

A Non-Intrusive Online FPGA Test Scheme Using A Hardwired Network on Chip

Muhammad Aqeel Wahlah
Computer Engineering
Delft University of Technology
Delft, The Netherlands
aqeel@ce.et.tudelft.nl

Kees Goossens
Electronic Systems
Eindhoven University of Technology
Eindhoven, The Netherlands
k.g.w.goossens@tue.nl

Abstract—Modern Field Programmable Gate Arrays (FPGAs) possess small features, and have gained popularity in mission-critical systems. However, due to small FPGA features and harsh external conditions that can be faced by a mission-critical system, an FPGA chip can suffer from faults. This in turn raises the need to test an FPGA to ensure a reliable system performance. However, a mission-critical system requires that the test process should be *non-intrusive*, i.e., applications & FPGA regions that are not being tested remain unaffected. Hence, an online test scheme is required that not only verifies the correctness of an FPGA, but also does not degrade the performance of other, running FPGA applications.

In this paper, we propose a Hardwired Network on Chip (HWNOC) as the Test Access Mechanism (TAM). Our online test scheme uses a HWNOC, to perform real-time streaming of test data that is non-intrusive to other communication traffic. Additionally, our online test scheme exhibits approx. 18 and 29 times lower spatial and temporal overheads as compared to existing schemes, respectively.

Keywords-FPGA; Hardwired NoC; Non-Intrusive; Online Test

I. INTRODUCTION & RELATED WORK

Field Programmable Gate Arrays (FPGAs) have evolved from a simple programmable logic device to complex platform-based devices [23]. These platform-based FPGAs contain millions of gates and use transistor as small as 40 nm. The increased gate density combined with low non-recurring engineering (NRE) costs and in-field products upgrade make an FPGA favorite for implementing System on Chips (SoCs). An SoC can have multiple applications that are dynamically swapped in and out of an FPGA due to area constraints. Before proceeding, we define the terminology that will be used in the paper.

A. Terminology

Typically a *soft* IP is mapped on FPGA reconfigurable computational blocks, e.g., lookup tables (LUTs). An IP is *hardwired* or *hard* when it is directly implemented in silicon, e.g., PowerPC. We define (*re*)*configuration* as the installation of new functionality (of soft IPs) in an 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. A *Usecase* defines the set of applications that execute in parallel, and an SoC can have multiple usecases. *Online testing* verifies the function of an FPGA chip while the system on FPGA is operational. It can be further classified into structural

and functional tests. *Functional test* verifies an FPGA resources for the intended set of system applications [3]. *Structural test* verifies an FPGA resources irrespective of the intended set of system applications [15].

B. Problem Description

Modern FPGA based SoCs can be used for mission-critical purpose, thus requiring them to operate with high reliability. However, due to small features, the reliability of an FPGA decreases [16]. Particularly, in harsh external conditions (e.g., cosmic radiations), the permanent faults (e.g., stuck-at faults) in programmable logic and interconnects of an FPGA can be induced with higher probability [16]. Hence, to ensure a reliable system performance, *online testing* of an FPGA chip is enforced [1], [18]. However, due to the mission-critical nature of an FPGA system, an online test can not be performed on the whole FPGA chip, simultaneously. Conventionally, it is performed after dividing an FPGA into multiple (logical) regions, which can be: i) a single configurable logic block (CLB) [3], ii) or a single / multiple columns of CLBs [15], [18]. 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 behavior of an on-FPGA system can be viewed from two aspects, i.e., system behavior in RUTs and system behavior in RNUTs.

Ideally, an online test process should be transparent to RNUTs, which means: a) programming and execution of SoC application that reside on RNUTs is not disrupted, b) and (if required) bitstream of new SoC applications is configured onto RNUTs. In other words, during online testing, the normal operation of RNUTs is not affected in terms of programming, execution, and configuration. Additionally, an ideal online test scheme should possess certain characteristics [19]. For instance, it should detect high percentage of faults (ideally 100% faults). It should have insignificant fault detection latency (ideally 1 cycle). Moreover, the *spatial* and *temporal* overheads that are induced by an online test scheme, should be insignificant. Here, spatial overhead accounts for an additional test hardware [1], e.g., test pattern generators (TPGs) and output response analysers (ORAs), which are needed to perform the online verification of an FPGA region. The TPGs and ORAs can be made up of FPGA reconfigurable resources, i.e., CLBs and interconnection wires, etc. [18]. The temporal overhead is determined by the amount of time required to configure and

use the spatial overhead.

We next expound the existing online test schemes and compare these with our proposed online test approach.

C. Related Work

In the literature, a number of online test schemes [1], [2], [3], [4], [14], [15], [18], [20] that detect faults for an FPGA can be found. The existing schemes scan through the FPGA chip for finding the perspective faults. In these schemes, a relatively small portion of an FPGA chip is taken off-line, while allowing the rest of an 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 I, use a conventional boundary scan infrastructure (BSI) as their TAM to perform structural [1], [4], [18] or functional [2], [3], [20] testing. The schemes can have different levels of test granularity, ranging from a single CLB [3] to multiple CLB columns [1]. The main focus of the existing online test schemes have been on maximising the percentage of fault detection with the minimum fault detection latency. However each one of these induces a level of intrusiveness, which could be a partial or a complete application, Table I. Additionally, none of these schemes, due to the limitations of existing TAM architecture, can achieve the interleaved test and configuration operations for multiple applications.

In contrast, our scheme uses a HWNoC as the TAM to perform an online structural test. We avoid any level of intrusiveness by triggering the test at the time of an application startup. The test is conducted for the application configuration functional regions (CFRs). A CFR consists of multiple CLB columns and its architecture is explained in Section III. The test is conducted through the HWNoC by establishing a connection in between the system manager (Section IV) and the destination CFR. Moreover by using the same HWNoC, we can achieve the interleaved operations of test, configuration, programming and execution during an ongoing test process.

In the remainder of this paper we present the details of our online test scheme by, explaining its basic idea (Section II-A), presenting the motivation for such a scheme (Section II-B), expounding the target platform that is being tested (Section III), and design flow to perform online testing (Section IV). We conclude at the end in Section VI.

