2020 IEEE 38th VLSI Test Symposium (VTS)

ATTEST: Application-Agnostic Testing of a Novel Transistor-Level Programmable Fabric

Mustafa Munawar Shihab, Bharath Ramanidharan, Suraag Sunil Tellakula, Gaurav Rajavendra Reddy, Jingxian Tian, Carl Sechen and Yiorgos Makris
Department of Electrical and Computer Engineering, The University of Texas at Dallas, Richardson, Texas, USA
{mustafa.shihab, bharath.ramanidharan, suraag.tellakula, gaurav.reddy, jingxian.tian, carl.sechen, yiorgos.makris}@utdallas.edu

***Abstract*—A recently introduced TRAnsistor-level Programmable fabric (TRAP) has demonstrated great promise towards seamless unification of high-density reconfigurable logic with Application-Specific Integrated Circuits (ASICs). However, practical deployment of TRAP relies on the development of a comprehensive mechanism for detecting manufacturing defects. Unfortunately, the state-of-the-art test schemes are developed either for ASICs or for Field-Programmable Gate Arrays (FPGAs) and do not support this new transistor-level architecture. To address this limitation, we present a novel application-agnostic test methodology specifically tailored to the TRAP fabric. We first introduce a multi-phase, cascadable scheme to efficiently test the programmable transistors in TRAP's Logic Elements (LEs). Then, we define the required test patterns for verifying the correct functionality of the built-in D flip-flop, full-adder, and multiplexer of each LE. Next, we present a systematic approach for testing the interconnect network. Lastly, we discuss the limitations in testing the memory cells used for storing the TRAP programming bits and we propose design modifications for improving test coverage.**

I. Introduction

A. Motivation

Reconfigurable computing has made remarkable progress in recent years and has become a key technology for a broad gamut of applications, ranging from design emulations to deep-learning neural network acceleration [1]. While the conventional Look-Up Table- (LUT-) based FPGAs continue to lead this domain, several studies have investigated alternative architectures based on fine-granular programmability [2]–[4]. Among them, a recent work [5] introduced a field-programmable transistor array design called TRAP, which demonstrated particularly compelling features, such as (i) better area utilization than LUT-based FPGAs in the same technology node, and (ii) CAD support through an ASIC-compatible design flow which enables seamless integration of the TRAP programmable fabric with custom ASIC designs. Consequently, TRAP is considered as a very promising technology for various novel applications both in reconfigurable computing (i.e., in-field hardware updates) and in hardware security (i.e., design obfuscation) [6], [7]. However, for TRAP to be successfully used in actual products, the capability to thoroughly examine the fabric for manufacturing defects that may jeopardize its correct functionality after fabrication is indispensable. Therefore, development of a robust and efficient test method which can ensure reliable and fault-free operation of a deployed TRAP fabric is paramount.

B. State-of-the-art and Challenges

For the most part, conventional integrated circuit (IC) testing methodologies have been developed for ASICs. While there exist different approaches to it, by definition, fault detection in ASICs is always *application-specific*. That is, the testing schemes probe ICs based on the function(s) they implement. On the other hand, testing of reconfigurable devices is a relatively small but expanding research area that predominantly focuses on FPGAs. Unlike its ASIC counterpart, FPGA testing can be either *application-agnostic* [8]–[10] or *application-specific* [11], [12]. Typically, the manufacturer tests the resources of an FPGA exhaustively to ensure correct functionality of any design that can be mapped onto it, while an end-user only tests a device for his/her target application. Prior studies have presented various schemes for efficiently testing reconfigurable devices, often leveraging ASIC testing techniques such as built-in self-test (BIST) [9] and design for test (DFT) [12]. However, such schemes are exclusively designed for and applicable to conventional LUT-based FPGAs.

TRAP's distinct architecture and intended applications introduce a unique set of challenges in testing the fabric for manufacturing faults. Specifically, TRAP is proposed to be integrated on-die with custom designs. However, unlike the custom parts of a design, the TRAP portion can be reconfigured for different applications and, therefore, demands application-agnostic testing. Furthermore, while TRAP and conventional FPGAs both consist of logic, interconnect, and memory components, their architectures bear fundamental differences: (i) FPGAs implement logic with LUTs that are tested at the functional level for stuck-at faults, whereas TRAP's logic elements (LEs) need to be tested for transistor-level stuck-on/off faults. (ii) The FPGA interconnect network routes signals through many-to-many switch matrices. In contrast, TRAP leverages a pass transistor-based switch design that enables routing via one-to-one track links. (iii) While FPGAs store the programming bits in standard SRAM cells, TRAP leverages a custom memory cell design that consists of a latch driven by two transmission-gate switches. As a result of these (and other) differences, there exists no straightforward mechanism to apply existing IC test methods to TRAP.

