Testing a Transistor-Level Programmable Fabric: Challenges and Solutions

Apurva Jain, Thomas Broadfoot, Carl Sechen, Yiorgos Makris
Electrical and Computer Engineering Dept., The University of Texas at Dallas, Richardson, TX 75080, USA
{apurva.jain, thomas.broadfoot, carl.sechen, yiorgos.makris}@utdallas.edu

2024 IEEE 42nd VLSI Test Symposium (VTS) | 979-8-3503-6378-4/24/$31.00 ©2024 IEEE | DOI: 10.1109/VTS60656.2024.10538901

***Abstract*—Test vector generation for a TRAnsistor-level Programmable (TRAP) fabric faces a number of feasibility and efficiency challenges. The former are caused by (i) the use of bi-directional pass transistors, which are beyond the capabilities of commercial Automatic Test Pattern Generation (ATPG) tools, and (ii) the design specifics of TRAP, which result in certain stuck-at faults not being logically testable and calling for a quiescent current-based test solution instead. The latter are caused by the fact that ATPG tools are oblivious to (i) the difference between programming bits and regular inputs, which results in lengthy test application times, and (ii) the role that different modules in the architecture of TRAP play in establishing logic circuits, which results in lengthy unguided exploration of a very large functional space to establish appropriate vector justification and response propagation paths. To address these challenges, we explore an array of solutions including (i) employing TRAP instances where bi-directional transistors are replaced by uni-directional ones, (ii) generating custom IDDQ tests, (iii) expressing test application time as the optimization objective of an Integer Linear Program (ILP) formulation, and (iv) leveraging design knowledge, resulting in perfect stuck-at fault coverage of TRAP and an order-of-magnitude savings in test application time.**

I. Introduction

Integrated Circuit (IC) redaction [1], [2], [3], [4] has been recently proposed for protecting hardware Intellectual Property (IP) from untrusted entities of the globalized semiconductor manufacturing industry. In IC redaction, sensitive portions of a design are replaced by configurable hardware and only instantiated through post-manufacturing programming after the chips are received. To contain the overhead incurred by IC redaction solutions, most of which are based on eFPGAs, a TRAnsistor-level Programmable (TRAP) fabric was introduced in [5]. By pushing programmability to the fine-grain transistor level, TRAP not only drastically reduces overhead but also introduces more formidable challenges for both brute-force and intelligent search-based attacks to overcome. Adoption of this novel fabric, however, requires an RTL-to-GDSII tool flow for implementing hybrid ASIC/TRAP ICs [6], [7], [8], as well as a solution for testing such chips for manufacturing defects.

Existing fault modeling and ATPG solutions for transistor-level designs (*e.g.*, [9], [10], [11], [12], [13]) rely on certain predetermined signal direction to abstract functional logic. TRAP, however, is a programmable fabric that can only be abstracted *after* it has been programmed, yet must be tested before it is programmed. FPGA test solutions (*e.g.,* [14], [15], [16], [17]) are also not directly applicable, as they explore design-specific or application-specific configurations to reduce test application time. Compounding these limitations, in order to protect the programming bitstream, TRAP does not have a read-out circuit; hence, memory tests (*e.g.,* [18], [19]) cannot be directly applied to its SRAM. Also, while a method to test logic elements by configuring them into inverter chains was proposed in [20], it only covered a limited set of possible faults in the fabric. To address these limitations, in this paper, we discuss the challenges associated with testing the TRAP fabric [21] and we develop a comprehensive solution.

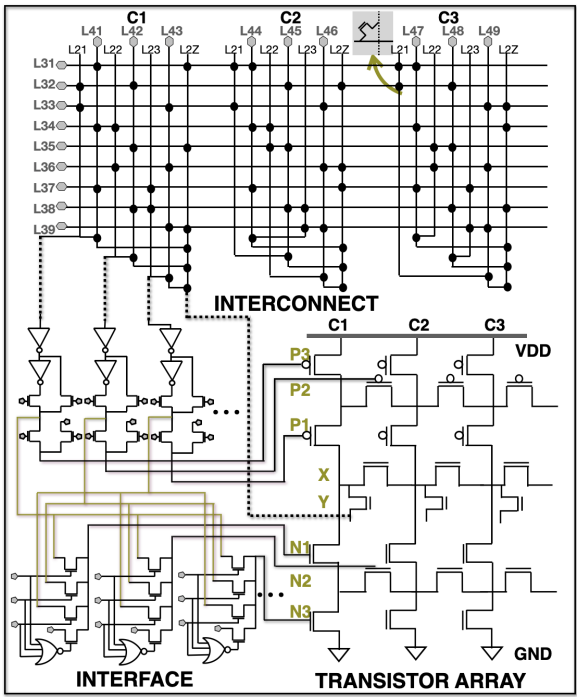

Fig. 1. Architecture of core components in a TRAP unit

II. TRAP Architecture

As shown in Fig. 1, the TRAP fabric architecture consists of three core components: (i) a programmable interconnect network, (ii) a transistor array (TA), and (iii) an interface that connects the interconnect network and the TA. These three components are hierarchically arranged into a *unit*, which is then replicated in a 2-dimensional array to produce a continuous fabric. An array of memory cells (*i.e.*, custom SRAM, not shown in the figure) is inter-weaved with the fabric to store the programming bits. TRAP has been designed, fabricated, and demonstrated in a 12nm FinFET technology.

