |  | IFR[Indx-3] = IFR[Indx- 3 - WrdBytes]
37 |  | IFR[Indx-4] = IFR[Indx- 4 - WrdBytes]
38 |  ELSE  // When All Frame Words Loaded:
39 |  | LoadWrdNo = 0;
40 |  | CurFrmNo++;
41 |  | LoadFrmNo++;
42 |  | CurMCR =  setAddressDecoder(LoadFrmNo, MCRFrms)
43 |  ENDIF
44 ENDFOR
45 ENDIF
```

Execute functional cycle: At the end of bitstream writing process, a test IP is placed on the logic plane of a TCFR. The logic circuit of test IP spans complete TCFR, i.e., the bitstream of test IP is loaded in all the minimum configuration regions (MCRs) of a TCFR, as shown in Fig. 12A. It is important that each minimum configuration region (MCR) unit in a TCFR is configured as a set of four chains of CLBs, as shown in Fig. 12B. Moreover, each of the eight (4-bit input to 1-bit

[5] Each of 8 LUTs in a CLB has 16 1-bit configuration memory cells, but the configuration memory that is associated wit a complete CLB is approximately 1800 bits [24,29].

[6] We refer to Fig. 4 while explaining the write process.

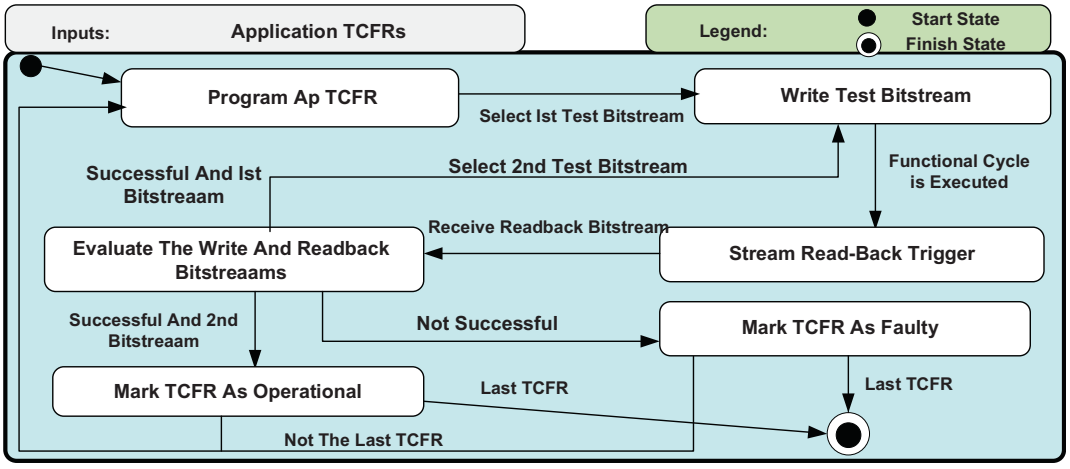

Fig. 10. Run time flow for the test process.

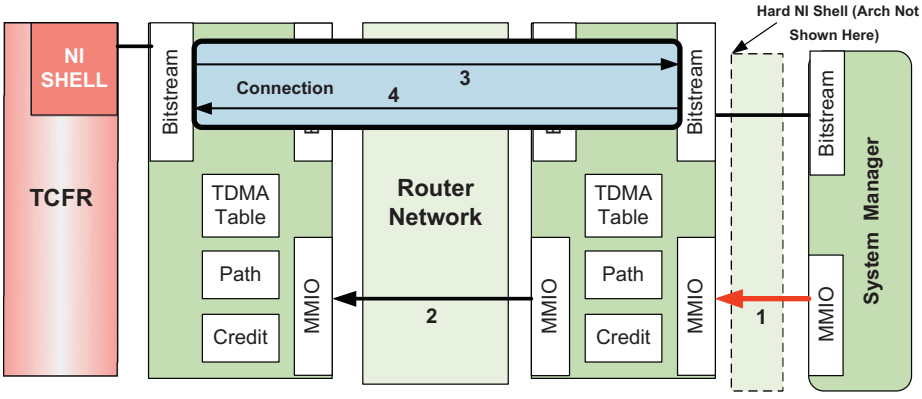

Fig. 11. Setup of the test connection from the system manager to a TCFR.

output) look-up-tables in a CLB is configured as an independent exclusive-OR or exclusive-NOR circuit, as shown in Fig. 12D and E respectively. Hence 32 bits are required to derive the inputs of all the LUTs in a CLB, which in turn produces an 8-bit output, see Fig. 12B.

Fig. 12B shows that for each CLB chain, the 32-bit inputs of the first CLB are derived from a 32-bit register. This can be a hardwired (Block RAM) register, which is programmed through the system manager. The 8-bit output of the first CLB is then input to the next cascaded CLB and so on, see Fig. 12B. As a result of the arrangement, each CLB chains has an 8-bit output that is fed into a 32-bit programmable register. The data is then sent back to the system manager that to analyse the response of an MCR for a particular set of input data. At this point it is important to mention that the use of hardwired registers for input and output of CLB chains is a design time choice. The hardwired registers are used, because it eliminates the need to configure additional hardware on the reconfigurable plane that is already under test.

Once the test IP is placed, the system manager executes the functional cycle to find out the prospective defects in the mapping of functional ports of an LUT. The system manager generates and transports test vectors for all the CLB chains by using the HWNOC. To test each CLB chains, the system manager sends 16 test vectors of 32-bit each. Each test vector is then further decomposed into 8 groups of 4-bit each that changes from 0000 to 1111 during. The reason is that each of the eight (4-bit input to 1-bit output) look-up-tables in a CLB is configured as an independent exclusive-OR or exclusive-NOR circuit. It is important that the whole process is pipelined and it takes approximately 4 cycles or to reach from the input to output of a CLB chain.

We use an example to explain the process of test generation and evaluation of response by the system manager. (1) The system manager sends a test vector of 1101 1101 1101 1101 1101 1101 1101 1101 to test a CLB chain. The test vector is stored in the 32-bit register that is connected to the first block of a CLB chain, see Fig. 13. All the LUTs of CLB chain are configured as XOR, whose expected response for a particular input is mentioned in Fig. 12D. (2) The fault-free LUTs of CLB 1 after receiving the input are then expected to produce an 8-bit high output, i.e., 1111 1111, as shown in Fig. 13. (3) The 8-bit output of first CLB then servers as input for 2 LUTs of the cascaded CLB 2. In addition, the 24-bits from the programmable register are input to the remaining 6 LUTs of CLB 2, see CLB2 inputs in Fig. 13. This means, CLB 2 receives an input test vector of 1101 1101 1101 1101 1101 1101 1111 1111. The output of fault-free CLB 2 is then expected to be 1111 1100. (4) Similarly, CLB 3 receives the input from CLB 2 and programmable register both, see Fig. 13. The fault-free CLB 3 and CLB 4 both should produce output of 1111 1100, as shown in Fig. 13. (5) The system manager then receives the output and evaluates the correctness of functional inputs of CLBs for a particular test vector.