II. BASIC IDEA & MOTIVATION

In this section we first explain the basic idea of our online test scheme. Afterwards, we motivate the need for such an online test scheme with the help of a case study.

A. Basic Idea And Contributions

In this paper, as a *contribution*, we propose a *non-intrusive* online test scheme that:

| Scheme | TAM | Test Type | Test Granularity | Intrusiveness Level | Test & Load |
|------------|--------------|-----------|------------------|---------------------|-------------|
| [1] | BSI | St | CLB Col | (P/C) App. | No |
| [4] | BSI | St | CLB | (P/C) App. | No |
| [15] | NA | St | CLB Col | (P/C) App. | No |
| [18] | BSI | St | CLB Col | (P/C) App. | No |
| [2] | BSI | Fu | NA | C. App. | No |
| [3] | BSI | Fu | CLB | P. App. | No |
| [20] | BSI | Fu | CLB Col | P. App. | No |
| Our | HWNoC | St | CFR | None | Yes |

Table I
OUR WORK POSITIONING W.R.T. EXISTING ONLINE APPROACHES;
ABBREVIATIONS, NA = NOT APPLICABLE, COL: COLUMN, ST:
STRUCTURAL, FU: FUNCTIONAL, P/C: PARTIAL/COMPLETE

- 1) uses a hardwired NoC as a test access mechanism (TAM),
- 2) tests un-disrupted in parallel with other application(s) configuration, programming, and execution,
- 3) tests at an application startup, i.e., before the configuration of an application,
- 4) uses structural test for the correctness of an FPGA,
- 5) reduces spatial and temporal overheads, with respect to conventional online test schemes.

1 & 2) HWNoC as TAM: Our FPGA is comprised of multiple configuration functional regions (CFRs). The online testing is performed at the granularity of CFR. For verifying the (structural) correctness of a CFR, our online test scheme uses a HWNoC [5], which has been extended in this paper to serve as a TAM. The extended HWNoC can now transport four types of data, i.e., test, configuration, and functional (programming and execution) data. For that purpose, it uses connections. The connections 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 does not produce interference with each other. This means, while testing, it is possible to achieve: a) un-disrupted execution for already running application(s), b) and the operations of configuration, programming, and execution for new applications.

However, as we shall explain in Section IV-B, the HWNoC connections must be allocated appropriately to achieve the above mentioned objectives (a & b). The reason is that at the time of online testing, the system on an FPGA can be in any of its usecases. Each usecase can have its own Quality of Service (QoS) [7] constraints that the HWNoC must efficiently fulfill. In such a situation, even a single connection that is allocated inappropriately can induce a certain level of intrusiveness with respect to achieving the above mentioned objective (a & b).

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 CFR(s), simultaneously. However, multiple CFRs can be occupied by a single application for its execution. In our online test scheme, before configuring an application the associated CFRs of an application are tested. However, while an application is executing, its associated CFRs are not tested.

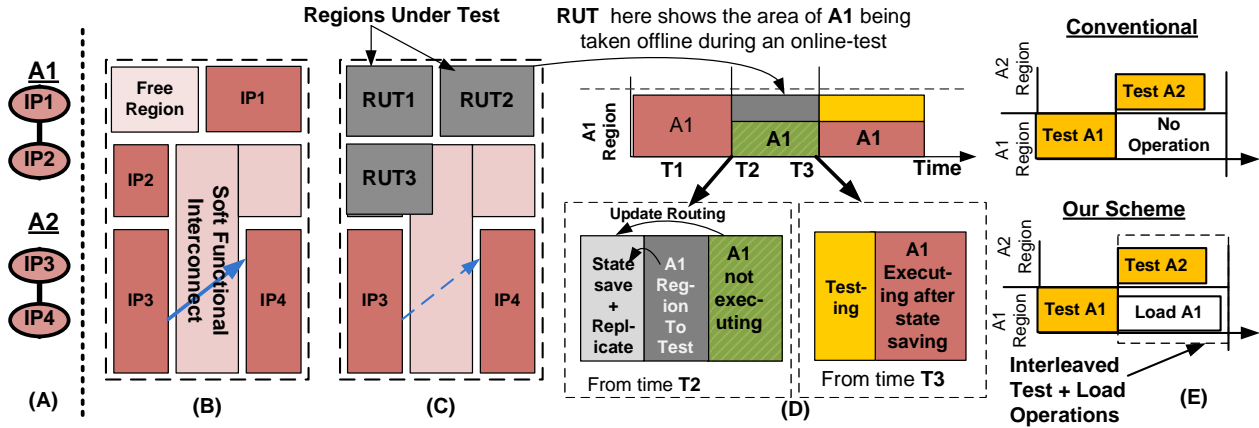


Figure 1. Application With: (A) Task Graphs; (B) FPGA binding; (C) Architectural and (D) Application Level Test Procedure Elaboration; (E) Abstract Comparison Of Our Scheme With Existing Conventional Schemes.

By doing so, the disruption in application execution that could be caused by the test procedure, is avoided. On the negative side, the application configuration is delayed by an amount of testing time. Additionally, a fault that had occurred during an application execution time, can only be detected after the application had finished execution and another application is invoked onto the same CFR(s).

4) *Structural test*: Our online scheme performs the structural test and currently detects the permanent stuck-at¹ faults. The structural test provides us with the increased reusability, which can be exploited in multiple ways. i) A single test suite (bitstreams and stimuli for an FPGA CFR) irrespective of the intended set of applications that run on the CFR. This means the test bitstream for a single CFR can be reused for multiple CFRs. ii) A single test (over a certain period of time) for multiple time-multiplexed applications executing on the same CFR(s). This means the result of a structural test can be reused for multiple applications that execute on same CFR(s).

5) *Reduced Spatiotemporal Overheads*: Moreover, the detection of structural faults in a CFR is performed by making use of HWNoC connections, and the system manager (SM) [21]. The same HWNoC is used for transporting functional data, whereas the SM can execute on a programmable hardware processor, e.g., PowerPc. The *additional resources* are in the form of test connections and the code to perform the analysis. In our scheme, no specialised test hardware (e.g., TPGs and ORAs) are imported to FPGA logic plane. This in turn eliminates the spatiotemporal overheads that are required to place and configure the test hardware on an FPGA logic plane.