*1) **Transistor Array**:* The TA of a unit consists of 24 transistors organized into three columns (`C1`, `C2`, and `C3`), as shown in the bottom right of Fig. 1. Each column contains three pull-up pMOS transistors (`P1` – `P3`) and three pull-down

979-8-3503-6378-4/24/$31.00 ©2024 IEEE

nMOS transistors (N1 – N3). Additionally, in each column (*e.g.*, C1), there is an nMOS transistor Y (controlled by a programming bit) that connects the output to the corresponding interconnect track in the same column (*e.g.*, C1L2Z). Finally, a connection between adjacent columns is established by a horizontally-oriented nMOS transistor X (controlled by a programming bit). In the TA, logic gates are implemented by programming individual transistors through programming bits.

2) ***Interface****:* The term 'interface' refers to the transistor-level multiplexer logic that links the interconnect to the column of transistors in the TA. The bottom left of Fig. 1 shows the interface structure for column C1. The pMOS transistors in that TA column can receive at their gate either (i) a signal, (ii) the complement of a signal, or (iii) the value of an SRAM bit. This is achieved by controlling the gate of transistors in the top portion of the interface using programming bits. Similarly, by controlling the bottom portion of the interface through programming bits, each of the three nMOS transistors in column C1 of the TA can receive one of the three signals from the pMOS interface in that column or can be programmed through a bit stored in SRAM. The interface structure is similar for the other two columns (C2 and C3) of the TA.

3) ***Interconnect Network****:* The top of Fig. 1 also shows the interconnect network of TRAP. Switches, shown as dots, are nMOS pass transistors that connect orthogonal metals (*e.g.*, L3 and L4) at multiple locations and also connect metal layers L2 and L4. The TRAP interconnect network consists of nine global horizontal (L3), nine global vertical (L4), and twelve local vertical (L2) tracks. The global tracks (*i.e.*, L3, L4) can connect to either a primary I/O or to the corresponding global tracks of a neighboring unit via pass transistors or bi-directional repeaters. When switches are turned ON, the two metal layers connected by the switch share the same signal.

III. HDL Description & Fault Model

Using existing ATPG tools for generating test vectors for TRAP requires an HDL representation of the fabric and an appropriate fault model. Recall that TRAP is a custom-designed transistor-level fabric, wherein transistors are programmed to form logic elements, and custom SRAM cells are used to hold the fabric configuration. Therefore, faithfully capturing the fabric structure in an HDL requires the use of transistor-level constructs (*e.g.*, pmos and nmos primitives in Verilog). In fact, since TRAP employs bi-directional pass-transistors, complex constructs (*e.g.*, tran, tranif0 and tranif1 primitives in Verilog) are required. A TRAP HDL model for simulation-based verification can be found in [8]. Herein, we use that same model to enable the use of ATPG.

Considering the transistor-level nature of TRAP, our fault model includes single stuck-at faults at each of the three terminals (*i.e.*, gate, drain, and source) of each transistor in the TA, interface, and interconnect. In addition, it includes single stuck-at faults in the SRAM cells, on the metal tracks of the interconnect network, and on the circuitry used for loading and shifting in the programming bits. These faults are equivalent to and manifest as a transistor gate input being stuck at a value.

In this work, we focus on understanding and overcoming the challenges of testing a single TRAP unit. Therefore, we consider as primary inputs the nine global vertical tracks (*i.e.*, L41 – L49) and as primary outputs the three global horizontal tracks that connect the interconnect of a unit to its adjacent unit. For ATPG purposes, the SRAM cells that are used for programming a unit are also considered primary inputs.

IV. ATPG Challenges for TRAP

A. Representing Bi-directional Transistors for ATPG

Challenge: While contemporary ATPG tools can handle transistor-level constructs, this ability is limited to designs employing strictly uni-directional transistors. TRAP, however, employs bi-directional transistors in the routing network of the interconnect and in the implementation of logic elements in the TA. While bi-directionality enables a large set of configurations that would otherwise not be possible, it prohibits the use of ATPG on TRAP. Converting the bi-directional transistor primitives to uni-directional ones can solve the problem. However, fixing the direction of these transistors restricts the functionality of the fabric since certain nodes may become redundant or unreachable. This, in turn, leads to untestable faults in the uni-directional version that would otherwise be testable in the original fabric version.

Solution: To overcome this problem, we curate multiple distinct versions of the TRAP HDL, each of which has only uni-directional transistors. The original version with bi-directional transistors and each of these multiple versions with uni-directional transistors have the exact same fault list, as the only difference is the primitive used for a transistor (*i.e.*, pmos or nmos, vs. tranif0 or tranif1). While applying ATPG on any one of these versions cannot generate a complete set of vectors to cover all faults, the union of vectors generated across all of these versions can achieve complete fault coverage. The direction chosen for replacing bi-directional transistors in these versions, however, has to be judiciously selected. For example, in order to test faults in the TA, the transistors should be oriented to pass the unit inputs (via the interface) to the gates of the TA, while the output of the TA (routed via the interconnect) should terminate at the unit outputs. Similarly, to target faults in the interconnect, the transistors should be oriented such that the signal from the unit inputs can be routed through as many transistors in the interconnect, and finally ends at the unit outputs. Collectively, as we discuss later in the experimental results section, a total of five versions suffice to achieve complete fault coverage on a TRAP unit. ATPG-generated vectors across these versions produce a complete *baseline test solution* for a TRAP unit.

