Modeling Bidirectional Switches for Enabling Logic Equivalence Checking in a Transistor-level Programmable Fabric

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

***Abstract*—We explore the challenges associated with developing a verification solution for a TRAnsistor-level Programmable fabric (TRAP). The TRAP architecture employs bidirectionally-operated pass transistors to implement its logic and interconnect network, aiming for high density. However, existing Logic Equivalence Checking (LEC) methods and tools do not support the primitives necessary to model such transistors in hardware description languages (HDL). Consequently, verifying the functionality programmed by a given bitstream on TRAP is not inherently feasible. To overcome this limitation, we propose a method that automates the determination of signal flow direction through bidirectional pass transistors for a given bitstream. Subsequently, we convert the HDL description of the programmed fabric to exclusively utilize unidirectional transistors. This transformation allows us to leverage commercial EDA tools for verifying logic equivalence between the transistor-level HDL representation of the programmed fabric and the post-synthesis gate-level netlist. We have successfully applied the proposed method to verify various benchmark circuits programmed on the TRAP fabric.**

***Index Terms*—Transistor-Level Programming, Pass Transistor, Verification, Logic Equivalence Checking**

I. Introduction

A TRAnsistor-Level Programmable fabric (TRAP) [1], shown in Fig. 1, was developed for the purpose of protecting hardware intellectual property (IP) from an untrusted foundry through integrated circuit (IC) redaction [2]. While traditional embedded Field Programmable Gate Arrays (eFPGAs) rely on Look-Up Tables (LUTs) to implement logic functions, TRAP pushes the granularity of post-fabrication programmability down to the transistor level. Essentially, TRAP can be seen as a sea of transistors whose programmable interconnectivity can be used to realize logic functions. To enable the use of conventional standard cell placement tools for mapping logic functions on TRAP, a standard cell library comprising cells of same height and variable width is utilized. TRAP-specific routing and bitstream generation tools round out the flow for implementing a circuit. TRAP has demonstrated an order of magnitude reduction in area and power, as well as significantly improved performance, as compared to LUT-based eFPGA solutions, while also presenting formidable obstacles to reverse engineering attacks [1].

However, for TRAP to gain widespread adoption and utility, it requires support from commercial computer-aided design (CAD) tools that encompass all design-related tasks, including verification. This capability is particularly crucial for unconventional custom fabrics, such as TRAP, to ensure confidence in the correctness of the circuit implementation. Although industry-standard LEC tools possess the ability to comprehend

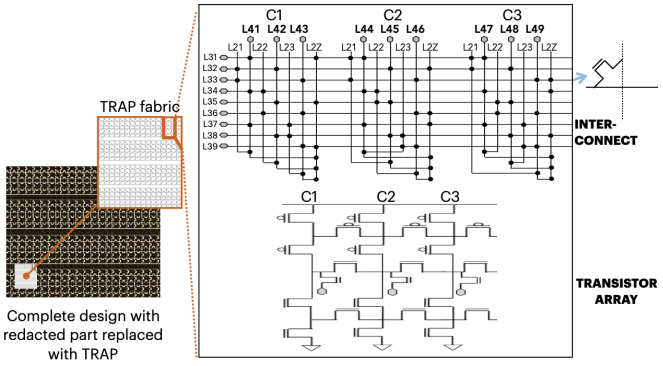

Fig. 1. TRAP-based IC redaction & core TRAP components

and handle transistor-level constructs, this alone is insufficient for verification of an entire design. Describing a design at the switch level results in an enormous number of nodes to compare, making it time-prohibitive. Instead, LEC tools employ a transistor abstraction method that converts switch-level designs into their gate-level equivalents. This conversion assumes that the signal flow direction in MOS transistors is resolved and known to the LEC tool. In TRAP, however, where bidirectional pass transistors are used for high density, determining the actual signal direction poses a significant challenge, even if programming information is available [3].