C. Contributions

To alleviate incompatibility with existing test methods and to enable TRAP's promising capabilities, herein we present ATTEST, an **A**pplication-agnos**T**ic **Tes**t method for **T**RAP. ATTEST is a robust fault detection method, carefully designed to address the unique test requirements of this new fabric. Specifically, the **key** contributions of this work are as follows:

- **Specifying Test Requirements:** We analyze TRAP's architecture from a test perspective and reveal its component-specific test requirements and challenges.
- **Logic Element Testing:** We propose a multi-phase, cascadable test scheme for the programmable transistors in TRAP. This scheme can be set up for either a fast, *detection-only* scan of the entire fabric or for a fine-granular test to locate the source of the fault(s). We also define test patterns for the built-in cells in each LE.

978-1-7281-5359-9/20/$31.00 ©2020 IEEE

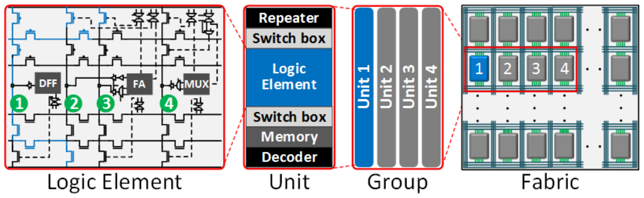


Fig. 1: Hierarchical organization of the TRAP fabric.

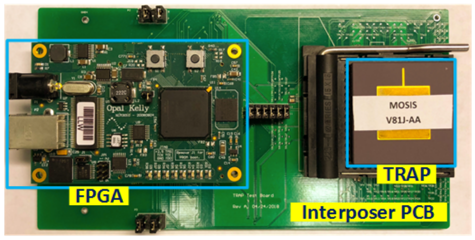


Fig. 2: Silicone prototype and hardware testbed for TRAP [6].

- **Interconnect Testing:** We present a route-based fault-detection scheme that tests the vertical and horizontal wire tracks via pad-to-pad bouncing of signals across the fabric. Our scheme also activates and tests the pass transistor switches and the bi-directional signal repeaters.
- **Memory Testing:** We discuss the limitations of the current design with regards to testing the memory cells that store the programming bits of TRAP and we propose design modifications that can maximize test coverage.

Our SPICE simulation results confirm a 100% fault coverage for the TRAP LEs with our proposed scheme. Furthermore, our parameterized Python-based tool can successfully generate a full set of signal routes to exhaustively test the interconnect network of a TRAP fabric of any size.

II. Transistor-Level Programming Fabric (TRAP)

TRAP is a field programmable transistor array that supports post-fabrication instantiation of arbitrary logic functions [5]. The target function, which can be either combinational or sequential, are instantiated by connecting cell-based primitives that are programmed at the transistor-level.

A. Architecture

The fabric is structured as a hierarchically arranged CMOS sea-of-transistors, as portrayed in Figure 1.

Logic element: The logic element (LE) is the core component for implementing a function on TRAP. As shown in the figure, each LE contains four *columns* of transistors. Each column consists of eight transistors, wherein four transistors have a vertical orientation and the other four are aligned horizontally. The transistors in an LE can be stitched together into gates or state-holding components. In order to improve TRAP's performance and area efficiency, each LE also includes a built-in flip-flop (DFF), a full adder (FA) and a multiplexer (MUX). The DFF is connected to column 1, while the MUX to column 4. The FA is connected to both columns 2 and 3.

Unit: Each logic element is combined with an address decoder, memory blocks, switch boxes and a repeater to form a unit. The memory blocks store the programming bits, while the switch boxes are used to deliver the bits to the LE. The repeater is used for boosting signals traveling across multiple units.

Group: At the top of the hierarchy, four neighboring units together form a functional block called a Group. The groups communicate between them and with the fabric's input/output (I/O) pads through programmable interconnect (not shown in the figure). We note that the role of a group is to assist in address mapping and storing of the programming bits. Otherwise, from a hardware/CAD perspective, TRAP can be defined as a connected array of units, as depicted in the figure.

TABLE I: TRAP components and corresponding fault models.

Category	Component	Fault/Test Type
Logic Element	Programmable Transistor	Stuck-on (SON) / Stuck-off (SOFF)
	Built-In Cell (DFF, FA, MUX)	Stuck-at-0/1 (SA0 / SA1)
Interconnect Network	Metal Track	Stuck-at-0/1 (SA0 / SA1), Bridging Fault
	Switch (Pass Transistor), Repeater	Stuck-on (SON) / Stuck-off (SOFF)
Memory	Flipflop	Stuck-at-0/1 (SA0 / SA1)
	Memory Cell	Memory Test

B. Programming Mechanism

In order to ensure compatibility and streamlined integration with the ASIC design-flow, TRAP utilizes a library of cells with fixed height but variable widths. Consequently, different logic gates require a varying number of (transistor) columns, can start from any column, and can extend across multiple LEs. As an example, the programming of a NAND3 gate in a TRAP LE is shown in Figure 1, where a suitable bitstream switches on the required transistors (highlighted in blue) and turns off the remaining ones. In addition, the programming bitstream needs to activate the required switches in the corresponding switch boxes. A prototype TRAP chip was implemented in GlobalFoundries 65nm technology along with an FPGA-based testbed, as shown in Figure 2, in order to verify and evaluate its design [6].