B. Need for IDDQ Testing

Challenge: To achieve high density, the TRAP interface employs pass transistors to implement multiplexer logic. The resulting circuit, however, has paths that, under faulty conditions, would conduct both a logic value and its complement. This makes logic-based testing of such faults impossible, as a transistor driven by such a path behaves unpredictably.

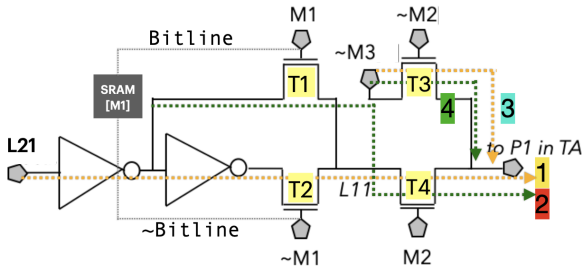

Fig. 2. pMOS interface showing different operational paths

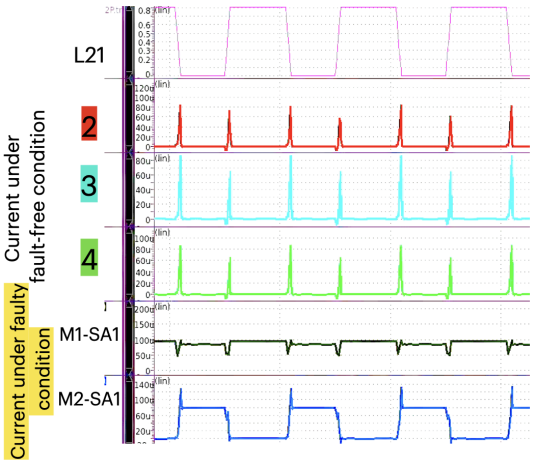

Fig. 3. Comparison of supply current under fault-free/faulty conditions

Solution: To address this challenge, we used a current-based solution akin to IDDQ testing. Fig. 2 shows the pMOS interface connecting the `L21` routing track to `pmos P1` via four possible paths passing a signal (`1`), the complement of the signal (`2`), SRAM logic '0' (`3`) and SRAM logic '1' (`4`). Fig. 3 shows the SPICE simulation under faulty and fault-free conditions. The first waveform is a pulsating `L21` signal. The next three waveforms show the supply current under fault-free conditions, when paths `2`, `3`, or `4` are individually conducting. The current averages near zero. A stuck-at-1 fault at the gate of transistor `T1` cannot be logically detected in a single stuck-at fault model. Under fault-free conditions, with `M1` set to 0, transistors `T1` and `T2` will be OFF and ON, respectively. The expected machine response will be `L21` following path `1`. However, under these faulty conditions (`M1` stuck-at-1), again with the `M1` value set to 0, both transistors `T1` and `T2` are ON. This leads to a faulty machine response where both signal `L21` and its complement `~L21` propagate along paths `1` and `2`, respectively. The logic value of this signal, when connected to the gate of a transistor in the TA, becomes unpredictable. However, as shown by the 5th waveform, the supply current under these circumstances rises to 100uA, irrespective of the `L21` value, which makes the `M1` stuck-at-1 fault testable through IDDQ testing. Also, the 6th waveform shows the current when `M2` is stuck-at-1. This causes two paths (`3,4` or `1,2`) to simultaneously write on `P1`. To detect this fault using IDDQ, we set those paths to conflicting values. The waveform is captured when `~M3` is set to 1, causing a high current to be detected when `L21` is low. The same conflict occurs with a stuck-at-0 fault at the gate of `T2` and `T4`.

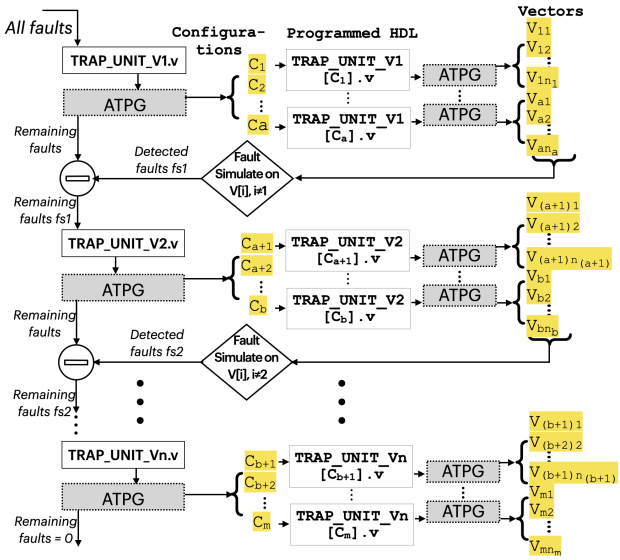

Fig. 4. Heuristic process for generating test vectors that maximize test coverage for each TRAP configuration required by the ATPG-based test set