After evaluating defects in the combinational part of a TCFR, the system manager triggers the read-back bitstream process. This is performed to evaluate defects in the configuration memory of a TCFR.

Read-back: The system manager, after writing the test bitstream, sends a read-triggered bitstream. It consists of the address

Please cite this article in press as: M.A. Wahlah, K. Goossens, TeMNOT: A test methodology for the non-intrusive online testing of FPGA with hardwired network on chip, Microprocess. Microsyst. (2012), http://dx.doi.org/10.1016/j.micpro.2012.05.011

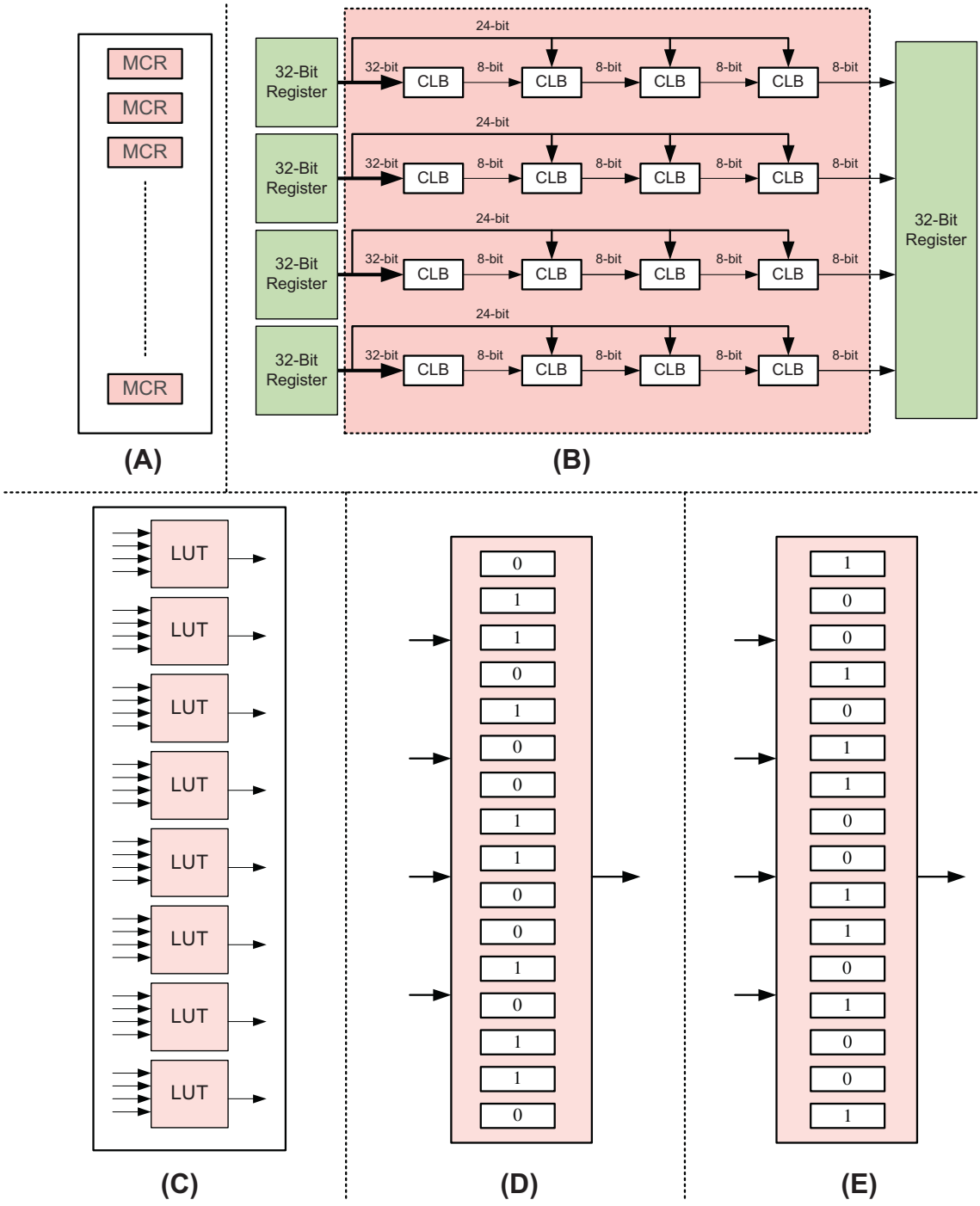

Fig. 12. Showing: (A) test IP circuit in terms of MCRs, (B) an MCR configured as a set of multiple CLB chains, (C) 8 LUTs in a CLB that are not cascaded within a CLB, (D and E) An LUT configured as XOR and XNOR gate respectively.

of first TCFR frame, and total number of frames to read. In response to this read-triggered bitstream, a TCFR starts reading-back the bitstream frames to the system manager. The process initiates by retrieving the frame, being currently pointed out by the address decoder, in the out frame register (OFR). The remaining process inside a TCFR is the exact reversal of the bitstream writing. However, the frame on its way back to the system manager is first serialised into *32* bit words by the hardwired NI shell of a TCFR. It is then sent over the response channel of the already established test connection. The whole process continues until all the TCFR frames are read-back to the system manager.