In the next section we build the motivation behind using these parameters for our scheme.

B. Motivational Case Study

For motivation purpose, we consider an SoC that comprises 2 applications, Figure 1A. Figure 1B shows the binding of these applications on the logical regions of an FPGA. 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 scheme, we examine the possible shortcomings that can arise while testing the SoC by using conventional test schemes, such as [3], [20].

An RUT in existing schemes, e.g., [1], [20] can be considered as a set of wires and CLBs. The conventional schemes [3] scan through an FPGA to find out faults. This means testing is performed by roving an RUT across the chip. RUT1, RUT2, and RUT3 in Figure 1C, indicate three such roving instances. Conventionally, the region under test is taken off-line, while allowing the rest of an 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 the functionality of the application (i.e., A1) is first replicated and saved in an already tested region. Additionally, the FPGA interconnection is routed accordingly for intra-application execution. In Figure 1D, time T2 indicates such a situation. However, this process requires an application (i.e., A1) to be stopped, atleast during the state saving and routing update periods. Afterwards, testing and application execution can be performed independently, as shown with T3 in Figure 1D. 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 on pre-tested regions.

In conventional test schemes [2], [20] application IPs and associated functional interconnect both coexist in the same reconfigurable plane, Figure 1B. Therefore, an RUT: i) can be an IP only (RUT2), ii) or can be comprised of IP plus interconnection (RUT3). The second situation could even disrupt another application (in this case A2), whose region is not aimed for testing. Because, during the state saving period, the soft functional interconnect is unavailable for the A2 traffic. The broken virtual connection in Figure 1C, illustrates this situation. Hence, it motivates us to have an application and the functional interconnect in *disjoint planes*, which is ensured by hardwiring the communication architecture (i.e., HWNoC) in an FPGA.

¹A logic block or wire is said to experience a stuck-at fault when its logic value always stays at 1 or 0 and can not be reversed.

Additionally, the existing approaches [2], [3] make use of boundary scan infrastructure (BSI) [17], which occupies an FPGA configuration circuit during the test operation. This means, the inherent nature of the TAM in these schemes doesn't 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 Figure 1E, the parallel operations of A1 testing and A2 loading (on a pre-tested region) are not possible with existing schemes, such as [2], [3]. Hence, imposing a delay on either test or configuration process. This motivates us for a TAM which does not interfere with the bitstream loading of an application.

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

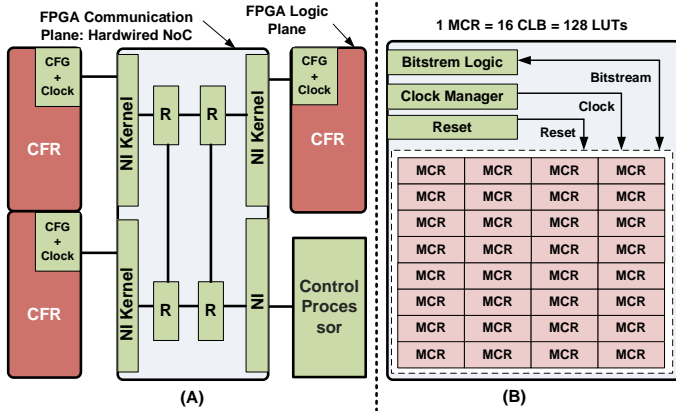


Figure 2. Showing: (A) FPGA With HWNoC, (B) CFR Architecture

III. TARGET FPGA ARCHITECTURE

Our architecture comprises an FPGA with: i) a reconfigurable logic plane [21], ii) a hardwired communication plane [5], iii) and a control processor. In further discussion we will explain them separately.

The FPGA logic plane consists of multiple CFRs, Figure 2A. The detailed architecture of a CFR, on which an application IPs are placed, is illustrated through Figure 2B. This shows that each CFR constitutes a number of minimum configuration regions (MCRs), where each MCR consists of 16 CLBs [22]. The CLBs are connected through programmable interconnect. Each CFR has a local *Clock manager*, which is memory-mapped, i.e., programmable clock frequency for the required MCR can be generated by writing to the registers that are accessible over the NoC. Similarly, a memory-mapped *Reset controller* is present, by using which the IP cores can be enabled or disabled from processing the input data. As shown with Figure 2B, a CFR has localised configuration infrastructure for bitstream writing. In this paper, the configuration infrastructure has been extended

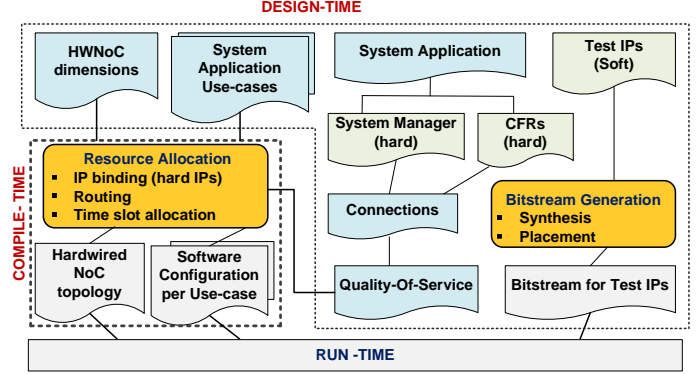


Figure 3. Design, Compile And Runtime Flow Interaction

to provide read-back facilities for a configuration bitstream. Importantly, the logic plane of our FPGA is not divided into multiple disjoint regions, which means CFRs are not isolated at functional level. This in turn, allows a soft IP to span in multiple CFRs, and hence removes the restrictions on IP placement. We assume a homogenous logic plane that consists of CLBs and programmable interconnect. The logic plane currently does not possess special computational / storage hardware, e.g., Block RAM, DSP units, etc.