C. ATPG is Agnostic to Test Application Time in TRAP

Challenge: Since TRAP is a programmable fabric, the application of test patterns involves two types of inputs: (i) configuration bits, which program a specific function on the fabric, and (ii) primary input bits, which assess whether the programmed function is affected by any faults in the fabric. While applying a different set of input bits for a given configuration of the fabric only requires one clock cycle, shifting in a new configuration is a lengthy serial process. Therefore, with respect to test application time, changing configurations should be done sparingly. From the point of view of ATPG however, while generating test vectors for a TRAP unit, configuration bits and primary input bits are treated in a similar fashion, leading to a generated set of test vectors with high test application time. This problem is further compounded by the multiple versions of the TRAP HDL that ATPG must be conducted on in order to deal with the bi-directional transistors, as explained earlier.

Solution: To minimize test application time, we devise a two-step approach: First, for each configuration that ATPG generates, we use it to model a programmed version and reinvoke ATPG to generate vectors that detect all testable faults for that programmed version. Then, after fault-simulating test vectors across all versions of the HDL, we compact the test set through an Integer Linear Programming (ILP) formulation that sets test application time as its optimization objective. Specifically, as highlighted in Fig. 4, we initially program the HDL (`TRAP_UNIT_V1.v`) using a configuration C_1. Executing ATPG on this programmed HDL (`TRAP_UNIT_V1[`C_1`].v`) restricts the tool from changing the configuration and forces it to detect faults by only changing the primary inputs. The process is repeated for every configuration $(C_1, C_2, \ldots, C_a)$ in this HDL version and vectors in each configuration $(V_{11}, V_{12}, \ldots, V_{ana})$ are tabulated. We then fault simulate the vectors generated from the current version on the other HDL versions. Any fault detected during fault

simulation (`Detected faults fs1`) is removed before running ATPG on the next version. The same procedure is followed for the next HDL version (`TRAP_UNIT_V2.v`) for the remaining faults (`Remaining faults fs1`) and the process iterates until all testable faults are detected. This generates the highlighted configuration list $C_1, C_2, \ldots, C_m$ and vectors within each configuration highlighted as $V_{11}, V_{12}, \ldots, V_{mn_m}$. Next, taking into account the fault coverage of each test vector and the time to load a configuration in TRAP, we formulate test application time minimization as an ILP problem [22].

ILP formulation: Let F be a finite set containing all k ATPG testable faults (`f1,...,fk`) and C_i (`i ∈ (1,m)`) be the list of `m` configurations required to test these faults. Also, let $(V_{i1}, V_{i2}, \ldots, V_{in_i})$ be the n_i ATPG generated vectors for the TRAP HDL programmed with configuration C_i, or $C_i = \{V_{il}\}_{l=1}^{n_i}$. Our approach starts by creating an `[N x k]` matrix `A`, where $N = \sum_{i=1}^{m} n_i$. Each matrix element is denoted as $A_{xl}^{C_i}$, for $x \in (1,k)$, $l \in (1,n_i)$ and $i \in (1,m)$. Element $A_{xl}^{C_i} = 1$ if and only if fault f_x is detected by vector $\{V_{il}\}$. Also, let decision variable $v_l^{C_i} = 1$, if the vector V_{il} in C_i is selected for $l \in (1,n_i)$, and let decision variable $c_i = 1$, if configuration C_i is selected. Furthermore, let w_c and w_v be the weights (*i.e.*, test application times) associated with selecting a configuration and a vector, respectively. Evidently, selecting a new configuration is more expensive than selecting a vector from an existing configuration. Our goal is to select configurations and vectors within these configurations such that each fault is covered at least once and total test application time is minimized. Thus, our objective function is modeled in Eq. (1). Ensuring that each fault is covered at least once is achieved by the constraint of Eq. (2). Similarly, ensuring that, if a vector is selected, its corresponding configuration is also selected is achieved by the constraint of Eq. (3). Finally, Eq. (4) and Eq. (5) restrict the decision variables to binary integers.

$$\text{Minimize: } \sum_{i=1}^{m}\sum_{l=1}^{n_i} w_v \cdot v_l^{C_i} + \sum_{i=1}^{m} w_c \cdot c_i \tag{1}$$

$$\text{Subject to: } \forall x = 1 \ldots k \sum_{i=1}^{m}\sum_{l=1}^{n_i} A_{xl}^{C_i} \cdot v_l^{C_i} \geq 1 \tag{2}$$

$$\text{Subject to: } \forall i = 1 \ldots m, l = 1 \ldots n_i \quad \left(c_i - v_l^{C_i} \geq 0\right) \tag{3}$$

$$\text{Variable: } 0 \leq c_i \leq 1, \forall i = \in 1 \ldots m \tag{4}$$

$$\text{Variable: } 0 \leq v_l^{C_i} \leq 1, \forall i = \in 1 \ldots m, l = 1 \ldots n_i \tag{5}$$

V. Design-Aware Test Generation for TRAP

Challenge: ATPG is unaware of the distinctive architectural features and the role played by the various modules within the TRAP fabric in constructing logic circuits. This results in sub-optimal test configurations due to the unguided exploration of the extensive functional space stemming from the large number of programming bits available as inputs to ATPG.