Evaluate: In this phase the system manager evaluates the read-back bitstream by comparing it with the originally written bitstream. The evaluation process is performed in parallel with the read-back bitstream and on frame-wise basis. As stated before, two complementary bitstreams are required to identify possible stuck-at 0 and stuck-at 1 faults in a TCFR. However, the second bitstream is sent only and only if the evaluation process is successful for the first test bitstream, as shown in Fig. 10. The reason is that our paper currently focuses on fault detection, and no fault tolerance mechanism is applied to replace the faulty portion of a TCFR. This allows us to save the time that can be spent in testing a TCFR,

Please cite this article in press as: M.A. Wahlah, K. Goossens, TeMNOT: A test methodology for the non-intrusive online testing of FPGA with hardwired network on chip, Microprocess. Microsyst. (2012), http://dx.doi.org/10.1016/j.micpro.2012.05.011

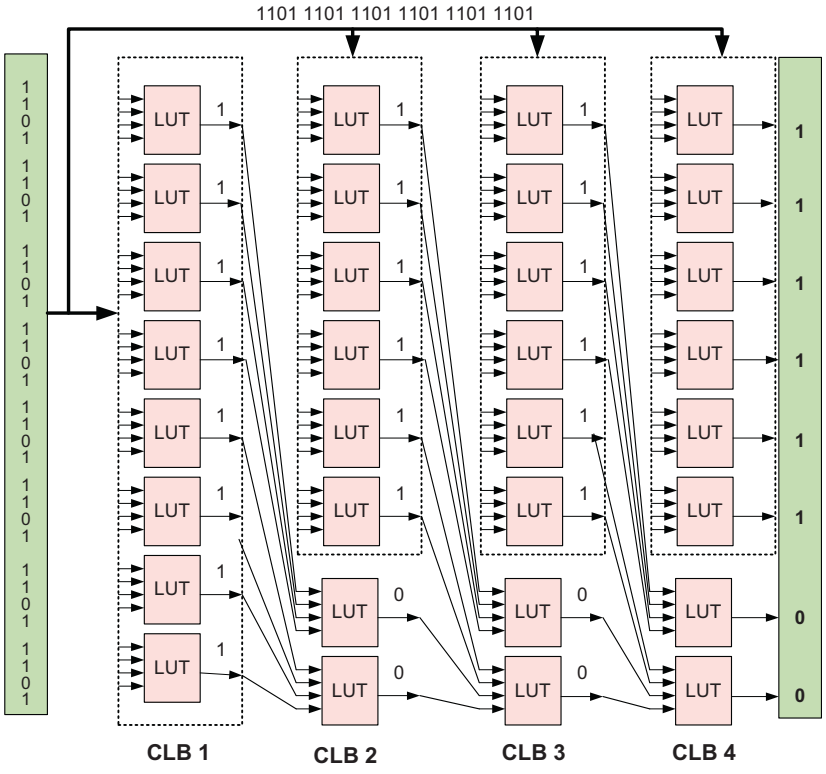

Fig. 13. Test IP test generation example.

which has already been detected faulty. Currently, we replace a faulty TCFR with the one that has already passed the structural test.

5.3.4. Perform HWNoC test

The HWNOC testing is associated with testing of network interfaces and routers. The network interfaces can be functionally tested by using the IP cores attached with them as suggested in [30]. The method proposed in [30] tests the functionality of NIs, because a fault-free NI can then produce the expected functionality for the attached core.

The routers can be tested online by using the approaches of [31] and [25] [http://ce.et.tudelft.nl/publications.php?author=1046]. Special wrappers are used in [25], which can switch a router in test or functional mode. The system manger first excludes the router under test from the list of available routers. Initially sequential part, i.e., flip-flops of router is tested. The system manager that acts as test pattern generator (TPG) and test response analyser (TRA) both, initially sends test vectors to the scan-in input of router under test and receives the output from the scan-out pin. On the contrary, the testing of combinational part is performed in two step. First the router is set to initial state by resetting its sequential block. Then the test vectors are provided to its functional inputs. The functional output port of router generates an output that is received again the system manager and is analysed. Based on sequential and combinational outputs, the system manager decides if the router is defected or not.

It is important that when a router is detected faulty, and it is part of the path that a packet should go through, then the packet should be rerouted. For this purpose, the router is taken out of the topology. The system manager onwards can update the path from a specific source to destination, after ensuring that the faulty router is not part of the path. This in turn requires online allocation of resources for the updated path, which can be achieved by using the approach of [32]. This way we can ensure that HWNOC is not proved to be bottleneck due to faulty routers while online testing is performed.

Once the test is finished, the dynamic configuration of the application is started[7] Additionally, a notification about the termination of test process is sent to the calling function. The calling function, depending upon the status of the test queue, can forward the next application to the *Perform Test* unit, as shown in Fig. 7. In this situation, both the configuration and test process will be performed in parallel with each other. It is implemented by using independent ports of the system manager, and allocating the contention free resources for connections of each port as explained earlier in Section 5.2. Additionally, to allow interleaved test and configure operations, we impose a restriction that no two applications share the same TCFR(s) simultaneously.

6. Results and analysis

We exercise our online test methodology in SystemC, and use ÆTHEREAL [35,36] NOC as the hardwired NOC. The HWNOC platform runs at 500 MHz, and consists of routers and NI kernels with FIFO sizes of 3 and 41 words, respectively. We use Virtex-4 XC4VLX200 chip to synthesis the applications, and then embed the synthesis results in the SystemC model of our FPGA. The FPGA architecture consists of multiple TCFRs, whose combined area is equivalent to 178,176 LUTs, i.e., equals to *Virtex-4 XC4VLX200* chip. We use two complementary test bitstreams to *verify a TCFR for stuck-at fault model*. The size of the each test bitstream is estimated from the following equation:

$$(IP_{LUT} * MCR_{frame})/(CLB_{LUT} * MCR_{CLB}) \quad (2)$$

[7] The details of dynamic reconfiguration are out of scope, and can be found in [33,34].

In Eq. (2) the first term, i.e., IP_{LUT} refers to LUT area of an IP, CLB_{LUT} refers to LUTs in a single CLB, and MCR_{frame} and MCR_{CLB} stand for frames and CLBs in a single MCR respectively. Each CLB consists of 8 LUTs, and an MCR consists of 16 CLB units.[8] This means, our FPGA architecture consists of (178,176/8) = 22,272 CLBs and 1392 MCRs. Moreover, an *MCR is configured by using 23 41-words frames* [29,24].

For our online test methodology, we (i) provide an evidence of its non-intrusive nature (Section 6.1), (ii) evaluate the performance in terms of fault detection latency (Section 6.2), (iii) evaluate the cost in terms of spatiotemporal overheads (Section 6.3), (iv) analyse the impact of TCFR area on the performance and cost of our methodology (Section 6.4), and (v) compare the performance and cost with two state-of-the-art schemes [15,18] (Section 6.5).

6.1. A non-intrusive test methodology

In this section we analyse the non-intrusive nature of our test methodology. First, we describe the experimental setup in the following discussion.

6.1.1. Experimental setup discussion

We use the behavioural models of H.264 IPs, i.e., (Residue, DCT, and Quantiser). Synthesis of the VHDL implementations of these IPs, on a Virtex-4 XC4VLX200 chip using Xilinx ISE 10.1, provide their MHz frequencies that are used in the SystemC simulations, Table 2. The size of their bitstreams is estimated from their *k*LUT areas by using the Eq. (2).

To verify the non-intrusive nature of our online test methodology, we use a *system with three applications*, i.e., A0 (Residue + DCT), A1 (DCT only), and A2 (Quantiser only) that execute in two usecases. A0 and A1 execute in parallel and belong to usecase 0 (UC0), whereas A1 and A2 execute in parallel and belong to usecase 1 (UC1). Importantly, A0 and A2 are the sub-applications of one larger application, and are time-multiplexed over the same set of TCFRs, i.e., TCFR0 and TCFR1. On the other hand A2 can execute on TCFR3, as shown in Fig. 14.