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 possess the architectures as detailed in [13], whereas a Network Interface is further split into NI kernels and NI shells. NI kernel is hard and its architecture is explained in [5], [12]. 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. The *control processor* [21], which is connected with the first NI of the HWNoC, is used to bootstrap an FPGA.

The next section elaborates our proposed online test scheme, which poses a non-intrusive nature.

IV. DESIGN FLOW & METHODOLOGY

In this section we will explain our online test methodology for verifying the correctness of our FPGA logic plane, i.e., the CFRs. Likewise conventional FPGAs [23], the CFRs in our FPGA use (SRAM based) memory cells to store the configuration bits that implement the required combinational and sequential logic. However, a fault occurred, e.g., due to radiation affects can be latched by memory cells of the logic plane [10]. This in turn causes the memory cells to get stuck-at wrong values, which is crucial because even a transient fault can then be turned into a permanent one. Hence, detection of faulty CFRs is important to take appropriate corrective measures [10]. Currently, we focus on verifying the correctness of FPGA logic plane and don't test the HWNoC. However, the technique proposed in [11] can be used for the verification of our HWNoC routers.

To meet the objectives of Section II-B, our online test scheme takes into account design, compile, and run time phases that

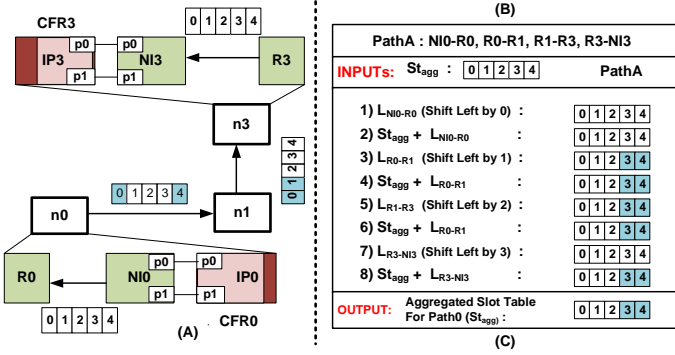


Figure 4. Slot Allocation Across The Path-Links

are explained in the following Sections.

A. Design Time Phase

At design time, we specify: a) the HWNoC dimensions and number of CFRs in an FPGA, b) the system application, c) and two *test IPs*. Initially, the hardware description of both the *test IPs* is fed to the bitstream generation tool (e.g., Xilinx ISE) to obtain their bitstreams. The bitstream of each *test IP* stands for a complete CFR bitstream. We assume relocatable bitstream for the *test IPs* that can be loaded to (any) required CFR for locating the stuck-at faults.

The system application, which is responsible for configuration and online testing of CFRs, is characterised as $SA = (IP, C)$. Here, *IP* accounts for CFRs and the SM, which is mapped onto the programmable control processor. On the other hand *C* represents the communication connections from the SM to CFRs. The resources for each connection are reserved during the compile time, as explained in the next Section.

B. Compile Time Phase

The inputs for this phase include: a) HWNoC dimensions, b) the system application usecases, c) and connections with required QoS constraints. While testing, the system behavior should not be affected in regions that are not under test. This in turn requires an independent virtual test platform for ensuring configuration, programming, and execution of user application(s) in RNUTs. Hence, to ascertain such an online test scheme the resources (paths, QoS and Flow control) should be reserved for the system application in all the usecases (U_i) of an SoC. Here $U_i = \{u_0, u_1, \dots, u_n\}$, constitutes jointly exhaustive and mutually exclusive subsets whose union completes an SoC usecases.

The resource reservation for system application starts with selecting the connections (from the SM to CFRs). Then, allocating the selected connection by using the approach of [7], [8] such that the required contention free resources are reserved. Our online test scheme can allow interleaved test and load operations for multiple applications. Therefore, the allocated connections are used for both of these operations, but by using independent ports of the system manager.

A connection requires a path for transporting data, but with certain constraints on its throughput demands. The throughput

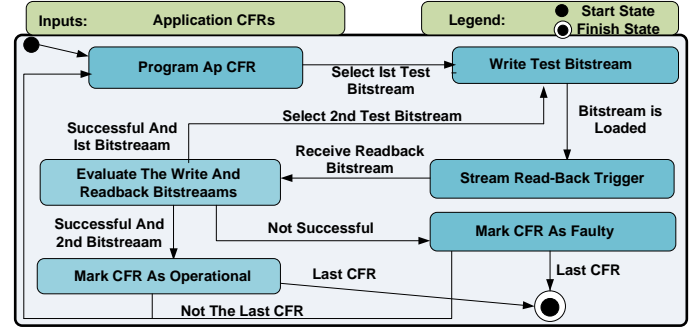


Figure 5. RUN-TIME Flow For The Test Process

demand can be converted into discrete number of time-slots. In our test scheme, a connection from the SM to a particular CFR is then said to be allocated: i) if it finds the path to the CFR, ii) and 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. For instance, for path-links L0 and L1, each of which has a slot table St of size N , the time-slots should be positioned such that:

$$IF \text{ slot}_n \in St_{L0} \text{ THEN } (\text{slot}_{n+1} \% |N|) \in St_{L1} \quad (1)$$

The above equation can be implemented by constructing an aggregated slot table, which represents the complete path slot table. To understand this, we refer to Figure 4, which shows 3 nodes of our FPGA architecture, as explained earlier in Section III. Figure 4A, shows the slots tables associated with each link. Each slot table comprises 5 time-slots. The dark colored slots represent the allocation for user applications (not shown here for the simplicity). Hence, an IP on CFR0 if communicates with an IP on CFR3, and it uses a path as shown in Figure 4B. 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 [8], which is based on merge and shift process, as shown in Figure 4C. As a starting point, the inputs of an empty slot table and the required path are provided, Figure 4C. Afterwards, the links are shifted and merged with the input slot table, step 1 to step 8 in Figure 4C. As, its a shift and merge operation. Therefore, in the aggregated slot table an empty slot on a particular position is found, if and only if it is empty across all the path-links, as shown with OUTPUT in Figure 4C.