The problem of resolving transistor-level signal flow directionality has been previously addressed in ASIC design by extracting signal strength from layout [4] in order to determine the source and drain terminals. This method relies on analysis of variable transistor sizes, rendering it unsuitable for programmable transistor-level fabrics such as TRAP, where all transistors have equal sizes. An informative discussion of the challenges encountered when abstracting configurable logic in FPGAs (i.e., at the MUX- and LUT-level) is provided in [5]. While it covers pass transistors used in the interconnect, it does not address bidirectional pass transistors with undetermined signal directionality. Lastly, high-level approaches to modeling a reconfigurable architecture in HDL were proposed in [6], [7]. The generated HDL can be simulated after programming it with a bitstream for simulation-based functional validation. However, the presence of thousands (or even millions) of metal tracks and switches and the limited visibility into the internal nodes of the programmed fabric make simulation-based validation a rather inefficient option for transistor-level programmable fabrics such as TRAP.

To address these limitations, we discuss a TRAP verification framework, depicted in Fig. 2, which consists of the following:

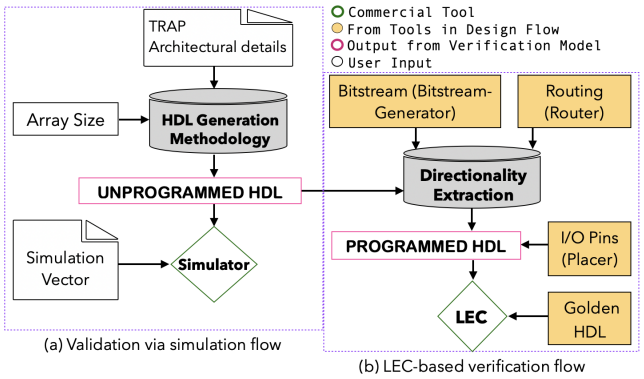

Fig. 2. Verification methodology framework for TRAP

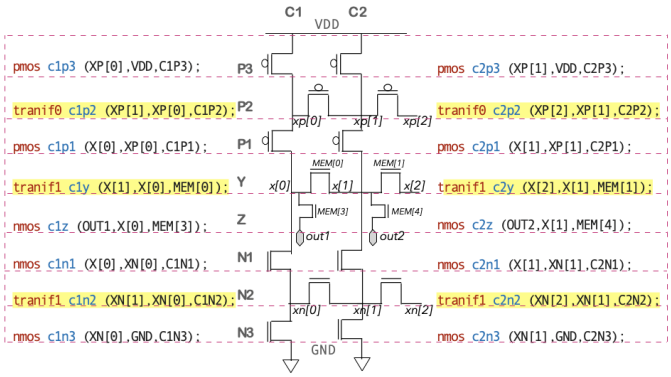

Fig. 3. HDL for two columns of an unprogrammed Transistor Array (TA)

HDL Modeling of Unprogrammed TRAP Fabric: We derive a parameterized HDL model for an *unprogrammed* TRAP fabric, which can be used with a programming bitstream for simulation-based validation of TRAP designs.

Automated Signal Directionality Extraction: We introduce algorithms which leverage the information produced by the placer, router and bitstream generator when mapping a design to a TRAP fabric, to generate a signal flow graph and deduce direction across bidirectional pass transistors. This information enables derivation of an HDL model for a *programmed* TRAP fabric, which can be used by LEC tools to mathematically assert that the Boolean expression implemented by a transistor-level netlist of TRAP is the same as the one implemented by the post-synthesis gate-level netlist.

II. TRAP Architecture and HDL Modeling

In the TRAP fabric of Fig. 1, programmable bidirectional transistors are used in (i) the Transistor Array (TA), to form logic gates and, (ii) the interconnect network, to stitch gates into a circuit. Below, we derive an HDL description of TRAP.

A. Transistor Array (TA)

As shown in Fig. 1 identical sets of three columns of transistors repeat along a TRAP row, where each set of three columns (`C1`, `C2`, and `C3`) comprises 24 transistors. Fig. 3 shows two of the columns, where each column contains three pull-up PMOS transistors (`P1` – `P3`) and three pull-down NMOS transistors (`N1` – `N3`). Additionally, in each column (*e.g.*, `C1`), there is an NMOS transistor that connects the output to the corresponding interconnect track in the same column. Finally, there are three horizontally-oriented MOS transistors that connect the adjacent columns.