6.1.2. Non-intrusive behaviour analysis

To evaluate the non-intrusive nature of our online test methodology, we *analyse the system in different modes*. This means, while testing the system goes through different operations, i.e., execution, configuration, programming, and usecase transitions. For this purpose, we (a) *start A1 test* while A0 is executing, and (b) *start A1 loading* (at the end of A0) while A1 is testing. It is important that we do not conduct test before starting the loading of A1. By doing so, we evaluate (i) the impact of (if any) dynamic configuration (A2) on the online testing (A1), and (ii) the ability to test and configure at the same time, as claimed in Section 3. Fig. 15A shows the timing details of test, load, and execution operations of the system applications in different usecases.

We analyse the behaviour of our methodology for above two scenarios, and *take a small portion of 300 μs* as illustrated with Fig. 15B. The time interval shows *the presence of all the three operations* of different applications, i.e., execute (A0), test (A1), and load (A2). Important observations can be drawn from the above scenarios. For instance, as shown in Fig. 15A, the *loading of A2 triggers the usecase transition*, which should be transparent to the ongoing test process of A1. In Fig. 15B, the first rectangle highlights the unaffected behaviour of A1 test, while the usecase transition is made due to t he A2 invocation. Importantly, as shown in Fig. 15B, our methodology supports *the interleaved test and load operations* of multiple applications. It is important that *A1 test and A0 execution both share a network path (R0-R1)*, as shown in Fig. 14. The shared path can lead to intrusiveness in between the test and execution operations. However, the compile time phase of our test methodology ensures that the test and execution data for multiple applications, on the shared R0-R1 path-link, do not produce any contention as shown in Fig. 16. It is made possible by making sure that the test data is forwarded, on the shared R0-R1 path-link, *by allocating fixed and non-contending time-slots*.

Table 2
IP Synthesized area, frequency and bitstream frames.

IP	Area (kLUTs)	Frequency (MHz)	Bitstream (Frames)
Residue	1.68	100	285
DCT	2.36	66	396
Quantizer	2.21	75	370

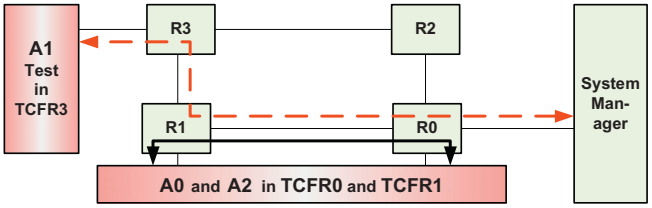

Fig. 14. Applications in different TCFRs.

6.2. Performance: fault detection latency

We determine the fault detection latency of the whole FPGA, which contains 1392 MCRs (i.e., equivalent to Virtex-4 XC4VLX200). For experiment purpose, we divide the FPGA into 4 TCFRs of 348 MCRs each. We determine the fault detection latency of a TCFR and then scale the calculations to the whole FPGA chip.

Fig. 17 illustrates the latency to detect a fault in a TCFR that could belong to any application. It is 3-phase procedure, which is explained earlier in Section 5.3.3, and initiates with programming the test connection in 21 μs. Afterwards, the writing of the test bitstream, which consists of 8004 frames (348 MCR × 23 frames/MCR), starts. We provide the detailed analysis of the *writing of the test bitstream* in Fig. 18, and is explained as below.

To test a TCFR, we reserve approx. 10% of link resources of the HWNoC architecture.[9] The slot table of each link consists of 80 slots, which means 8 slots are reserved for the test connection. It is important that each test connection consists of a request channel to write the bitstream, and a response channel to read-back the bitstream. The request and response channels are allocated equally, i.e., *each channel with 4 number of slots*, where each slot can transport a single flit (i.e., 3 words). This means, in a single slot table iteration a maximum of 12 words can be written *from the SM to the attached router*, as shown in Fig. 18A. The 12 words contain a 1 word header with an embedded path to the destination TCFR, and a payload of 11 words. Hence to *write a test bitstream frame* (41 words), from the system manager to the attached router, *the slot table iterates for 4 times* (Fig. 18A). It is important that the time in between the two iteration of the slot table, which consists of 80 slots and iterates at a frequency of 500 MHz, equals to 80 × 0.002 = 0.16 μs.

Once the data is on the router network, the flits are transported across the connection links in a pipelined fashion. This is possible due to the assurance of resources at compile time. A router in our

[8] In Virtex-4 the minimum configuration region, i.e., an MCR consists of a column of 16 CLBs and the associated programmable interconnect [24].

[9] Reserving more than 10% of resources for test purpose can significantly affect the success rate of applications to FPGA binding. For details see [26].

Please cite this article in press as: M.A. Wahlah, K. Goossens, TeMNOT: A test methodology for the non-intrusive online testing of FPGA with hardwired network on chip, Microprocess. Microsyst. (2012), http://dx.doi.org/10.1016/j.micpro.2012.05.011

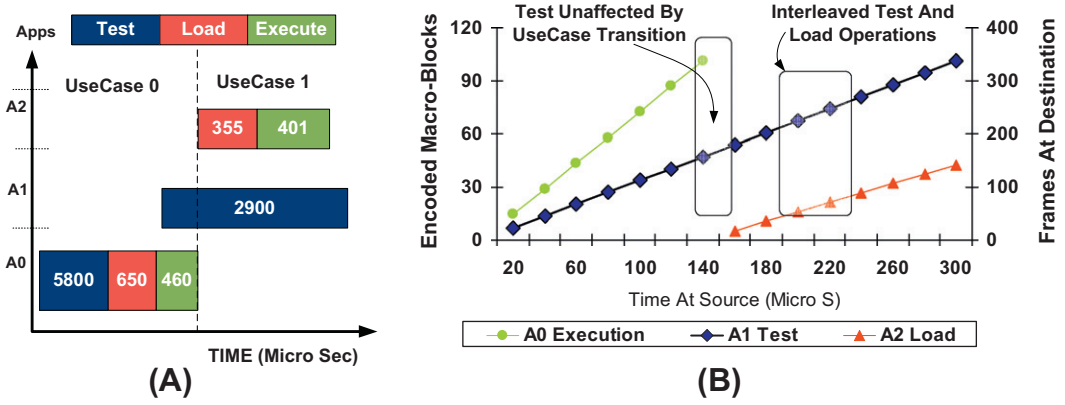

Fig. 15. (A) Application timing details in different phases, (B) interleaved test and load for A1 and A2.