C. Run Time Phase

At run time, the online test is performed when an application is started. The test is carried out for the application CFRs. The online test scheme targets stuck-at fault model, therefore it is carried out with two complementary bitstreams. A 4 phase (program, write, read and evaluate) test process is carried out for each bitstream, Figure 5. The test process starts with programming a test connection for the required CFR. This is followed by writing the test bitstream to that CFR. Once writing is finished, a read-trigger is sent to that CFR for reading back the bitstream. To find out the faults, the read-back bitstream is

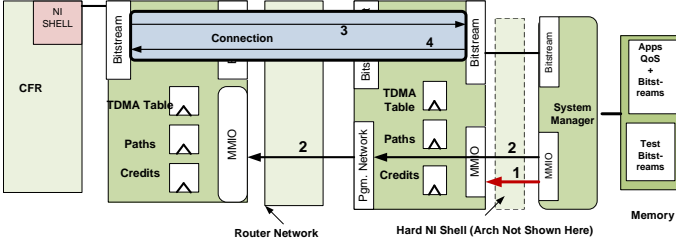


Figure 6. Test Connection Setup From SM to CFR

then evaluated against the originally written bitstream. We will expound each of these phases as following.

Program: Figure 6, illustrates the steps involved in establishing a connection from the system manager to a CFR. 1) It starts with programming the required tables (path, slot, and channel tables) of the system manager NI, so as to reach and program a CFR NI. 2) After reaching a CFR NI the system manager programs the CFR NI tables by sending path, slot and flow control information. The programming of CFR NI ensures a contention free path in between a CFR and the system manager. 3) A response channel, which is used for test bitstream read-back, is then opened from a CFR to the system manager. 4) This is followed by opening a request channel, which is used for test bitstream writing, from the system manager to a CFR.

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 CFR. The test bitstream is sent frame-wise over an established test connection, and the process is repeated until the last set of test frames is sent to the destination CFR.

The destination CFR, after finding a 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 for activating the respective MCR, algorithm 1 (line 5-9). Onwards, the input frame register (IFR) is used to store the test bitstream frames, algorithm 1 (line 11). Figure 7, shows the portion of CFR configuration architecture that is used for test bitstream writing.

In our design, the frame length is 41 words and hence the size of the IFR. A 1 word wide data-bus is connected to the IFR that goes through all the MCRs, Figure 7. The data-bus

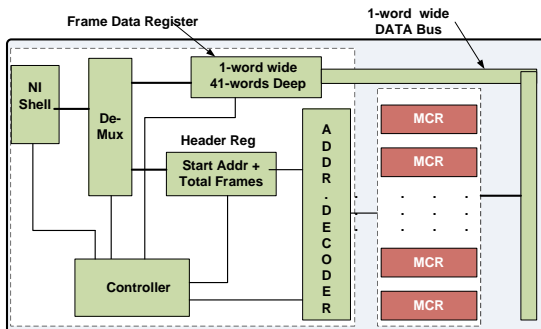


Figure 7. Showing: Configuration Architecture Used To Write Test Bitstream

is designed as one word wide for area saving purpose, 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 an MCR are shifted to the next locations, serially. This forms a scan chain, which spans the whole MCR, algorithm 1 (line 15-26). Meanwhile, the IFR data is shifted rightwards by an amount of sent words, algorithm 1 (line 30-36). The process repeats until the full frame is loaded into the respective MCR, algorithm 1 (line 12-43). The online test process loops over the required number of bitstream frames to load the test IP over the respective CFR.

Algorithm 1: WRITING The Test Bitstream AT CFR

```

// INPUT: bitstream
CFRMCRs = 32, MCRFrms = 23, FrmBytes = 164, WrdBytes = 4,
IFRsize=41, FrmWrds = FrmBytes / WrdBytes.
//Each Time On Receiving the Test Bitstream
IF (Header)
    InHeader = getFrame(bitstream)
    FrmCnt = getFrameCount(InHeader)
    StFrmAddr = getStartFrameAddr(InHeader)
    CurMCR = setAddressDecoder(StFrmAddr, MCRFrms)
ELSE
    IFR = getFrame(bitstream) //IFR = In Frame Register
    For(i =0; i < FrmWrds; i++) // Loop over Frame Words
        // In the following loop, Shift-right the MCR-words (4 CFR
        // bytes) before pushing a word from the DataBUS to an MCR.
        // for instance if MCR=1, j=1, then CFR bytes Index = 3776.
        IF (LoadWrd > 0)
            For(j = LoadWrdNo; j>0; j--)
                Indx = CurMCR * MCRFrms * FrmBytes + j*WrdBytes
                CFR[Indx-1 + WrdBytes] = CFR[Indx-1] //Shift-Byte
                CFR[Indx-2 + WrdBytes] = CFR[Indx-2]
                CFR[Indx-3 + WrdBytes] = CFR[Indx-3]
                CFR[Indx-4 + WrdBytes] = CFR[Indx-4]
            ENDFor
        ENDIF

    DATABUS(InFrmRg, RegAddr, CurMCR, CFR) //Push Word-to-MCR
    LoadWrd++

    // Shift-Right i.e. 40th -> 41st Register in IFR etc.
    // Shifts Required: IFRsize - (Loaded Words In MCR)
    IF (LoadWrdNo < FrmWrds)
        For (i=0; i < (IFRsize-LoadWrdNo); i++)
            Indx = (IFRsize * WrdBytes) - (i*WrdBytes)
            IFR[Indx-1] = IFR[Indx-1 - WrdBytes] //Shift-Byte
            IFR[Indx-2] = IFR[Indx-2 - WrdBytes]
            IFR[Indx-3] = IFR[Indx-3 - WrdBytes]
            IFR[Indx-4] = IFR[Indx-4 - WrdBytes]
        ELSE // When All Frame Words Loaded:
            LoadWrdNo = 0;
            CurFrmNo++;
            LoadFrmNo++;
            CurMCR = setAddressDecoder(LoadFrmNo, MCRFrms)
        ENDFor
    ENDFor
ENDIF