Solution: We leverage our knowledge of functional capabilities of core components of TRAP architecture to craft custom combinations of configurations and input vectors to *efficiently* detect all the faults in a TRAP unit. For the interconnect network, which is designed for dense net routing, we employ a path-based test approach. For the interface and the TA, which are designed for implementing a variety of logic functions with high utilization, we employ a program-based test approach.

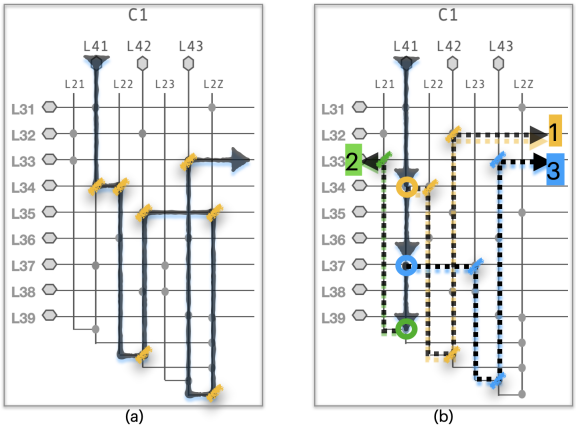

Fig. 5. Testing interconnect transistors with gate (a) Stuck-at-0, (b) Stuck-at-1

A. *Path-based test approach for interconnect network*

The interconnect network of a single unit consists of 72 bi-directional nMOS switches. Stuck-at faults at the gate of these switches will result in them being permanently ON or OFF, while stuck-at faults at the source or drain will result in one of the two possible logic values not being attainable in paths that include that switch. Our path-based approach leverages the observation that multiple switches that are expected to be ON can be tested simultaneously by connecting them in series, while multiple switches that are expected to be OFF can be tested simultaneously through connection in parallel.

Paths for in-series testing of interconnect faults: Testing a stuck-at-0 fault at the gate of an nMOS transistor requires the transistor to be turned ON so that a specific logic value can pass through it. This specific logic value remains a differentiator between fault-free and faulty circuits even when multiple transistors-under-test are connected in series in a path. In addition to testing for stuck-at-0 faults at the gates of these transistors, such a path can also test for stuck-at faults in the source and drain ports of these transistors.

For example, a test route starting at input track L41, traversing through multiple tracks, and ending at output track L33 is shown in Fig. 5(a). The configuration enabling this route tests the 7 highlighted switches for the above-mentioned faults. When the highlighted switches are turned ON and the rest are turned OFF, a logic '1' vector at input L41 tests for stuck-at-0 faults on the source and drain terminals of switches and tracks on this route. Similarly, a logic '0' vector tests for stuck-at-1 faults. Either one of these values also tests for stuck-at-0 faults on the gate terminals of these 7 switches.

For effective testing, the path should utilize each track exactly once. Any overlap of tracks within a path could mask faults, rendering them undetectable. This constraint imposes a limit of up to two switches on a track being tested within a single path. However, when the selected path terminates at an output track, only one switch on that horizontal track can be

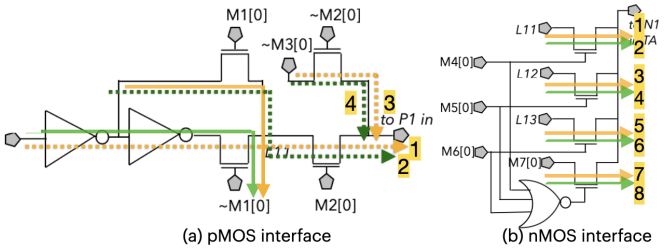

Fig. 6. Programming paths through (a) pMOS interface (b) nMOS interface

tested. Since there are a total of six switches on each horizontal track, *a minimum of 4 configurations with 2 vectors each* are needed in order to comprehensively test all single stuck-at-0 faults on interconnect transistor gate terminals. For each of them, both logic '1' and logic '0' must be propagated through the corresponding paths to test source and drain terminals for stuck-at-0 and stuck-at-1 faults, respectively.

Paths for parallel testing of interconnect faults: Testing a stuck-at-1 fault at the gate of an nMOS transistor requires the transistor to be turned OFF. In this case, we cannot connect turned OFF transistors in series to concurrently test whether any one of them conducts, since subsequent ones will mask the fault. Instead, we need an individual path for each transistor under test. Nevertheless, we can establish multiple such paths in a single fabric configuration to test transistors in parallel.

For example, Fig. 5(b) shows three transistors-under-test (in circles) along with interconnect transistors (highlighted) that are turned ON to create unique paths to outputs. For instance, the switch circled in yellow is tested through the path marked as 1. Here, the input on track L41 connects to track L34 through the transistor-under-test (which is turned OFF), the path then continues through L22 and L42 and ends on output track L32. Under fault-free conditions, the output remains in a high-impedance state. However, if and only if the transistor-under-test is stuck-at-1, then a value will be observed at the output. Similarly, switches encircled in blue and green are concurrently tested in the same configuration through the matching colored paths. Considering the limitations imposed by the number of outputs and the specific switches required to create a path from every transistor to an output, *a minimum of 4 configurations with 1 vector each* are needed to test for stuck-at-1 faults in all 72 switches in the interconnect network.