III. Overview of the Proposed Methodology

In order to develop a robust testing methodology for TRAP, it is essential to understand the impact of potential manufacturing defects on its operation. The TRAP fabric consists of three main components: (i) programmable logic elements, (ii) programmable interconnect network, and (iii) memory cells for storing programming bits. In the following, we consider these components and the corresponding type of (potential) manufacturing faults they need to be tested for. A summary of our findings is shown in Table I.

Logic Element (LE): Instead of using a functional unit, such as an LUT, TRAP implements logic functions by directly stitching together the programmable transistors in its LEs. Consequently, we need to test the thirty-two transistors in each LE for transistor-level stuck-on (SON) and stuck-off (SOFF) faults. On the other hand, the three built-in cells in each LE implement either a combinational (FA and MUX) or a sequential (DFF) function. Therefore, we need to test them for functional-level stuck-at 0 (SA0) and stuck-at 1 (SA1) faults.

Interconnect Network: TRAP leverages an island-style interconnect model where metal tracks run in the vertical and horizontal directions, connecting LEs and I/O pads as required by the mapped design. In addition, TRAP employs pass transistors as switches that, when turned on, enable a signal to travel from a vertical track to a horizontal track or vice versa. Finally, TRAP has bi-directional repeaters placed at regular intervals to avoid signal attenuation. To ensure fault-free operation, we need to test the tracks for stuck-at (SA0/SA1) and bridging faults. In addition, we must test the switches and the repeaters for stuck-on/off (SON/SOFF) faults.

Memory Cells: TRAP's memory components can be categorized into two groups: (i) I/O-associated memory that stores

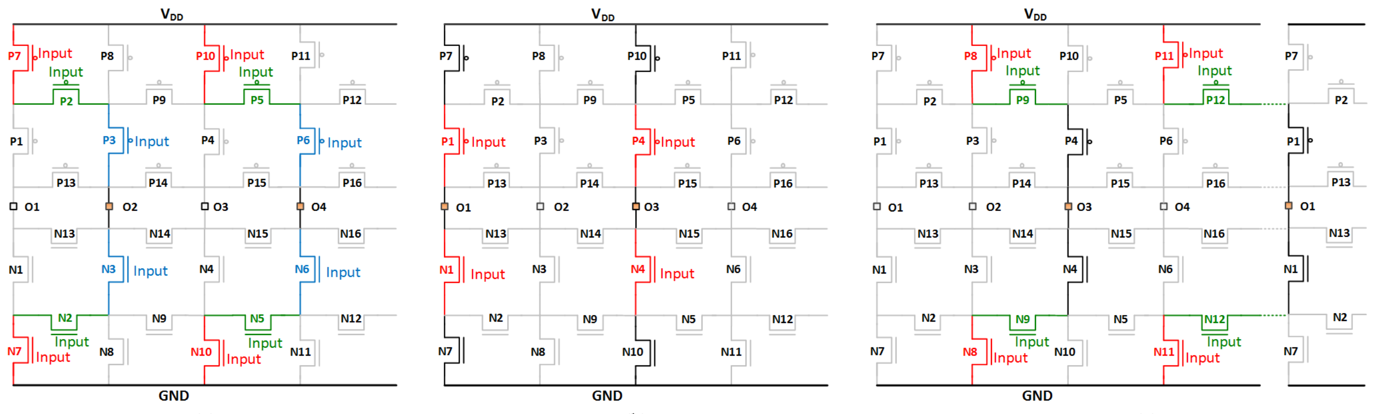


Fig. 3: Proposed inverter-based testing (IBT) of a TRAP LE: (a) Phase 1, (b) Phase 2, and (c) Phase 3.

TABLE II: Test configurations used in the IBT scheme.[1]

Phase 1						
	Test Vector	**Input Value**	**Transistors Under Test**	**Transistors on Path (Pgmd. as wire/short)**	**Output O2**	**Fault being tested**
Config. A	(i)	0	P7, N7	P2, P3, N2, N3	1	P7→SOFF & N7→SON
	(ii)	1			0	P7→SON & N7→SOFF
Config. B	(i)	0	P2, N2	P7, P3, N7, N3	1	P2→SOFF & N2→SON
	(ii)	1			0	P2→SON & N2→SOFF
Config. C	(i)	0	P3, N3	P7, P2, N7, N2	1	P3→SOFF & N3→SON
	(ii)	1			0	P3→SON & N3→SOFF

Phase 2						
	Test Vector	**Input Value**	**Transistors Under Test**	**Transistors on Path (Pgmd. as wire/short)**	**Output O1**	**Fault being tested**
Config. A	(i)	0	P1, N1	P7, N7	1	P1→SOFF & N1→SON
	(ii)	1			0	P1→SON & N1→SOFF

Phase 3						
	Test Vector	**Input Value**	**Transistors Under Test**	**Transistors on Path (Pgmd. as wire/short)**	**Output O3**	**Fault being tested**
Config. A	(i)	0	P8, N8	P9, P4, N9, N4	1	P8→SOFF & N8→SON
	(ii)	1			0	P8→SON & N8→SOFF
Config. B	(i)	0	P9, N9	P8, P4, N8, N4	1	P9→SOFF & N9→SON
	(ii)	1			0	P9→SON & N9→SOFF