```

Read-back: The system manager after writing test-stream, sends a read-triggered bitstream. It is comprised of the address of the first CFR frame, and total number of frames to read. In response to this read-triggered bitstream, a CFR 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 CFR 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 CFR. It is then sent over the response channel of the already established test connection. The whole

Table II
IP SYNTHESIZED AREA, FREQUENCY AND BITSTREAM FRAMES

| IP | Area (<i>kLUTs</i>) | Frequency (<i>MHz</i>) | Bitstream (<i>Frames</i>) |
|-----------|--------------------------|-----------------------------|--------------------------------|
| Residue | 1.68 | 100 | 285 |
| DCT | 2.36 | 66 | 396 |
| Quantizer | 2.21 | 75 | 370 |

process continues until all the CFR 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. As stated before, two complementary bitstreams are required to identify possible stuck-at 0 and stuck-at 1 faults in a CFR. However, the second bitstream is sent only and only if the evaluation process is successful for the first test bitstream, Figure 5. 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 CFR. This allows us to save the time that can be spent in testing a CFR, which has already been detected faulty. Currently, we replace a faulty CFR with the one that has already passed the structural test.

V. RESULTS AND ANALYSIS

We exercised our online test scheme in SystemC using the design flow of [6]. We used two complementary test bitstreams to verify a CFR for stuck-at fault model. The size of each test bitstream is estimated from the following equation:

$$(IP_{LUTs} * MCR_{frames}) / (CLB_{LUTs} * MCR_{CLBs}) \quad (2)$$

For Virtex-4 [24] the minimum configuration region, i.e., an MCR consists of a column of 16 CLBs. Each CLB contains 8 LUTs and associated programmable interconnect. Notably, in our calculations, a *CLB* stands for a CLB unit and its associated programmable interconnect. Moreover, an MCR is configured by using 23 41-word frames [24]. In our architecture a CFR comprises 32 such MCRs. This means, the test IP bitstream consists of $32 * 23 = 736$ frames. The HWNoC platform runs at 500 MHz, and consists of 4 routers and NI kernels with FIFO sizes of 3 and 41 words, respectively.

For our online test scheme, we provide: i) an evidence of its non-intrusive nature, ii) fault detection latency, iii) and the comparison with two state-of-the-arts [4], [20].

A. A Non-Intrusive Test Scheme

In this section we analyse the non-intrusive nature of our scheme. We used the behavioral 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 provided their MHz frequencies, which were used in SystemC, Table II. The size of their bitstreams was estimated from their *kLUT* areas using the equation 2.

To verify the non-intrusive nature of our online test scheme, we used a system with three applications: A0 (Residue + DCT), A1 (DCT only), and A2 (Quantiser only). A0 and A1 can execute in parallel and belong to usecase 0 (uc0), whereas A1

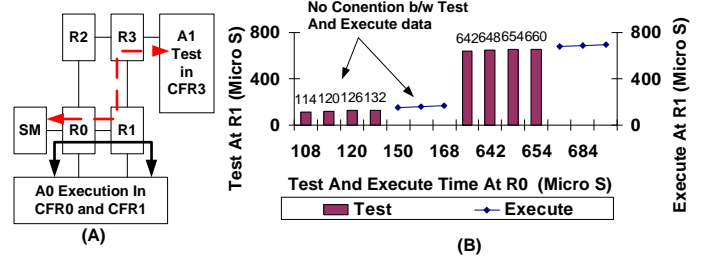


Figure 8. Non-Intrusive During Application Execution

and A2 can execute in parallel and belong to usecase 1 (uc1). Hence, there are two usecases in the system. Moreover, A0 and A2 represent two sub-applications of one larger application, and are time-multiplexed over the same set of CFRs, i.e., CFR0 and CFR1. On the other hand A3 can execute on CFR3, as shown with Figure 8A.

For evaluating the non-intrusive nature of our online test scheme, we analysed the behavior of the above-mentioned system. While testing the system, different operation (execution, configuration, programming, and use-case transition) were performed for its applications. It was done by: a) conducting A1 test while A0 was executing, b) and at the end of A0 execution (A1 test still in progress) by starting up application A2. Figure 9A shows that both A0 and A2 are time-multiplexed in a relatively shorter time. We assume that during A0 execution CFR0 and CFR0 don't get faulty. Therefore, it allowed us to trigger the bitstream loading of A2 without conducting A2 testing. By doing so, we aimed to observed the impact (if any) of configuration (A2) on the online testing (A1). Figure 9A, illustrates, the above discussion with test, load and execution timings for these applications.

Important observations can be drawn from the above scenario. For instance, as shown in Figure 9A, the A2 startup triggered the usecase transition, which should be transparent to the ongoing A1 test process. We analysed, the behavior of our scheme during that time and took a small portion of $300\mu s$, as illustrated with Figure 9B. It shows the presence of all the three operations for different applications, i.e., execute (A0), test (A1) and load (A2). In Figure 9B, the first rectangle highlights the unaffected behavior of A1 test, while a usecase transition was made due to A2 startup. Importantly, as shown in Figure 9B, our scheme supports the interleaved test and load operations for multiple applications. Additionally, Figure 8A, illustrates that both A1 test and A0 execution shared a network path (R0-R1). The shared path can lead to intrusiveness in between the test and the execution operations. However, Figure 8B illustrates that test and execution data for multiple applications, over the shared R0-R1 link, don't produce any contention. It is made possible by making sure that the test data is forwarded across the shared path, on fixed and non-contending time-slots, Figure 8B.

B. Fault Detection Latency

Figure 10(A), illustrates the latency to detect a fault in an application CFR. It is a 4 phase process for each test

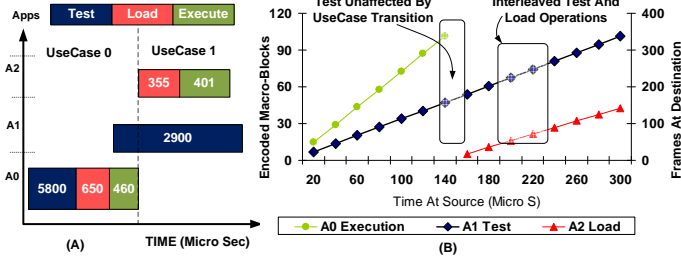


Figure 9. (A) Application Timing Details In Different Phases, (B) Interleaved Test And Load For A1 and A2

bitstream and was started after programming the test connection in $10.6 \mu s$. We reserved approx. 10% of the HWNoC link resources for our connections, which accounts for 8 out of 80 time-slot of a link slot table. As each test connection consists of a request and a response channel, the slots were equally divided (4 slots) in between the two. Each time-slot can transmit a single flit of 3 words. In each slot table iteration, where 4 slots were reserved for a request or response channel, the actual payload of 11 words was transmitted, Figure 10B. 1 word was used as header, which contained the path to the destination CFR.