Fig. 3 shows transistor-level HDL code written for two columns of the TA. The left side shows the code for switches in column `C1`, while the right side shows the code for switches in column `C2`. This code represents the actual topology of the various transistors in a column and how they are interconnected with neighboring columns. In HDL, modeling with built-in *pmos/nmos* primitives requires that we define the input and output for a transistor, while a *tranif0/tranif1* construct, which is a bidirectional PMOS/NMOS respectively, has no such requirement. Thus, switches with known signal directions (*i.e.*, one node is clearly at a higher potential than the other) are modeled as *pmos/nmos*, while switches for which signal direction is not established are modeled as *tranif0/tranif1*.

Depending on the logic being implemented on a set of columns in the TA, the so-called horizontal transistors can have a signal flow in either direction (highlighted in yellow in Fig. 3). In each column, transistors `P3` and `P1` are defined as unidirectional PMOS devices because the signal flow direction through them is known. However, the specific signal direction for transistor `P2` will depend on how the transistor gates are programmed in this column; therefore, `P2` must be defined as *tranif0*. Similarly, `N1` and `N3` are defined as unidirectional NMOS devices in the pull-down network, while `N2` is defined through the *tranif1* construct. Transistor `Y` must similarly be defined as *tranif1* since its signal direction depends on the orientation of neighboring transistors. Meanwhile, transistor `Z` can be defined as a conventional NMOS device since the signal flow is always outwards to the interconnect network. Gate inputs connected to *MEM[n]* are directly controlled by the SRAM that holds the programming configuration of the design being implemented.

B. Interconnect Network

Shown in Fig. 1, the TRAP interconnect network for three columns (`C1, C2 and C3`) consists of nine global horizontal (`L3`), nine global vertical (`L4`) and twelve local vertical (`L2`) tracks. The global tracks connect to the corresponding global tracks of a neighboring set of three columns via pass transistors or bidirectional repeaters. Inside the set of three columns, tracks (global or local) connect to various neighboring tracks through switches, implemented as pass transistors.

Fig. 4 shows the transistor-level HDL for one column of the interconnect network of TRAP. Switches, shown as dots, are NMOS pass transistors that connect orthogonal metals (*i.e.*, (`L3`) and (`L4`)) at multiple locations and also connect metal layers (`L2`) and (`L4`). When a switch is turned ON, the two metal layers connected by the switch share the same signal. The signal flow direction in the interconnect network depends on the application mapped on it. Hence, the bidirectional NMOS construct *tranif1* is used to represent each switch. Fig. 4 also shows the HDL for the interconnect network, where a switch is modeled as a connection between two metal layers. Switches are controlled by programming bits (`MEM[n]`), which are stored in SRAM and configure the TRAP fabric.