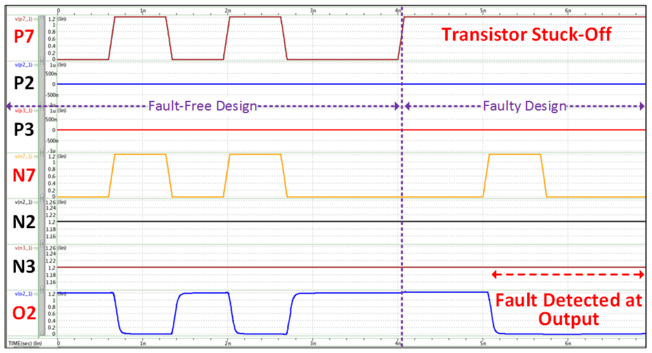


Fig. 4: Sample SPICE simulation of IBT (Phase 1 – Config. A) confirms detection of `P7`→`SOFF` fault at output `O2`.

I/O pad programming bits, and (ii) LE-associated memory which stores programming bits for all the LEs. The memory cells need to be tested with standard test techniques such as March-based algorithms [13].

IV. Testing the Logic Elements

As mentioned in Section III, in order to ensure fault-free operation of an LE we need to test the 32 programmable transistors, as well as the built-in cells it contains, namely, a D-flipflop (DFF), a full-adder (FA), and a multiplexer (MUX). In this section, we first introduce a multi-phase scheme for identifying the faulty transistors in an LE. Then, we discuss how the proposed scheme can be cascaded to achieve a fast, parallelized sweep of the fabric for detecting fault(s). Finally, we present test patterns for the three built-in cells in an LE.

A. Inverter-based Testing (IBT) of Programmable Transistors

In the proposed scheme, we stitch together a particular set of transistors to form an inverter logic gate. The inverter can span over either one or two columns of an LE. In each test *configuration*, the pair of *transistors under test* receives the input test vectors, whereas the remaining *transistors on the path* to the output are programmed to act as wire (i.e., shorted). Since each LE has four columns of transistors, we maximize test efficiency by duplicating the aforementioned inverter formation setup on the remaining column(s) of the LE. Furthermore, we group multiple configurations to form a *phase*. Within each phase, the configurations represent which pair of transistors in the inverter is being tested, where the different phases represent a pair of inverters programmed in the LE. As shown in Figure 3, our IBT scheme has three phases and each phase contains up to three configurations. In the figure, the transistors under test in each configuration are marked with different colors: Configuration A (red), Configuration B (green), and Configuration C (blue). The details for each configuration, i.e., *input value*, *transistor under test*, *transistors on path*, expected *output*, and *faults being tested* are summarized in Table II.

[1]For better readability, we have represented each phase and its corresponding configurations in terms of the *left* inverters in the LE. The test patterns will be the same for the transistors forming the *right* inverters.

As shown in Figure 3(a), *Phase 1* consists of three configurations. In *Configuration 1*, transistors `P7` and `N7` are tested and given inputs, while `P2`, `P3`, `N2` and `N3` are transistors on the path to the output (`O2`) and are programmed as wires. First, we apply 0 as input and observe the output to verify that `P7` and N7 are not stuck-off and stuck-on, respectively. We, then, apply 1 as input to test whether `P7` is stuck-on or `N7` is stuck-off. In *Configuration 2*, `P2` and `N2` receive input values and are tested for stuck-on and stuck-off faults, while `P7`, `N7` along with `P3` and `N3` are programmed as wires to form the inverter. Similarly, *Configuration 3* tests `P3` and `N4` for stuck-on/off faults, whereas `P7`, `P2`, `N7`, and `N2` are programmed as wires. In the exact same process, Configurations 1, 2, and 3 also test the transistors `P10`, `P5`, `P6`, `N10`, `N5`, and `N6`, i.e., the six transistors that form the *right* inverter on the third and fourth columns of the LE, as shown in Figure 3.

The inverter arrangements for *Phase 2* and *Phase 3* are shown in Figures 3(b) and (c), respectively. As summarized

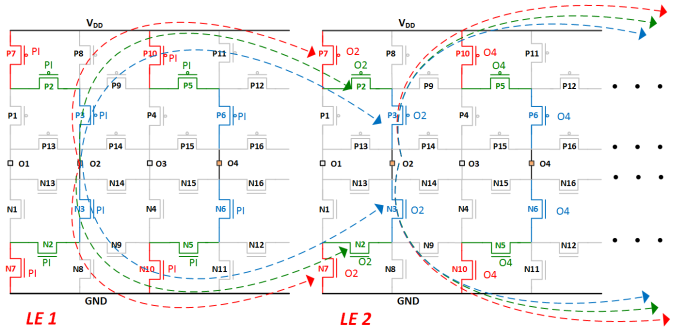


Fig. 5: Example of cascading the IBT scheme across LEs to reduce I/O overhead and enable parallelized fault detection.

in Table II, *Phase 2* involves a single configuration that tests four transistors: P1 and N1 through output O1, and P4 and N4 through output O3. On the other hand, *Phase 3* consists of two configurations that test a total of eight transistors: P8, P9, N8, and N9 via output O2, and P11, P12, N11, and N12 via output O1 of the adjacent LE. As the execution process for these two phases is very similar to the aforementioned process for *Phase 1* and due to space limitations, a more detailed explanation of their operation is omitted.