B. Program-based test approach for Interface and TA

Faults in the interface and the TA can be tested by understanding the choices offered by the fabric and selecting a small subset that exposes the difference between fault-free and faulty circuits. Fig. 6(a) shows two possible options for programming the TA pull-up network via the pMOS interface, which include (i) a **signal path** from the interconnect (1-2), or (ii) a **memory path** from the SRAM (3-4). The paths marked in yellow (1-3) are tested by passing a logic '0' while the paths marked in green (2-4) are tested by passing a logic '1'.

The remaining paths are assessed through the nMOS interface of Fig. 6(b). Options for programming the pull-down network of the TA via the nMOS interface include (i) a **signal path** from the interconnect (1-6), or (ii) a **memory path** from the SRAM (7-8). We note that since the pull-down network has more paths than the pull-up portion, testing it requires more configurations. To cover all stuck-at faults, we need to program the fabric so that each path is included in a configuration. Paths in the interface connect to TA transistor gates and become observable by programming appropriate logic on the TA.

The 9 pMOS and 9 nMOS transistors in the TA need to be tested for 4 and 8 interface paths, respectively. Additionally, the middle 6 nMOS transistors (X and Y) must be tested for both ON and OFF paths. Thus, a total of 120 paths need to be tested in a unit. Our objective is to program the TA in a minimal number of configurations such that all of these paths are established in at least one configuration and all faulty responses remain observable through these paths. Evidently, there exist four options for establishing paths through the interface, as summarized in Fig. 7 for the pull-up network.

1) ON signal path testing: When transistors are turned ON, they can be stitched in series and tested together. In this configuration, the pMOS transistors are programmed through path 1. Fig. 7(a) shows an example where pMOS P3 and P1 are connected in series in each column and tested together. The fault-free output is logic 1, while a fault results in high impedance. When programmed through a signal, the pull-up and pull-down networks can be tested with the same configuration by varying the input vector. Fig. 8 shows a configuration where the nMOS transistors are programmed through path 2. Vector0 tests the pMOS P1 and P3. Vector1 tests the nMOS N1 and N3. Vector2 and Vector3 test P2 and N2, respectively. The interface allows the nMOS transistors to be programmed via 2 additional paths (4-6), thus requiring another 2 configurations with 2 vectors each. Collectively, *testing ON signal paths requires 3 configurations and 8 vectors*.

2) ON memory path testing: When the interface passes a value stored in SRAM, the TA transistors can, again, be tested in series. In this case, however, we can only test either the pull-up or the pull-down network in a single configuration. Indeed, allowing both networks to simultaneously pass their output leads to an undefined state. Fig. 7(b) shows the pMOS transistors of the TA being tested via an ON memory path. Two additional configurations are required to test transistors N1-N3 and transistors P2 and N2, respectively. In total, *testing ON memory paths requires 3 configurations and 3 vectors*.

3) OFF signal path testing: Only one transistor can be tested in its OFF state, while others have to be ON to create a path to the output. Fig. 7(c) shows a test for transistor P1 turned OFF. P3 is turned ON to create a path to the output. Another 2 vectors are required to test P2 and P3 in a similar fashion. The same set of vectors can also test that the nMOS transistors in the pull-down network are turned OFF. The nMOS transistors require additional configurations to test the multiple programming paths through the interface. Collectively, *testing OFF signal paths requires 3 configurations and 9 vectors*.

4) OFF memory path testing: Similar to the previous case, Fig. 7(d) shows a test for P1 turned OFF through a memory bit. P3 is, again, turned ON to create a path to the output. We can test the pull-up and pull-down network in the same

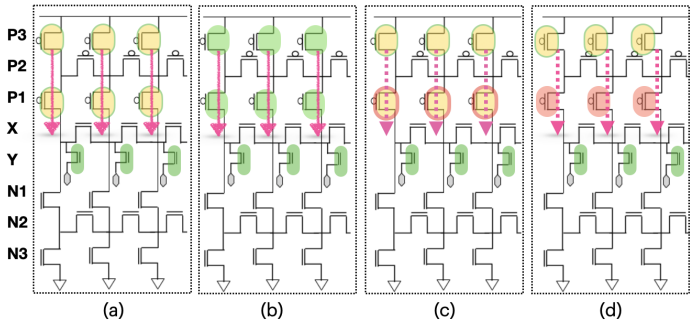

Fig. 7. Testing configurations for the pull-up network in the transistor array

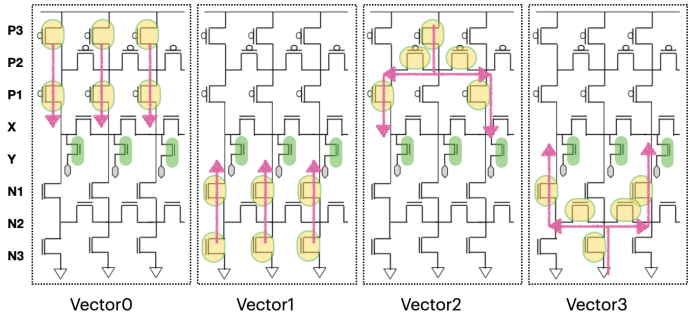

Fig. 8. Four vectors required to test the ON signal path

configuration by varying the input vector value of the ON transistor. Transistors P3 and N3 can be tested using a similar configuration and vector. Transistors P2-N2 can be tested in another configuration. Finally, 2 more configurations with 1 vector each are needed to test that the middle nMOS transistors (X and Y) are turned OFF. In total, *testing OFF memory paths requires 5 configurations and 7 vectors*.