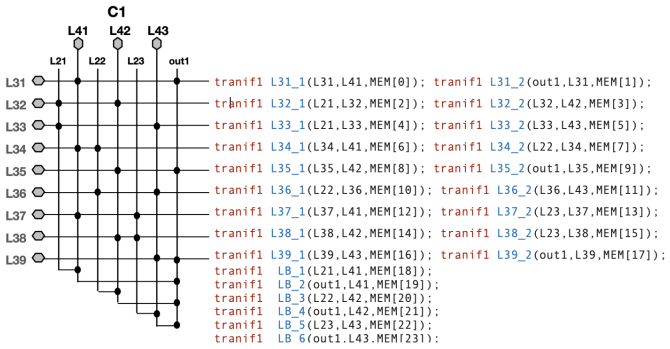

Fig. 4. Unprogrammed interconnect network of TRAP

III. Automated Directionality Extraction

The HDL model of the unprogrammed TRAP fabric can be combined with a programming bitstream to perform simulation-based validation. However, the use of the HDL construct *tranif1/tranif0* for modeling bidirectional pass transistors is not supported by LEC tools. Therefore, these bidirectional constructs must be converted into their equivalent pmos/nmos form, which requires designation of the source and drain terminals of these transistors. This, in turn, requires knowledge of the signal flow direction through these bidirectional pass transistors, which is unavailable until after the device is programmed and, even then, it is not trivial to extract. We note that an alternative approach could replace each bidirectional pass transistor with a LEC-compatible propositional logic clause describing the transistor operation as a switch. However, additional challenges revolving around the notion of electrical drive and the possible formation of cyclical structures arise while seeking to faithfully model the electrical behavior of the TRAP fabric in propositional logic. Addressing these challenges ultimately also requires modeling signal flow direction.

In this section, we describe a methodology for automatically extracting directionality of bidirectional pass transistors and generating an HDL description of the programmed TRAP fabric which can be used by LEC tools for verification. To this end, our method requires (i) the HDL description of the unprogrammed TRAP fabric, (ii) the typical information generated by commonly used placement and routing tools, when mapping a circuit onto the TRAP fabric and (iii) the programming bitstream itself. Such information reflects the location of primary I/Os, as well as the routing path for each net, which we use to extract signal directionality. First, we describe how to determine signal-flow direction in the TA, which determines the logic gates implemented thereon. Then, we describe how to determine the signal-flow direction of pass transistors in the interconnect network, which determines the connections between these logic gates.

A. Resolving Signal Direction in the Transistor Array

Bidirectional pass transistors in the TA provide flexibility for implementing any standard logic gate on the array by enabling signal paths within and across multiple columns of transistors. Evidently, depending on how such transistors are programmed (and therefore also connected together), not only is the logic being programmed on the fabric different,

Algorithm 1: Signal Direction Extraction in TRAP TA

```
PT : Pass Transistor; N_i,N_{i+1} : Nodes of PT
Input: Prog. Bitstream; Unprogrammed TA HDL
for each bidirectional PT in sub-column C_i of
  unprogrammed TA HDL do
    if Gate port of PT is connected to 1/0) then
        "remove" PT from TA /* Case-A */
    else if Gate port of PT is connected to 0/1) then
        "replace" PT with tran /* Case-B */
    else if Gate port of PT is connected to signal then
        "replace" PT with nmos/pmos /* Case-C */
        if PT has either N_i/N_{i+1} connected to
          power/[output] via ON transistor then
            /* Case-Ca */
            Source = N_i/N_{i+1}/ [N_{i+1}/N_i]
            Drain = N_{i+1}/N_i/ [N_i/N_{i+1}]
        else if PT has either N_i/N_{i+1} connected to
          power/[output] via transistor with signal then
            /* Case-Cb: find complete path */
            if N_{i+1}/N_i connected to output/[power] then
                Source = N_i/N_{i+1}/[N_{i+1}/N_i]
                Drain = N_{i+1}/N_i/[N_i/N_{i+1}]
            else
                repeat (Case-Cb for i=i+1)
```

but also the direction of how signals flow across each pass transistor is different. While prior work exists in assessing signal flow direction in CMOS circuits [8], resolving signal directionality in TRAP solely by analyzing the TA is rather challenging. Indeed, a key issue encountered is that, in addition to the transistors implementing the functionality, additional permanently turned ON or OFF transistors are used in order to stitch together the logic functionality and to delineate/isolate logic gates from the rest of the array, respectively. In TRAP, for example, while extracting signal-flow direction for the vertically-oriented transistors irrespective of the bitstream is reasonably straightforward, as explained in Section II, doing so for the horizontally-oriented pass transistors is not possible.