In conjunction, the three phases of the IBT scheme can test twenty-four transistors in a TRAP LE and precisely identify the location of a fault, if one exists. It is also worth noting that IBT considers a single-fault model. Figure 4 depicts a SPICE simulation result for IBT where the impact of a stuck-off fault in transistor P7 is propagated to the output O2.

B. Cascading IBT for Parallelized Fault Detection

In many cases, detecting the presence of a fault is the critical goal, while identifying its location is optional or secondary. On the other hand, while IBT can accurately detect and locate faults in the TRAP LE, for all but one configurations, it requires two dedicated output pads to read out the test results from each LE. Consequently, if there are *n* LEs in a TRAP fabric, the number of I/O pads required to implement IBT for all the LEs in parallel would be *2n+1*. This is because each configuration requires one input value and all the LEs tested in parallel can share that input. For larger fabric sizes or high-density applications, the availability of sufficient output pads maybe become a bottleneck. Nevertheless, the architecture of our IBT scheme can provide an elegant solution for such cases.

Leveraging "inverters" for fault-detection enables us to cascade the transistor configurations in each IBT phase across multiple LEs and form a chain that can then propagate the impact of a fault to the final output. Specifically, the *transistors under test* in the first LE of the chain receive the test input, the identical transistors in the second LE receive the output of the first LE as their input, the transistors in the third LE receive the output from the second LE and so forth. Also, the tests being purely combinational, no timing constraint needs to be considered. Figure 5 demonstrates an example of the cascaded IBT for *Phase 1*, where the first LE (LE1) receives the primary inputs (PIs) for the three configurations (red, green, and blue). The test output(s) from LE1 is then forwarded to the second LE (LE2) as input(s).

The cascaded IBT enables exploration of the trade-off between fast, parallelized fault detection with low I/O overhead and the ability to zero in on the location of a fault. In other words, a longer IBT chain represents faster testing with fewer I/O pads, while a shorter chain ensures better and faster fault localization (with subsequent single LE IBT testing).

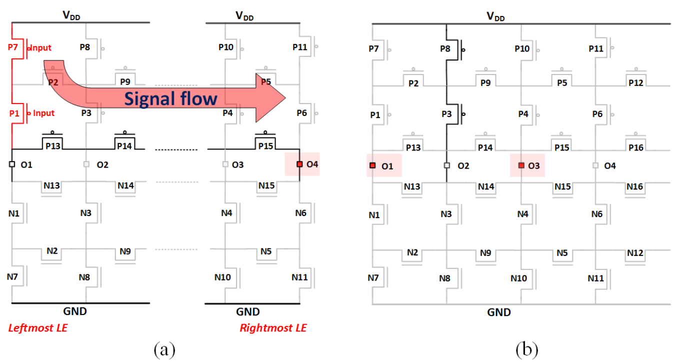


Fig. 6: Testing the center-row horizontal transistors (CRHTs) for (a) Stuck-off (SOFF), and (b) Stuck-on (SON) faults.

C. Testing the Center-Row Horizontal Transistors (CRHT)

The proposed IBT scheme cannot test the eight horizontally-oriented transistors P13–P16 and N13–N16 located in the center two rows of the TRAP LEs. This is due to an architectural limitation in TRAP that prohibits us from controlling the aforementioned transistors with inputs from external I/O. Specifically, these transistors can be switched on/off only by programming through memory. To address this limitation, we propose two additional test configuration pairs that can successfully test the CRHTs for stuck-on/off faults.

1) Stuck-Off Test: Figure 6(a) shows the first configuration for this test, where we cascade the LEs while programming them to switch on only P13–P16 in each of them. We then apply a 0 value input to P7 and P1 and sample the output from O4 of the last LE. A high (1) value of the sampled output confirms absence of any stuck-off (SOFF) faults in any of the P13–P16), as the only path for signal flow is through them. Similarly, in the second configuration P13–P16 are switched off and N13–N16 are switched on and tested for SOFF faults.

2) Stuck-On Test: In the first configuration of this test, we switch off all the transistors in the LE except for P8 and P3, as shown in Figure 6(b). The value of the outputs O1 and O2 are tested, which should be low (0) unless at least one of the four adjacent CRHTs P13, N13, P14, and N14 has a stuck-on (SON) fault. In the second configuration, we test for P15, N15, P16, and N16 by switching on P11 and P6 and observing O3 and O4 (adjacent LE). However, this test must be done for individual LEs to accurately detect SON faults.