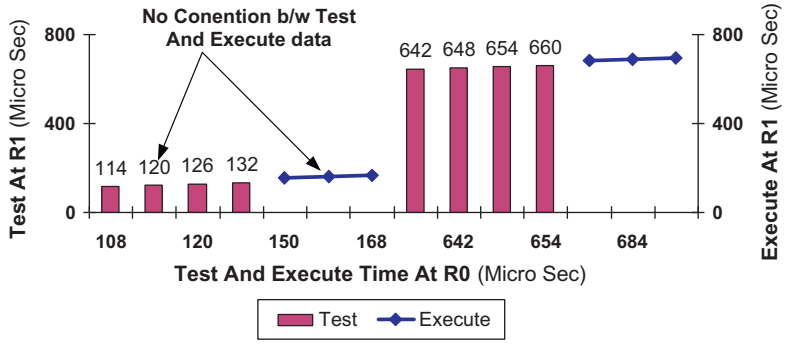

Fig. 16. Non-intrusive test and execute on a shared R0-R1 path-link during application execution.

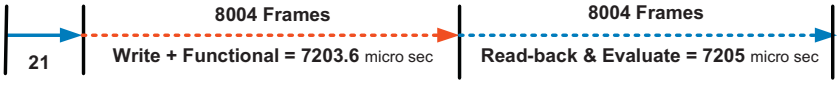

Fig. 17. Timings of different phases to detect a fault in TCFR of 348 MCRs.

HWNOC takes threes cycles to forward a flit from the input to the required output. This means, the router, running at 500 MHz, takes $3 \times 0.002 = 0.006$ μs to forward an input flit to the required output. In our system the first flits takes 0.024 μs to reach a TCFR that is 3 routers away, as shown in Fig. 18B. The flits are pipelined, therefore, the remaining flits come after each other with a time interval of 0.006 μs. This means, a maximum of $0.024 + 0.006 + 0.006 + 0.006 = 0.042$ μs are spent to transport 4 flit data *from the SM router to the destination TCFR*, as shown in Fig. 18C. There is an important observation that the test bitstream flits reach the destination TCFR before the next slot iteration, as shown with rectangle in Fig. 18C.

The TCFR, running at 200 MHz, after receiving each frame follows the procedure of Section 5.3. The TCFR takes approx. 0.22 μs to write a test bitstream frame to the desired MCR location. The writing of a test bitstream frame is completed in approx. $0.16 \times 4 + 0.042 + 0.22 = 0.90$ μs, as detailed in Fig. 18C. For the complete TCFR of 348 MCRs, a total of $348 \times 23 \times 0.90 = 7203.6$ μs are spent to test the TCFR.

Read-back process is the inverse of the writing process. It takes approx. 7205 μs to read-back the complete test bitstream. It is important that the evaluation process is performed on the frame-wise basis, and in parallel with the read-back. The system manager, after receiving a test bitstream frame, evaluates a test frame for the possible stuck-at faults. The system manager runs at 250 MHz, and it takes approx. 0.17 μs to evaluate a single frame. However, the evaluation process does not add to the fault detection latency, because it is performed in parallel with the read-back process. This means, while reading the next frame, the previous frame is evaluated for the possible faults.

In short the total fault detection latency of a single test bitstream, for a TCFR of 348 MCRs, equals to $10.4 + 7203.6 + 7205 = 14{,}419$ μs. With two test bitstreams, the worst case fault detection latency of a TCFR is approx. $2 \times 14{,}419 = 28{,}838$ μs. For the *complete* FPGA *chip, which consists of 4 TCFRs of 348 MCRs each*, the worst case fault detection latency accounts to $4 \times 28{,}838 = 115{,}352$ μs.

Next, we discuss the cost in terms of spatiotemporal overhead to test a TCFR in the current FPGA architecture.

6.3. Cost: spatiotemporal overhead

We use a test connection to write and read-back the test data. Therefore, the resources acquired by the connection are accounted as the spatiotemporal overheads of our methodology. The temporal overhead of the test connection is found to be approx. 21 μs, which is the time required to setup a connection.

The spatial overhead of a test connection, between the source SM and destination TCFR, accounts for a number of hardware resources. (1) Two NI-Shell to serialise and deserialise the test frame in between the SM and the destination TCFR. (2) Two 41 words FIFOs at the NI kernels of the SM and destination TCFR to send

Please cite this article in press as: M.A. Wahlah, K. Goossens, TeMNOT: A test methodology for the non-intrusive online testing of FPGA with hardwired network on chip, Microprocess. Microsyst. (2012), http://dx.doi.org/10.1016/j.micpro.2012.05.011

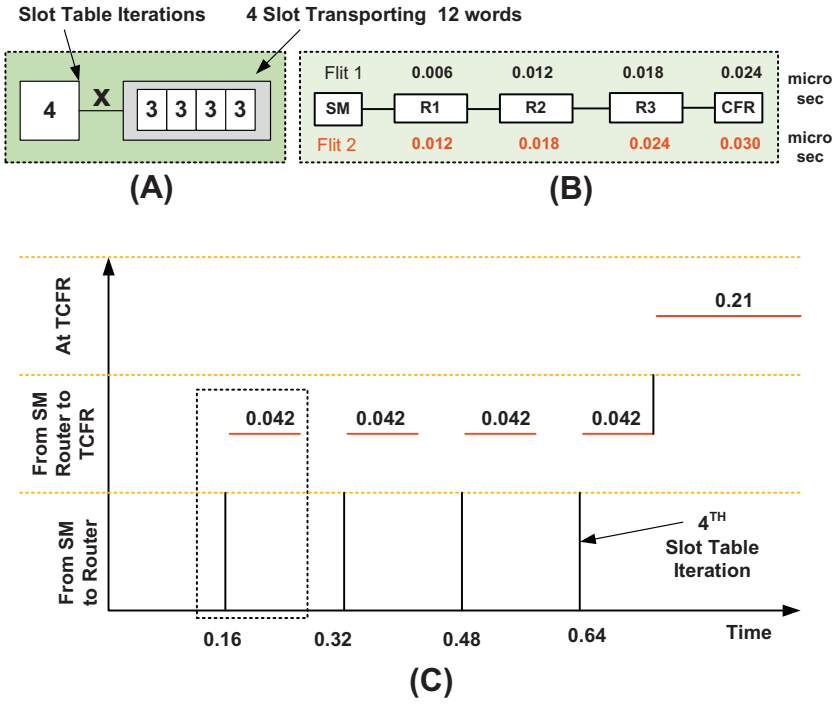

Fig. 18. (A) Slot table iterations to write a test frame of 41 words, (B) time for 2-pipelined flits to reach a TCFR 3-hops away (μs), (C) detailed timings to write a test frame from the system manager to the destination TCFR.

and receive a test frame, respectively. (3) Additionally, the FIFOs (3 words deep) of each router that are used during the connection time also contribute towards the spatial overhead. The connection, at worst-case, can utilize the FIFOs of all the 4 routers of the HWNOC architecture. The *connection resources are hardwired*, and to obtain the (ASIC) area numbers, we refer to [37].