VI. Experimental Evaluation

A TRAP unit has a total of 1360 uncollapsed faults. Among them, 196 can only be detected using IDDQ testing, as explained in Section IV-B. In addition, there are 6 faults on the source terminals that connect transistors of the TA to power and ground, which are redundant (*i.e.*, stuck-at-1 fault on a line connected to power, or stuck-at-0 fault on a line connected to ground) and 14 faults that are associated with the floating nodes that connect a unit to its adjacent unit in the fabric, which are untestable when testing a single unit. The remaining 1144 faults, which are testable by ATPG, require five HDL versions wherein bi-directional transistors are converted in different combinations to uni-directional ones, as explained in Section IV-A. Collectively, as summarized in Table I, the baseline approach of running ATPG (*i.e.*, Synopsys TestMax) on these five versions, each time removing from the fault-list the faults that were detected in the previous versions, generates a total of 253 patterns, each consisting of 9 primary input bits and 141 configuration bits. Since ATPG is oblivious to the difference between inputs and configuration bits, it ends up generating 253 distinct configurations, resulting in a test application time of ∼36K clock cycles.

Table II summarizes the results of the ILP-based test compaction approach of Section IV-C. The Python PuLP library was used to formulate the ILP and CPLEX was used to solve it. The solution of the ILP-based test compaction yields 131 test vectors in 101 configurations to detect all 1144 faults. This translates to a test application time of ∼14K clock cycles, or a ∼2.5x reduction over the baseline approach.

TABLE I
Baseline ATPG Approach

HDL Version	ATPG Generated Configurations	Detected Faults
V1	109	813
V2	57	107
V3	52	152
V4	18	23
V5	17	49
Total	**253**	**1144**
Test Application Time		**35,926 Cycles**

TABLE II
ILP Approach

ILP Selected Configurations	ILP Selected Vectors
47	56
21	29
18	24
3	5
12	17
101	**131**
14,372 Cycles	

TABLE III
Design-Aware Test Generation Approach

Target Tests	Manually Selected Configurations	Manually Selected Vectors
Interconnect Faults		
In-series testing	4	8
Parallel testing	4	4
Interface and TA Faults		
ON signal path	3	8
ON memory path	3	3
OFF signal path	3	9
OFF memory path	5	7
Total	**22**	**39**
Test Application Time		**3,141 Cycles**

Lastly, as shown in Table III the design-aware test generation method of Section V requires only 39 test vectors in a total of 22 configurations to cover all 1144 faults. This, in turn, translates to a test application time of ∼3K clock cycles, or more than an 11x reduction over the baseline approach.

VII. Conclusion & Future Directions

We discussed the limitations and challenges commercial ATPG encounters while generating test vectors for a transistor-level programmable fabric. To overcome the inability of ATPG to handle bi-directional pass transistors, we curated multiple versions of the fabric HDL having the same fault list and differing only in transistor orientation. To detect faults in custom transistor-level structures of the fabric, which do not result in a logically detectable discrepancy, we employed an IDDQ-based test solution that exposes the presence of such faults through high steady-state current values. To surmount the fact that ATPG is oblivious to the difference between programming bits and primary input bits, we developed an ILP-based test compaction solution that considers the relative cost of reconfiguring the fabric vis-a-vis running more test vectors for existing configurations, seeking to optimize test application time. Lastly, to further improve test efficiency, we leveraged our knowledge of the TRAP fabric architecture and devised design-aware test configurations that maximize the excitation and propagation of faults. Collectively, these solutions resulted in *an order of magnitude improvement* in test application time for a complete unit of the TRAP fabric. Future efforts will focus on extending and parallelizing our test solution to an entire TRAP fabric, which consists of a two-dimensional array of units with limited I/O connectivity.

Acknowledgement

This research was supported by the National Science Foundation (NSF 2155208) and the I/UCR Center For Hardware and Embedded System Security and Trust (CHEST P16_22).

REFERENCES

[1] P. Mohan, O. Atli, J. Sweeney, O. Kibar, L. Pileggi, and K. Mai, "Hardware redaction via designer-directed fine-grained eFPGA insertion," in *IEEE/ACM Design, Automation Test in Europe Conference (DATE)*, 2021, pp. 1186–1191.

[2] K. Shamsi, M. Li, K. Plaks, S. Fazzari, D. Z. Pan, and Y. Jin, "IP protection and supply chain security through logic obfuscation: A systematic overview," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 24, no. 6, pp. 1–36, 2019.