Hence, to write 736 frames of tests IP bitstream on a CFR, it took approx. 2744 slot table iterations. Due to the assurance of resources at compile time, the flits were transported across the connection links in a pipelined fashion, Figure 10C. Here two flits took $0.30 \mu s$ to reach a CFR that was 3 routers away. In our HWNoC executing at 500 MHz, a router took $(0.002 * 3) = 0.006 \mu s$ to transport a flit to the next destination. On the contrary, a CFR was executing at a frequency of 250 MHz, and followed the procedure of Section IV-C to receive the incoming bitstream. It took approx. $0.17 \mu s$ to write a test bitstream frame to the desired MCR location. The test bitstream writing was completed in approximately $660 \mu s$.

Read-back process was the inverse of the writing process. It took approx. $662 \mu s$ to read-back the complete test bitstream. It was followed by the evaluation of the received bitstream against the originally written bitstream. Each bitstream comprised 30176 words. Therefore $120.7 \mu s$ were taken by the SM, which was executing at 250 MHz, to find out the perspective stuck-at faults in a CFR. Hence, the total fault detection latency for a single bitstream stayed at $1445.6 \mu s$. With two test bitstreams, the worst case fault detection latency for a CFR was found to be approx. $2891.2 \mu s$.

C. Comparison with the State-Of-The-Art

We compared our scheme with the existing state-of-the-arts [4], [20]. As a *fairness of comparison*, we used the same platform of Virtex-4 for each of these schemes. We provide the cost of our scheme at compile time and run time, and compare it with [4], [20]. The run time phase comparison is quantitative that explains the spatiotemporal overheads and fault detection latency of our scheme. On the other hand the compile time comparison is qualitative and indicates the affect of our online test scheme on the allocation of user application resources.

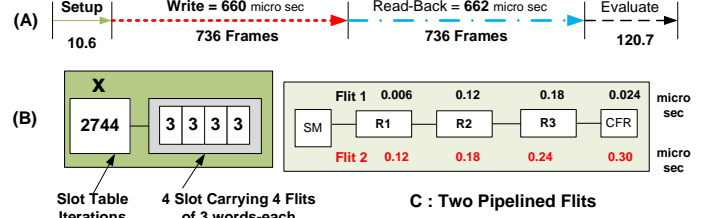


Figure 10. (A): fault detection Latency For A Test-Stream, (B) Frame Data In SlotTable, (C) Time For 2-Pipelined Flits To Reach A CFR 3-hops Away(μs),

1) *Run Time (Spatiotemporal Overheads)*: We used a test connection to transport test data. Therefore, resources acquired by the connection were considered as the spatiotemporal overheads of our scheme. The temporal overhead of the test connection was found to be approx. $21 \mu s$, which is the time required to setup a connection, Figure 11 [5]. The spatial overheads of a test connection, between the source SM and the destination CFR included: a) NI-Shells to (de)serialise frame, b) and 41 words FIFOs at NI kernels to send / receive a test frame. Additionally, the FIFOs (3 words deep) of each router that were used during the connection time were also taken into account. The connection resources were hardwired, and to obtain the (ASIC) area numbers, we referred to [9]. The authors in [9], illustrate that an ASIC implementation can take approx. 35 times lower area than its equivalent FPGA implementation. Therefore, we first synthesised (on Virtex-4 XC4VLX200 chip and by using Xilinx ISE 10.1) the connection resources, then reduced these by (conservative) 30 times to obtain the ASIC equivalents. The overall hardwired spatial overhead was found to be equivalent to 27 CLBs per CFR, Figure 11.

In comparison, the work of [4] tests at the granularity of a single CLB, but after replicating it on an already tested CLB. Hence, this results in 100% spatial overhead. To find out the temporal overhead, i.e., the time to configure the spatial overhead (1 CLB), we used equation 2. It gave us 10 bytes (80 bits) to configure 1 CLB. In [4] authors used BSI, which is 1-bit wide and runs at 20 MHz. However, *as a fairness*, we used Virtex-4 BSI (66 MHz) for [4]. This gave us a temporal overhead of $1.2 \mu s$ for testing a single CLB. On a scaled granularity of a CFR, i.e., 512 CLBs, the temporal overhead for [4] raised to approx. $614 \mu s$, Figure 11.

The scheme [20] on the contrary, tests 4 CLBs at one time and uses 6 test sessions for their verification. In each test session 2 out of 4 CLBs are served as test-stimuli and response analysis blocks. Hence resulting in a 100% overhead. The cumulative spatial overhead for these 6 sessions turns out to be 12 CLBs, which causes a temporal overhead of $14.4 \mu s$. Figure 11, indicates the spatiotemporal overheads of [4], [20] at the granularity of a CFR.

2) *Run Time (Fault Detection Latency)*: [4] uses a TAM at 20 MHz to replicate and test a CLB in 24 ms. However, *as a fairness of comparison*, we used the same Virtex-4 features for all the three test schemes, i.e., ours, [4] and [20]. Therefore, [4], fault detection latency for a single CLB reduces from 24 ms to 8 ms with Virtex-4 BSI, which runs at 66 MHz.