The authors in [37], illustrate that an ASIC implementation can take approx. 35 times lower area than its equivalent FPGA implementation. Therefore, we first synthesis (on Virtex-4 XC4VLX200 chip and by using Xilinx ISE 10.1) the connection resources, then reduce these by (conservative) 30 times to obtain the ASIC equivalents. The synthesis numbers for each component are: (1) 23 CLBs for an NI-Shell, (2) 390 CLBs for a 41-word FIFO, and (3) 29 CLBs for a 3-word FIFO. In addition, the cost of hardwired test resources that are reserved in a TCFR is equivalent to $5 \times 32 = 160$-word FIFO. This in turn requires approximately 1520 CLBs. The total spatial overhead of a synthesised test connection then accounts to $2 \times 23 + 2 \times 390 + 4 \times 29 + 1520 = 2472$ CLBs. After hardwiring the connection resources, *the actual spatial overhead to test a TCFR is approx. 82.4 CLBs.*

6.4. TCFR area impact on performance and cost

In this section, we analyse the impact of TCFR area on the performance and cost of our test methodology. For this purpose, we select four different archtiectures of the FPGA, each of which has a different TCFR area as shown in Fig. 19. For instance T-174 in Fig. 19 represents an FPGA architecture with 8 TCFRs of 174 MCRs each. Fig. 19 also shows that the worst-case fault detection latency decreases from approx. 29 ms to 3.6 ms when TCFR area is varied from 348 MCRs to 44 MCRs. The high performance, in terms of reduced fault-detection latency, induces high costs in terms of spatiotemporal overheads.

The second and third columns of Table 3 illustrate the temporal and spatial overheads of each FPGA architecture, respectively. For instance, the spatial overhead of T-174 FPGA architecture is found to be 138 CLB. In T-174, the test connection can access any of the 8 TCFRs. Therefore, the worst-case spatial overhead is obtained by accounting the area costs of all the 8 NI-Shell, 41 words FIFOs (one in each NI kernel), and 3 words FIFOs in the routers. In addition, the area cost of hardware registers, which are used for input and output of test IP CLB chains, in a TCFR is also included. The area cost of hardware register is found to be 51 CLB equivalent.

For the whole FPGA, decreasing the TCFR area from 348 MCRs to 44 MCRs, the temporal overhead increases from approx. 84 to 672 μs, whereas spatial overhead increases from approx. 127 to 559 CLB equivalent.

6.5. Comparison with the state-of-the-art

We compare the performance and cost of our methodology with the existing state-of-the-arts [15,18]. The comparison is made for the compile and run time phases, and as a *fairness of the comparison*, we use the same platform of Virtex-4 for each of these schemes.

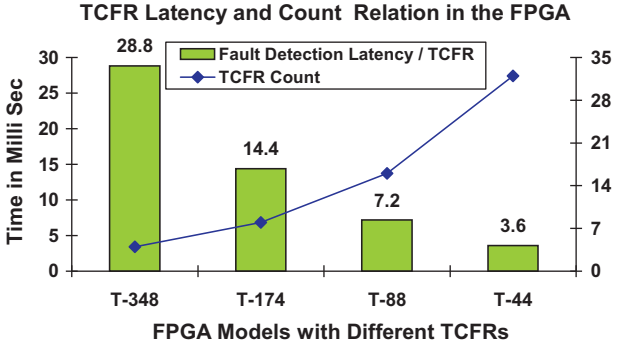

Fig. 19. Different FPGA architectures with variable TCFR area and count. also showing fault detection latency per TCFR.

Please cite this article in press as: M.A. Wahlah, K. Goossens, TeMNOT: A test methodology for the non-intrusive online testing of FPGA with hardwired network on chip, Microprocess. Microsyst. (2012), http://dx.doi.org/10.1016/j.micpro.2012.05.011

Table 3
Cost Evaluated for *the Complete* FPGA after Varying TCFR Area.

FPGA arch.	Temporal overhead (μs)	Spatial overhead (CLB equivalent)				
		NISh	NIK	Routers	TCFR	Overall
T-348	84	3.8	68.3	4	51	127.1
T-174	168	7	123	8	51	189
T-88	336	13	232.3	16	51	312.3
T-44	672	25.3	451	32	51	559.3

It is important that the range of faults that we cover in the paper closely matches with the ones that authors in [15,18] have addressed. For instance, the work of [15] performs the online test at the granularity of a single CLB. Similarly, the authors in [18] test CLBs in group of 4. The nature of faults that are covered in [15,18] is stuck-at faults in the sequential and combinational logic of CLBs. In comparison, we perform the test at the granularity of a TCFR, i.e., 512 CLBs or 32 MCRs. However, we test the combinational part of CLBs (i.e., LUTs), but in addition we perform the test for configuration memory cells as well.

At the same time it is important to mention that our test scheme ensures a non-intrusive nature of online testing with respect to already running applications. In comparison, though the approach of [15] also ensures un-disrupted application execution by using active replication method, but no such claim is found for the other state-of-the-art, i.e., [18].

6.5.1. Run time (fault detection latency)

The authors in [15] first replicate a CLB, which is to be tested, on an earlier tested CLB. The test-access-mechanism runs at 20 MHz to replicate and test a CLB in 24,000 μs. However, *as a fairness of the comparison*, we use the same Virtex-4 features for [15], which runs BSI TAM at 66 MHz. Therefore in [15], the fault detection latency of a single CLB reduces from 24,000 μs to 8000 μs. On a scaled level of TCFR with 348 MCRs (5568 CLBs), the fault detection latency of the scheme [15] is approx. 5568 × 8000 = 44,544,000 μs or 44,544 ms, as shown in Fig. 20.

The scheme [18], on the contrary, tests 4 CLBs at one time and uses 6 test sessions to test the CLBs. In each test session 2 out of 4 CLBs serve as test-stimuli and response analysis blocks. This means, to find the fault in the set of 4 CLBs, the authors conFig. 4 × 6 = 24 CLBs. We calculate the optimistic fault detection latency by simply accounting the time to configure the CLBs. For this purpose, we use Eq. (2) that *gives 59 words or 236 bytes of data to conFig. 1 CLB*. This means 236 × 24 = 5664 bytes are required to conFig. 24 CLBs. The one-bit wide BSI TAM, running at 66 MHz, takes approx. 680 μs to detect a fault in the set of 4 CLBs. On a scaled level of TCFR with 348 MCRs, the fault detection latency of the scheme [15] accounts to 680 × (5568/4) = 946,560 μs or 946.5 ms.

Similarly, we obtain the fault detection latencies of [15,18] for the smallest TCFR area that we have used for our architecture, i.e., TCFR with 44 MCRs. The fault detection latencies for [15,18] are found to be approx. 5568 ms and 118.3 ms respectively.