To handle the latter, we developed a method summarized in Algorithm 1, which uses the programming bitstream along with two important observations regarding the TRAP architecture. First, for signal integrity, the maximum number of NMOS or PMOS transistors that can be stitched in series is small (i.e., three). Second, the location of the output nodes in each column of the TA is pre-determined. Our algorithm categorizes horizontal bidirectional pass transistors based on the signal that drives their gate and accordingly acts as follows:
If gate input is OFF: When bidirectional pass transistors are programmed to receive a gate input of logic 0 (for NMOS) or logic 1 (for PMOS), they are permanently turned off. In this case (Case-A), we model the transistor as an open circuit and remove the corresponding *tranif0/tranif1* HDL code construct.
If gate input is ON: When bidirectional pass transistors are programmed to receive a gate input of logic 1 (for NMOS) or logic 0 (for PMOS), they are permanently turned on. In this case (Case-B), the nodes connecting the source and drain are shorted. Therefore, we replace the corresponding

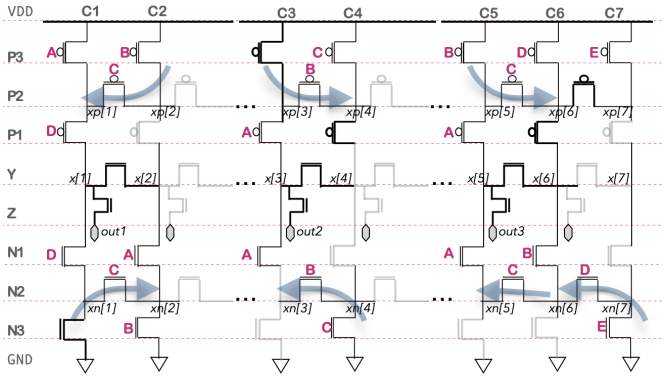

Fig. 5. Signal flow deduction in logic programmed on Transistor Array

TABLE I
SUMMARY OF TRANSISTOR ORIENTATION FOR CASES IN FIGURE 5

Trans.	Case	HDL (unprogrammed)	HDL (programmed)
C1P2	Case-Cb	tranif0 (xp[2],xp[1],C);	pmos (xp[1],xp[2],C);
C1Y	Case-B	tranif1 (x[2],x[1],MEM);	tran (x[1],x[2]);
C1N2	Case-Ca	tranif1 (xn[2],xn[1],C);	nmos (xn[2],xn[1],C);
C3P2	Case-Ca	tranif0 (xp[4],xp[3],B);	pmos (xp[4],xp[3],B);
C3Y	Case-B	tranif1 (x[4],x[3],MEM);	tran (x[3],x[4]);
C3N2	Case-Cb	tranif1 (xn[4],xn[3],B);	nmos (xn[3],xn[4],B);
C5P2	Case-Ca	tranif0 (xp[6],xp[5],C);	pmos (xp[6],xp[5],C);
C5Y	Case-B	tranif1 (x[6],x[5],MEM);	tran (x[5],x[6]);
C5N2	Case-Cb	tranif1 (xn[6],xn[5],C);	nmos (xn[5],xn[6],C);
C6P2	Case-B	tranif0 (xp[7],xp[6],B);	tran (xp[6],xp[7]);
C6N2	Case-Cb	tranif1 (xn[7],xn[6],D);	nmos (xn[6],xn[7],D);
C6Y	Case-A	tranif1 (x[7],x[6],MEM);	N/A

tranif0/tranif1 HDL code construct with the primitive *tran*, which permanently shorts the two nodes in its arguments.

If gate input is a signal: In this case (Case-C), we convert *tranif0/tranif1* to a directional *pmos/nmos* construct by resolving the two remaining ports as source and drain, based on their proximity to power/output ports to which they are connected through conducting transistors. For example, a node connected to a power port through an ON transistor is classified as source.

Fig. 5 shows the implementation of an **OAOI211** = `!(((A+B)C)+D)` in columns C1-C2, **NAND3** = `!(ABC)` in columns C3-C4, and **AOAAI2111** = `!((AB+C)DE)` in columns C5-C6-C7 of the TA, with outputs at out1, out2, and out3, respectively. The horizontally-oriented bidirectional pass transistors (receiving input C in C1, input B in C3, input C in C5, and input D in C6) receive a signal as the gate input, while those shown in bold are permanently ON (to establish inter-column connections) and those in grey are permanently OFF (to isolate adjacent columns). Arrows show the signal flow direction through the bidirectional pass transistors in Fig. 5, as determined by Algorithm 1. Table 1 lists the bidirectional transistors (in columns C1, C3, C5, and C6) from the example in Fig. 5, along with their configuration and the HDL representation in the unprogrammed and programmed fabric. For example, transistor C1P2 in the first column of Table I represents Case-Cb in Algorithm 1, as transistors with signals B and D establish a path from the power net to the gate output. Thus, C1P2 is defined as a *pmos (drain, source, gate)* with drain as xp[1] and source as xp[2].