D. Testing the Built-In Cells

The output pins in a TRAP LE are shared between the built-in cells and the programmable transistors. Specifically, the DFF is connected to O1, the FA is connected to O2 and O3, and the MUX is connected to O4. To avoid potential "race" conditions, TRAP implements a switch between the built-in cells and their corresponding output pins. Consequently, testing of a built-in cell also involves testing its switch via the enable (En) signal, as shown in Table III. The signals that need to be tested for each built-in cell are discussed below.

D Flip-Flop (DFF): Apart from the clock signal (tested separately), the built-in DFF has three inputs (data (D), enable (En), and Reset), and one output (Q). To test the DFF, we

TABLE III: Test patterns for the built-in cells in TRAP LE.

D Flip-Flop (DFF)						
	En	**Reset**	**Clock**	**D[t]**	**Q (O2) [t+1]**	**Test Description**
(i)	1	0	$\uparrow \gg 1$	1	1	Initialization vector
(ii)	1	0	$\uparrow \gg 1$	0	0	D: SA1, Q: SA1
(iii)	1	0	$\uparrow \gg 1$	1	1	D: SA0, Q: SA0
(iv)	1	1	$\uparrow \gg 1$	1	0	Reset signal works
(v)	0	0	$\uparrow \gg 1$	1	0	Enable signal works

Full-Adder (FA)							
	En	**A**	**B**	**C**	**Sum (O2)**	**Carry (O3)**	**Test Description**
(i)	1	0	0	0	0	0	Initialization vector
(ii)	1	1	0	0	1	0	A: SA0, Sum: SA0
(iii)	1	1	1	0	0	1	B: SA0, Sum: SA1, Carry: SA0
(iv)	1	1	1	1	1	1	C:SA1
(v)	1	0	1	1	0	1	A:SA1
(vi)	1	0	0	1	1	0	B:SA1, Carry: SA1
(vii)	1	0	0	0	0	0	C:SA1
(vii)	0	1	1	1	0	0	Enable signal works

Multiplexer (MUX)						
	En	**Sel (1: A, 0: B)**	**A**	**B**	**Out (O4)**	**Test Description**
(i)	1	1	0	0	0	Initialization vector
(ii)	1	1	1	0	1	A: SA0, Out: SA0
(iii)	1	1	0	1	0	A: SA1, Out: SA1
(iv)	1	0	0	1	1	Sel: SA1
(v)	1	0	0	0	0	B: SA1
(vi)	1	0	0	1	1	B: SA0
(vii)	1	1	0	1	0	Sel: SA0
(vii)	0	1	1	0	0	Enable signal works

examine its input and output signals for stuck-at faults and check whether the control signals perform as expected. Table III summarizes the test vectors required for the DFF.

Full Adder (FA): The built-in FA has a total of four inputs (enable (`EN`) and three data inputs `A`, `B`, and `C`) and two outputs (`Sum` and `Carry`). As shown in Table III, we apply eight vectors to test the signals for potential logical (stuck-at) faults.

Multiplexer (MUX): The built-in MUX has two control inputs (enable `En` and select (`Sel`)), two data inputs (`A` and `B`), and a data output `Out`. Table III summarizes the vectors used to test these signals and, thereby, ensure correct MUX operation.

V. Testing the Interconnect Network

A high-level layout of TRAP's island-style interconnect network is depicted in Figure 7, where each I/O pad is connected to a multiplexer, which, in turn, can connect to specific tracks of a switch-box (SB). A switch-box consists of vertical (VT) and horizontal (HT) tracks that surround an LE. All the switch-boxes are cascaded together, vertically and horizontally, with a bi-directional repeater placed between each pair to ensure signal integrity. Each LE has upper and lower connection-boxes (CB) that connect the LE to the horizontal tracks of the surrounding switch-box. The interconnect network of TRAP has limited flexibility, that is, each vertical track can only connect to a specific set of horizontal tracks and vice versa. Finally, specific vertical and horizontal tracks can be connected via pass transistor switches.

A. Testing Requirements for the Interconnect Network

In order to ensure a fault-free interconnect network, the testing scheme must execute the following tasks:

1. I/O Pad MUXs: The MUXs need to be tested for confirming that they can connect to the specified switch-box tracks.

2(a). Switch-box – Tracks: The horizontal and vertical tracks need to be tested for *open*, *short*, and *bridging* faults. In addition, the repeaters need to be tested for stuck-open faults.

2(b). Switch-box – Switches: The pass transistor switches need to be tested for stuck-on/off faults to ensure that a vertical track can connect to *and only to* the intended horizontal tracks.

3. Connection-box: The vertical tracks and the switches connecting those tracks to the surrounding switch-box need to be tested for the faults mentioned in Tasks 2(a) and (b).