6.5.2. Run time (spatiotemporal overheads)

The work of [15] requires 1 additional CLB to replicate the CLB that is going to be tested. This accounts for 100% spatial overhead, i.e., additional 5568 CLBs are required to test a TCFR of 348 MCRs. The *one-bit wide BSI TAM, running at 66 MHz, takes approx. 28.3 μs to conFig. 236 bytes of CLB*. The temporal overhead, i.e., time to configure the spatial overhead of 5568 CLB is approx. 5568 × 28.3 = 157574.4 μs or 157.6 ms, as shown in Fig. 21.

The scheme [18], on the contrary, tests 4 CLBs at one time and uses 6 test sessions for their verification. Each test incurs a spatial overhead of 2 CLBs, because 2 out of 4 CLBs serve as test-stimuli and response analysis blocks. This means, an overhead of 12 CLBs is incurred in 6 test sessions to test 4 CLBs. This is 3 times the CLBs that are going to be tested. On a scaled level, i.e., for the TCFR of 348 MCRs, the total spatial overhead then accounts to 3 × 5568 = 16,704 CLBs. The temporal overhead is, therefore, approx. 16,704 × 28.3 = 472723.2 μs or 472.7 ms, as shown in Fig. 20.

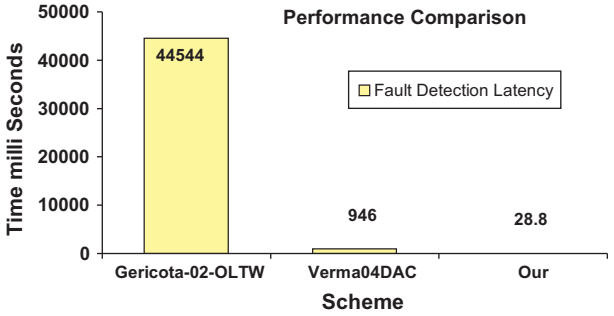

Fig. 20. Per TCFR: fault detection latency (ms).

Similarly, we obtain the spatiotemporal overheads of [15,18] for the smallest TCFR area that we have used for our architecture, i.e., TCFR with 44 MCRs. For scheme [15], the spatial overhead is approx. 704 CLBs and temporal overhead is approx. 19.7 ms. For scheme [18], the spatial overhead is approx. 2088 CLBs and temporal overhead is approx. 59 ms.

6.5.3. Compile time (impact on user application)

From a compile time perspective, we can qualitatively compare our methodology with [15,18] in a sense, that an additional time is required to reserve and allocate the test resources across the HWNoC. In addition, as explained earlier in Section 5.2, we reserved approx. 10% of our HWNoC resources for conducting the online test. Therefore, a user application would be allocated from the remaining 90% resources.

7. Conclusion

We presented an online test methodology for FPGAs that used HWNoC as test access mechanism. Our online test scheme ensured the non-intrusive behaviour by: (a) invoking test at application startup time, (b) allowing un-disrupted execution for already existing applications, and (c) not restricting the parallel operations of dynamic reconfiguration and test for multiple applications. To achieve the objectives, our methodology took into account the design, compile, and run time phases.

We analysed the performance and cost of our test methodology for different test functional regions (TCFRs) of an FPGA architecture. The largest TCFR area was 348 MCRs (5568 CLBs) and the smallest

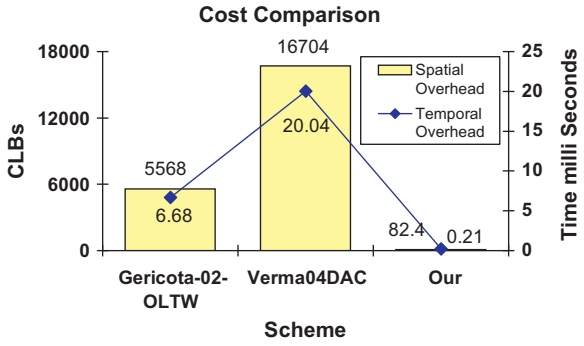

Fig. 21. Per TCFR: spatiotemporal overheads.

one was with 44 MCRs (704 CLBs). Our methodology detected faults in the largest TCFR in 28.8 ms and at the cost of temporal overhead of 0.021 ms and spatial overhead of 82.4 CLBs. Similarly, for the smallest TCFR, the fault detection latency was found to be 3.6 ms and at the cost of temporal overhead of 0.21 ms and spatial overhead of 65 CLBs.

We then compared the performance and cost of our scheme with two other schemes [15,18] for the same set of TCFRs. *For the smallest TCFR* (44 MCRs), the fault detection latency of [15] was approx. 5568 ms with the spatiotemporal overheads of 5568 CLBs and 157.6 ms respectively. Similarly, for [18], the fault detection latency was approx. 118.3 ms and at the spatial cost of 2088 CLBs and temporal cost of 59 ms. Our scheme, therefore, in comparison to [18] possess approx. 32.7 times better fault detection latency. Moreover, it comes at lower spatiotemporal costs which are found to be 67 and 280 times lower, respectively.

References

[1] Xilinx Inc., Virtex-6 Data Sheets, 2009. <http://www.xilinx.com>.
[2] J. Collins, G. Kent, J. Yardley, Using the starbridge systems FPGA-based hypercomputer for cancer research, in: International Conference on Military and Aerospace Programmable Logic Devices, 2004.
[3] C. Chang, J. Wawrzynek, R. Brodersen, BEE2: a high-end reconfigurable computing system, IEEE Design & Test of Computers 22 (2) (2005) 114–125.
[4] S.D. Craven, P. Athanas, Examining the viability of FPGA supercomputing, EURASIP Journal on Embedded Systems (1) (2007).
[5] D.P. Schultz, S.P. Young, L.C. Hung, Method and structure for reading, modifying and writing selected configuration memory cells of an FPGA, Xilinx, Inc., Patent US 6,255,848, August 1999.
[6] B. Dutton, C. Stroud, Single event upset detection and correction in Virtex-4 and Virtex-5 FPGAs, in: International Conf. on Computers and Their Applications, 2009, pp. 57–62.
[7] S. Srinivasan, R. Krishnan, P. Mangalagiri, Y. Xie, V. Narayanan, M. Irwin, K. Sarpatwari, Toward increasing FPGA lifetime, IEEE Transactions on Dependable and Secure Computing 5 (2008) 1050–1069.
[8] Al-Asaad, On-line built-in self-test for operational faults, in: AUTOTESTCON, 2000.
[9] M.G. Gericota, G.R. Alves, M.L. Silva, J.M. Ferreira, AR2T: implementing a truly SRAM-based FPGA on-line concurrent testing, in: European Test Workshop, 2002.
[10] E.S. Reddy, V. Chandrasekhar, M. Sashikanth, V. Kamakoti, N. Vijaykrishnan, Online detection and diagnosis of multiple configuration upsets in luts of SRAM-based FPGAs, in: IPDPS, 2005.