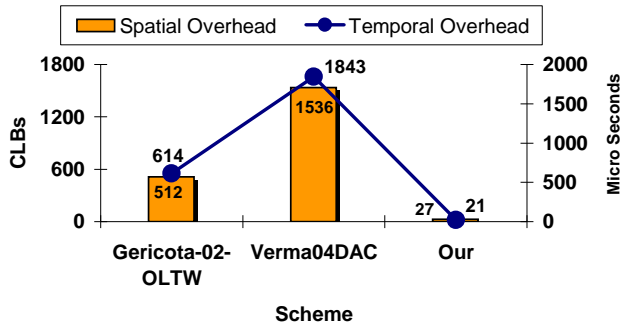


Figure 11. Per CFR: Spatiotemporal Overheads

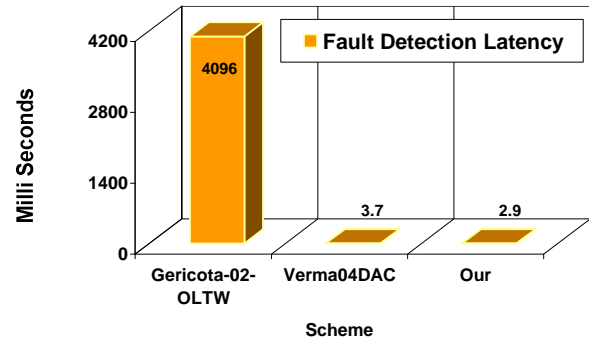


Figure 12. Per CFR: fault detection Latency (mili sec)

For scheme [4], the fault detection latency increases to 4096 ms for verifying a single CFR, Figure 12.

On the other hand the optimistic fault detection latency of [20] is approx. 3.7 ms. Our scheme, in comparison to [4] has 1412 times less fault detection latency and approx. the same as that of [20], Figure 12. However, in contrast to [20] functional test we run the structural test, which ensures 100% stuck-at fault detection.

3) *Compile Time (Impact on User Application)*: From a compile time perspective, we can qualitatively compare our scheme with [4], [20] 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 V-B, 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.

VI. CONCLUSION

We presented an online test scheme for FPGAs that used a HWNoC as the TAM. Our online test scheme ensured a non-intrusive behavior by: (a) invoking test at application startup time, (b) allowing un-disrupted execution for already existing applications, (c) and not restricting parallel operations of reconfiguration and test for multiple applications. To achieve the objectives, our scheme took into account design, compile, and run time phases. Additionally, when compared to [4], our scheme possessed 18 times lower spatial overhead and 29 times lower temporal overhead.

REFERENCES

- [1] M. Abramovici, et al., "Roving STARS: An Integrated Approach to On-Line Testing, Diagnosis, and Fault Tolerance for FPGAs in Adaptive Computing Systems," in *Workshop on Evolvable Hardware*, Jul. 2001.
- [2] Al-Asaad, et al., "On-line built-in self-test for operational faults," in *AUTOTESTCON*, 2000.
- [3] M. G. Gericota, et al., "AR2T: Implementing a Truly SRAM-based FPGA On-Line Concurrent Testing," in *European Test Workshop*, 2002.
- [4] M. G. Gericota, et al., "Active Replication: Towards a Truly SRAM-based FPGA On-Line Concurrent Testing," in *OLTW*, July 2002.
- [5] K. Goossens, M. Bennebroek, J.Y. Hur and M.A. Wahlah, "Hard-wired networks on chip in FPGAs to unify data and configuration interconnects," in *NoCS*, Apr. 2008.
- [6] K. Goossens, et al., "A design flow for application-specific networks on chip with guaranteed performance to accelerate SOC design and verification," in *DATE*, Mar. 2005.
- [7] A. Hansson, et al., "Undisrupted quality-of-service during re-configuration of multiple applications in networks on chip," in *DATE*, Apr. 2007.
- [8] A. Hansson, et al., "A unified approach to mapping and routing on a network on chip for both best-effort and guaranteed service traffic," in *VLSI Design*, May 2007.
- [9] I. Kuon, et al., "Measuring the gap between FPGAs and ASICs," *IEEE Trans. on CAD of Int. Circ. and Sys.*, 26(2):203215, Feb. 2007.
- [10] F. Lima, et al., "Designing Fault Tolerant Systems into SRAM-based FPGAs," in *DAC*, June 2003.
- [11] GH. Nazarian, "On-line Testing of Routers in Networks-on-Chips," in *MSc. Thesis, TUDelft*, Dec 2008.
- [12] A. Rădulescu et al., newblock "An Efficient On-Chip Network Interface Offering Guaranteed Services, Shared-Memory Abstraction, and Flexible Network Programming," *DATE*, Feb. 2004.
- [13] E. Rijpkema, et al., "Trade-Offs in the Design of a Router with Both Guaranteed and Best-Effort Services for Networks on Chip," in *Computers and Digital Techniques*, Sep. 2003.
- [14] E. S. Reddy, et al., "Online Detection and Diagnosis of Multiple Configuration Upsets in LUTs of SRAM-based FPGAs," *IPDPS*, 2005.
- [15] N. Shnidman, et al., "On-line Fault Detection for Bus-Based Field Programmable Gate Arrays," in *TVLSI*, Vol. 6, 1998.
- [16] S. Srinivasan, et al., "Toward Increasing FPGA Lifetime," in *Transactions on Dependable and Secure Computing*, Vol. 5, Apr. 2008.
- [17] "Standard Test Access Port and Boundary-Scan Architecture, IEEE Standard, P1149, 1990."
- [18] C. Stroud, et al., "On-Line BIST and Diagnosis of FPGA Interconnect Using Roving STARS," in *Proc. On-Line Testing Workshop*, 2001.
- [19] S. Toutouchi, et al., "FPGA Test and Coverage," in *ITC*, 2002.
- [20] V. Verma, et al., "Efficient on-line testing of FPGAs with provable diagnosabilities," in *DAC*, Jun. 2004.
- [21] M.A. Wahlah, et al., "Modeling reconfiguration in a FPGA with a hardwired network on chip," in *RAW*, May. 2009.
- [22] Xilinx Inc., "Virtex-4 Data Sheets," 2005.
- [23] Xilinx Inc., "Virtex-6 Data Sheets," 2009.
- [24] Xilinx Inc., "Virtex-4 Configuration Guide."