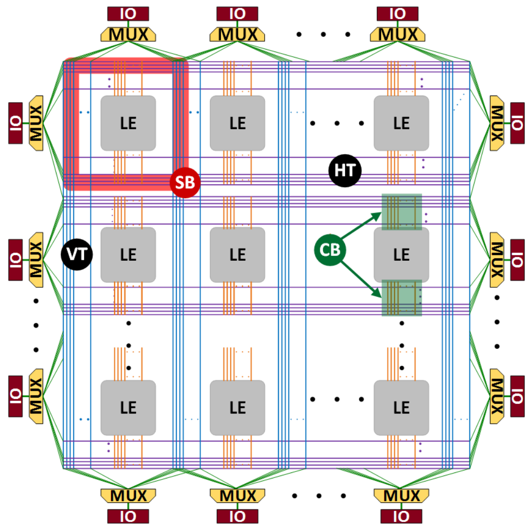
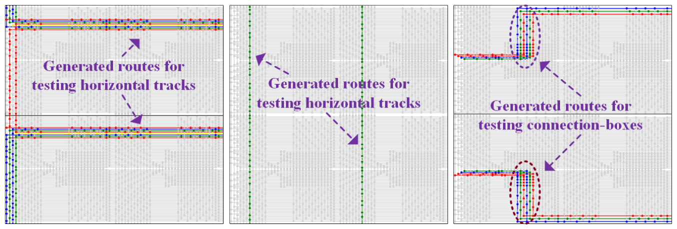


Fig. 7: The interconnect network of TRAP.


Generated routes for testing horizontal tracks
Generated routes for testing horizontal tracks
Generated routes for testing connection-boxes
Fig. 8: Examples of automatically generated routing patterns for testing a 2X2 TRAP fabric interconnect network.

B. Routing-Based Fault Detection

We propose a fault detection methodology for the TRAP interconnect network that leverages pad-to-pad signal propagation across the fabric. Specifically, our scheme applies the test vector through an I/O pad, guides the signal across the fabric through a specific pre-determined *route*, and then takes the output through another I/O pad for verification. The "routes under test" are carefully designed to cover all the interconnect components that need to be tested, based on the four aforementioned tasks/requirements. Therefore, testing the full set of routes ensures that the network is tested exhaustively. Once a route is identified, a corresponding test pattern is generated based on the components connected by the path and all potential faults that may affect them. Our method leverages existing fault-detection schemes for generating test patterns for the paths and can simultaneously test multiple (mutually exclusive) paths through different pairs of I/O pads.

Automating the Route Generation Process: For an efficient and accurate implementation of the proposed scheme, the path generation process needs to be automated and scalable. To this end, as a proof of concept, we have implemented a Python-based tool that captures the TRAP architecture as a graph in which the I/O pads are defined as start/end nodes, while the tracks are the edges. The switches and the repeaters together represent the intermediate nodes but are marked differently to reflect their purpose. Out tool leverages a parameterized model of TRAP and can successfully generate the smallest set of *paths* required to exhaustively test the interconnect network of a given TRAP fabric. Figure 8 shows various examples of test paths generated by the tool for a 2X2 TRAP fabric.

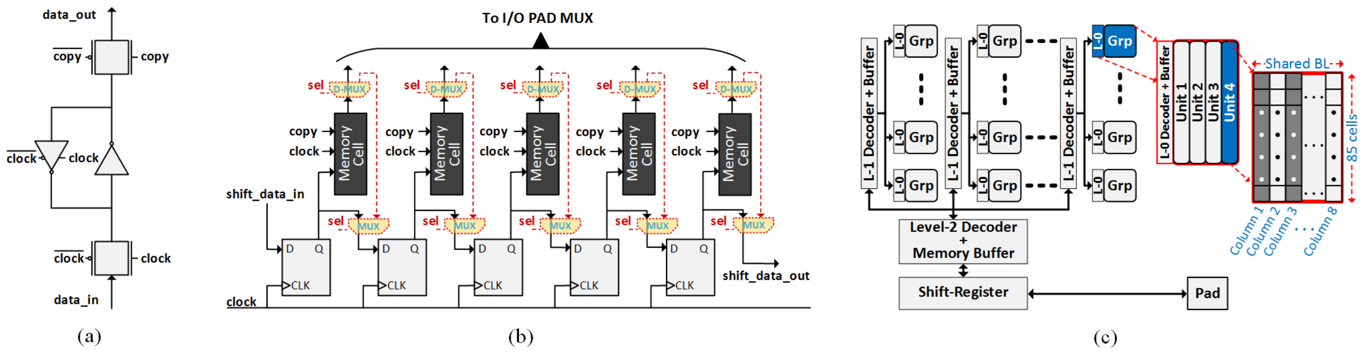


Fig. 9: Storing programming bits on TRAP – (a) single memory cell, (b) I/O-associated, and (c) LE-associated memory array.

VI. Testing the Memory Cells

The TRAP memory cell consists of a latch and two transmission gate-based switches to control data movement, as shown in Figure 9(a). The `clock` signal is used to load data into the latch, while the `copy` signal controls transferring of the stored bit to its designated location on the fabric. TRAP uses two separate arrays of such cells to store programming bits, one for the I/O pads and one for the LEs. In the following, we discuss current design constraints that prohibit testing the TRAP memory arrays with conventional methods and we propose design modifications to enable superior test coverage.