[3] J. Bhandari, A. K. Thalakkattu Moosa, B. Tan, C. Pilato, G. Gore, X. Tang, S. Temple, P.-E. Gaillardon, and R. Karri, "Exploring eFPGA-based redaction for IP protection," in *IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, 2021, pp. 1–9.

[4] M. M. Shihab, J. Tian, G. R. Reddy, B. Hu, W. Swartz, B. Carrion Schaefer, C. Sechen, and Y. Makris, "Design obfuscation through selective post-fabrication transistor-level programming," in *IEEE/ACM Design, Automation Test in Europe Conference (DATE)*, 2019, pp. 528–533.

[5] J. Tian, G. R. Reddy, J. Wang, W. Swartz, Y. Makris, and C. Sechen, "Field programmable transistor array featuring single-cycle partial/full dynamic reconfiguration," in *IEEE/ACM Design, Automation Test in Europe Conference (DATE)*, 2017, pp. 1336–1341.

[6] M. M. Shihab, B. Ramanidharan, G. R. Reddy, J. Tian, W. Swartz, C. Sechen, and Y. Makris, "CASPER: CAD framework for a novel transistor-level programmable fabric," in *IEEE International Symposium on Circuits and Systems (ISCAS)*, 2020, pp. 1–5.

[7] Q. Huang, J. Tian, T. Broadfoot, X. Xu, B. Hu, M. Shihab, A. Jain, V. Salimath, Y. Makris, and C. Sechen, "Toward accurate timing analysis for transistor-level programmable fabrics," in *IEEE Dallas Circuit And System Conference (DCAS)*, 2022, pp. 1–6.

[8] A. Jain, T. Broadfoot, Y. Makris, and C. Sechen, "Quo vadis signal? automated directionality extraction for post-programming verification of a transistor-level programmable fabric," in *IEEE/ACM Design, Automation Test in Europe Conference (DATE)*, 2023, pp. 1–2.

[9] S. Kundu, "Gatemaker: a transistor to gate level model extractor for simulation, automatic test pattern generation and verification," in *IEEE International Test Conference (ITC)*, 1998, pp. 372–381.

[10] R. L. Wadsack, "Fault modeling and logic simulation of CMOS and MOS integrated circuits," *The Bell System Technical Journal*, vol. 57, no. 5, pp. 1449–1474, 1978.

[11] R. Bryant, "Extraction of gate level models from transistor circuits by four-valued symbolic analysis," in *IEEE International Conference on Computer-Aided Design (ICCAD)*, 1991, pp. 350–353.

[12] T. McDougall, A. Parashkevov, S. Jolly, J. Zhu, J. Zeng, C. Pyron, and M. Abadir, "An automated method for test model generation from switch level circuits," in *IEEE Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2003, pp. 769–774.

[13] X. Lin, W.-T. Cheng, T. Kobayashi, and A. Glowatz, "On modeling CMOS library cells for cell internal fault test pattern generation," in *IEEE Asian Test Symposium (ATS)*, 2021, pp. 103–108.

[14] M. Renovell, J. Portal, J. Figueras, and Y. Zorian, "Minimizing the number of test configurations for different FPGA families," in *IEEE Asian Test Symposium (ATS)*, 1999, pp. 363–368.

[15] M. Abramovici, J. Emmert, and C. Stroud, "Roving STARs: an integrated approach to on-line testing, diagnosis, and fault tolerance for FPGAs in adaptive computing systems," in *Proceedings Third NASA/DoD Workshop on Evolvable Hardware*, 2001, pp. 73–92.

[16] W. K. Huang, F. Meyer, X.-T. Chen, and F. Lombardi, "Testing configurable LUT-based FPGAs," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 6, no. 2, pp. 276–283, 1998.

[17] J. Fugate, G. Stitt, N. V. R. Masna, A. Dasgupta, S. Bhunia, N. Dorairaj, and D. Kehlet, "An exploration of ATPG methods for redacted IP and reconfigurable hardware," in *IEEE VLSI Test Symposium (VTS)*, 2023, pp. 1–7.

[18] A. van de Goor and I. Tlili, "March tests for word-oriented memories," in *IEEE/ACM Design, Automation Test in Europe Conference (DATE)*, 1998, pp. 501–508.

[19] M. Renovell, J. Portal, J. Figueras, and Y. Zorian, "SRAM-based FPGA's: testing the LUT/RAM modules," in *International Test Conference (ITC)*, 1998, pp. 1102–1111.

[20] M. M. Shihab, B. Ramanidharan, S. S. Tellakula, G. Rajavendra Reddy, J. Tian, C. Sechen, and Y. Makris, "ATTEST: Application-Agnostic Testing of a Novel Transistor-Level Programmable Fabric," in *IEEE VLSI Test Symposium (VTS)*, 2020, pp. 1–6.

[21] C. Sechen, G. Makris, and T. Broadfoot, "Field programmable transistor arrays," Jun. 14 2022, US Patent 11,362,662.

[22] P. Drineas and Y. Makris, "Independent test sequence compaction through integer programming," in *IEEE International Conference on Computer Design (ICCD)*, 2003, pp. 380–386.