[11] N.R. Shnidman, W.H. Mangione-Smith, M. Potkonjak, On-line fault detection for bus-based field programmable gate arrays, TVLSI 6 (4) (1998) 656–666.
[12] IEEE Computer Society, IEEE Standard Test Access Port and Boundary-Scan Architecture, IEEE Press, 1990.
[13] M. Abramovici, J. Emmert, C. Stroud, Roving STARs: an integrated approach to on-line testing, diagnosis, and fault tolerance for FPGAs in adaptive computing systems, in: NASA/DoD Workshop on Evolvable Hardware, 2001.
[14] M. Abramovici, C. Stroud, M. Lashinsky, J. Nall, J. Emmert, On-line BIST and diagnosis of FPGA interconnect using roving STARs, in: On-Line Testing Workshop, 2001.
[15] M.G. Gericota, G.R. Alves, M.L. Silva, J.M. Ferreira, Active replication: towards a truly SRAM-based FPGA on-line concurrent testing, in: Int'l On-Line Testing Workshop, 2002.
[16] F. Lima, Designing fault tolerant systems into SRAM-based FPGAs, in: DAC, 2003.
[17] S. Dutt, V. Verma, V. Suthar, Built-in-self-test of FPGAs with provable diagnosabilities and high diagnostic coverage with application to online testing, IEEE Transactions on CAD of Integrated Circuits and Systems 27 (2) (2008) 309–326.
[18] V. Verma, S. Dutt, V. Suthar, Efficient on-line testing of FPGAs with provable diagnosabilities, in: DAC, 2004.
[19] K. Goossens, M. Bennebroek, J.Y. Hur, M.A. Wahlah, Hardwired networks on chip in FPGAs to unify data and configuration interconnects, in: NOCS, 2008.
[20] E. Rijpkema, K. Goossens, A. Rădulescu, J. Dielissen, J. van Meerbergen, P. Wielage, E. Waterlander, Trade-offs in the design of a router with both guaranteed and best-effort services for networks on chip, IEE Proceedings: Computers and Digital Techniques 150 (5) (2003) 294–302. http://dx.doi.org/10.1049/ip-cdt:20030830.
[21] A. Rădulescu, J. Dielissen, K. Goossens, E. Rijpkema, P. Wielage, An efficient on-chip network interface offering guaranteed services, shared-memory abstraction, and flexible network programming, Proceedings of Design, Automation and Test in Europe Conference and Exhibition (DATE), vol. 2, IEEE Computer Society, Washington, DC, USA, 2004, pp. 878–883. doi:http://dx.doi.org/10.1109/DATE.2004.1268998.
[22] J.-P. Diguet, G. Gogniat, S. Evain, R. Vaslin, E. Juin, NOC-centric security of reconfigurable SoC, in: NOCS, 2007.
[23] M.A. Wahlah, K. Goossens, Modeling reconfiguration in a FPGA with a hardwired network on chip, in: Proc. Reconfigurable Architecture Workshop (RAW), 2009.
[24] Xilinx Inc., Virtex-4 Configuration Guide. <http://www.xilinx.com>.
[25] G. Nazarian, On-line Testing of Routers in Networks-on-Chips, M.Sc., Thesis, TUDelft, 2008.
[26] M.A. Wahlah, K. Goossens, PUMA: placement unification with mapping and guaranteed throughput allocation on an FPGA using a hardwired NoC, in: DSD, 2011.
[27] A. Hansson, M. Coenen, K. Goossens, Undisrupted quality-of-service during reconfiguration of multiple applications in networks on chip, in: DATE, 2007.
[28] A. Hansson, K. Goossens, A. Rădulescu, A unified approach to mapping and routing on a network on chip for both best-effort and guaranteed service traffic, in: VLSI Design 2007, 2007. Article ID 68432, 16 pages, Hindawi Publishing Corporation. doi:http://dx.doi.org/10.1155/2007/68432.
[29] P. Sedcole, B. Blodget, T. Becker, J. Anderson, P. Lysaght, Modular dynamic reconfiguration in Virtex FPGAs, IEE Proceedings on Computers and Digital Techniques 153 (3) (2006) 157–164.
[30] G.D. Micheli, P. Pande, A. Ivanov, C. Grecu, R. Saleh, Design, synthesis, and test of networks on chips, IEEE Design & Test of Computers 22 (5) (2005) 404–413.
[31] M.N.B.M. Hosseinabadi, A. Banaiyan, Z. Navabi, A Concurrent Testing Method for NoC Switches, in: DATE, 2006.
[32] R. Stefan, A.B. Nejad, K. Goossens, Online allocation for contention-free-routing NoCs, in: Interconnection Network Architecture: On-Chip, Multi-Chip (INA-OCMC), 2012.
[33] M.A. Wahlah, K. Goossens, 3-Tier reconfiguration model for FPGAs using hardwired network on chip, in: FPT, 2009.
[34] M.A. Wahlah, K. Goossens, Composable and persistent-state application swapping on FPGAs using hardwired network on chip, in: ReConFig, 2009.
[35] K. Goossens, A. Hansson, The aethereal network on chip after ten years: goals, evolution, lessons, and future, in: DAC, 2010, invited paper.
[36] A. Hansson, A Predictable and Composable On-Chip Interconnect, Eindhoven University of Technology, 2009.
[37] I. Kuon, J. Rose, Measuring the gap between FPGAs and ASICs, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 26 (2) (2007) 203–215.

Muhammad Aqeel Wahlah has a BSc in Electrical Engineering and Masters in Information Technology from the prestigious institutes of Pakistan named respectively University of Engineering And Technology (UET) Lahore And Pakistan Institute of Engineering and Applied Sciences (PIEAS), Islamabad. Since Nov 2006, he has been a PhD scholar in the computer engineering laboratory of TUDelft. His research is based on Hardwired Network on Chip (HWNoC) in FPGAs, which is carried out under the supervision of Prof. Kees Goossens. During the PhD tenure, he has published 6 International and 2 local conference articles.

Kees Goossens received his PhD from the University of Edinburgh in 1993 on hardware verification using embeddings of formal semantics of hardware description languages in proof systems. He worked for Philips/NXP Research from 1995 to 2010 on networks on chip for consumer electronics, where real-time performance and low cost are major constraints. He was part-time full professor at the Delft university of technology from 2007 to 2010, and is currently full professor at the Eindhoven university of technology, where his research focusses on composable (virtualised), predictable (real-time), low-power embedded systems.

Please cite this article in press as: M.A. Wahlah, K. Goossens, TeMNOT: A test methodology for the non-intrusive online testing of FPGA with hardwired network on chip, Microprocess. Microsyst. (2012), http://dx.doi.org/10.1016/j.micpro.2012.05.011