A. I/O Associated Memory (IAM)

Figure 9(b) portrays the architecture of the IAM array which leverages a DFF chain to load the data into the memory cells (MC). It is rather straightforward to test the DFFs by *scanning* test data in and out through the chain. Unfortunately, the current design of IAM has no mechanism to read out the content of the MCs, leaving no viable means for testing them directly. To resolve this limitation, we propose to incorporate a multiplexer (MUX) and a demultiplexer (DMUX) for each MC. As shown in the figure, the MUX and the DMUX can allow us to read data out of the MCs and load them back into the DFF chain, respectively. If implemented, the proposed modification can create an alternate *scan path* that can be used to test the MCs in the IAM with a robust memory testing scheme such as a March test [13]. A March test consists of a finite sequence of test primitives (also known as *elements*) applied to the memory cells to detect a range of faults such as stuck-at, stuck-on/off, transition, coupling, linked, and address decode faults.

B. LE Associated Memory (LAM)

The LAM leverages a three-tier address decoder and memory buffer pipeline for loading the programming bits into the MCs, as shown in Figure 9(c). Specifically, a 96-bit data is sequentially fed in through an I/O pad, out of which an 11-bit address is used by the decoders and an 85-bit payload is loaded into a memory column of a unit (LE). The memory array for each unit consists of 8 such columns of cells. While the LAM does support data read-out, it is the 85-bit wide granularity of the read-write operations that prevents it from leveraging conventional test methods. This is because memory testing techniques, such as March tests, are predominantly designed for bit-wise memory operation. Indeed, modifying the address decoder design for cell-specific access and inserting a designated scan path connecting each cell in the array can enable the most accurate testing of the LAM. However, considering the overall size of the LAM, implementing such hardware modifications can incur overwhelming, if not prohibitive, area overhead. As a more efficient and feasible solution, we propose to test the LAM with a unique, albeit lesser-known, variation of March test for word-oriented memory (March-WOM) [14]. Specifically, applying March-WOM on TRAP's LAM, with the word-size set to 85 bits (i.e., considering each memory column as a single word) can enable us to test the array for most memory-related errors in the form of inter-word and intra-word faults. Finally, we recommend augmenting the March-WOM test to include coverage for detecting data retention faults (DRFs) in TRAP memory cells.

VII. Conclusion

We developed a novel application-agnostic testing method for the TRAP fabric. This method consists of a multi-phase, cascadable scheme to efficiently test the programmable transistors in the LEs and includes carefully crafted test patterns for ensuring that the built-in DFF, full-adder, and multiplexer of each LE operate correctly. It also comprises a systematic approach for testing TRAP's interconnect network. To complete the TRAP fabric test method, we also proposed design modifications to enable direct testing of the memory cells that store the programming bits of the TRAP fabric, which is impeded by current design limitations.

References

[1] C. Wang, L. Gong, Q. Yu *et al.*, "DLAU: A Scalable Deep-Learning Accelerator Unit on FPGA," *IEEE TCAD*, vol. 36–3, pp. 513–517, 2016.
[2] P. Layzell, "A New Research Tool for Intrinsic Hardware Evolution," in *International Conference on Evolvable Systems*, 1998.
[3] A. Stoica, "Towards Evolvable Hardware Chips: Experiments with a Programmable Transistor Array," in *MicroNeuro*, 1999.
[4] J. Langeheine, K. Meier *et al.*, "Intrinsic Evolution of Analog Electronic Circuits Using a CMOS FPTA Chip," in *EUROGEN*, 2003.
[5] J. Tian, G. Reddy *et al.*, "A Field Programmable Transistor Array Feat. Single-Cycle Partial/Full Dynamic Reconfiguration," in *DATE*, 2017.
[6] M. M. Shihab, J. Tian *et al.*, "Design Obfuscation through Selective Post-Fabrication Transistor-Level Programming," in *DATE*, 2019.
[7] B. Hu, J. Tian, M. Shihab *et al.*, "Functional Obfuscation of Hardware Accelerators through Selective Partial Design Extraction onto an Embedded FPGA," in *GLSVLSI*, 2019.
[8] W. K. Huang, F. J. Meyer, X.-T. Chen *et al.*, "Testing configurable lut-based fpga's," *IEEE TVLSI*, vol. 6, no. 2, pp. 276–283, 1998.
[9] C. Stroud, S. Wijesuriya, C. Hamilton *et al.*, "Built-In Self-Test of FPGA Interconnect," in *ITC*, 1998.
[10] M. B. Tahoori and S. Mitra, "Automatic Configuration Generation for FPGA Interconnect Testing," in *VTS*, 2003.
[11] M. Tahoori, "Application-dependent testing of fpgas," *IEEE TVLSI*, vol. 14, no. 9, pp. 1024–1033, 2006.
[12] M. Renovell, P. Faure, J. M. Portal, J. Figueras, and Y. Zorian, "Is-fpga: a new symmetric fpga architecture with implicit scan," in *ITC*, 2001.
[13] A. J. Van De Goor, G. N. Gaydadjiev, V. Mikitjuk *et al.*, "March LR: A Test for Realistic Linked Faults," in *VTS*, 1996.
[14] A. J. Van de Goor, Tlili *et al.*, "March Tests for Word-Oriented Memories," in *DATE*, 1998.