B. Resolving Signal Direction in the Interconnect Network

The regular interconnect architecture in TRAP has orthogonal metal lines connected via bidirectional NMOS pass transistors. The orientation of these pass transistors varies with each design mapped on the fabric. To resolve their directionality and develop the HDL that describes the programmed interconnect, we rely on the path generated by the router for each net, a list of active switches and a graph of the unprogrammed TRAP interconnect architecture (see Fig. 4). Specifically, as summarized in Algorithm 2, to generate the HDL of the interconnect of the programmed fabric, our algorithm selects a routing path at a time and traverses it starting from the origin of the net until it encounters the first active switch. Then, it determines that the origin of the net is the driving net for the next segment. It then continues traversing the routing path and every time it encounters an active switch, the previous net becomes the driving net, eventually determining independently the signal flow path through each active pass transistor in each branch of the path. Once the source and drain terminals of each active switch are determined by this traversal, our algorithm generates the HDL for the programmed interconnect by replacing each encountered *tranif1* construct of the unprogrammed fabric with an appropriately directed *nmos*. We clarify that Algorithm 2 only considers active switches, *i.e.,* those chosen by the router to be turned ON. All other switches are removed from the graph of the unprogrammed architecture when formulating the HDL of the programmed fabric.

As an example, Fig. 6 shows the routing and the HDL of a partial net on the interconnect network. This net originates at horizontal track L31 and ends at vertical track L42. The placement provides the origin and destination of the net. As shown, the router generates the path of the net (highlighted)

Algorithm 2: Signal Direction Extraction in TRAP Interconnect Network

```
Input: Routing path, Unprogrammed Interconnect HDL
inactive_trans = Total transistors in interconnect
for each net do
    Initialize: src_list = origin;
    tran_list = active transistors on net
    inactive_trans ← inactive_trans - {tran_list}
    for each active transistor (T) on net do
        while tran_list ≠ null do
            if (N_i/N_{i+1} in src_list) then
                Drain = N_{i+1}/N_i ; Source = N_i/N_{i+1}
                Replace T "tranif1" in HDL with "nmos"
                src_list ← src_list + {N_{i+1}/N_i}
                tran_list ← tran_list - {T}
```

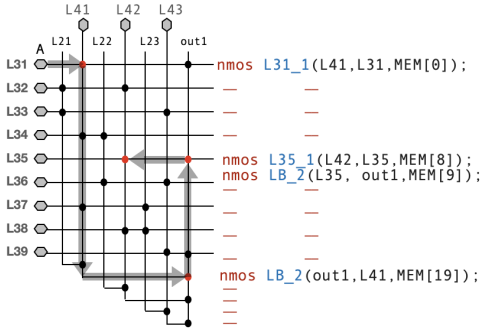

Fig. 6. Programmed Verilog code for the column C1 of the interconnect

and identifies every active transistor (in red). For example, the switch at the junction of wire L31 and L41 is controlled by gate bit MEM[0], receives a signal from L31 (i.e., net origin) and delivers an output to net L41. Hence, this switch is represented by `nmos (L41,L31,MEM[0])`, indicating that L31 (source) is the input and L41 (drain) is the output. Following this, net L41 becomes the input for the next active transistor, which is found at the junction of L41 and out1, and which is controlled by gate bit MEM[19]. Consequently, this switch is represented by `nmos (out1,L41,MEM[19])`.

IV. Scope of Verification Framework

With all *tranif1/tranif0* statements removed, the resulting HDL expressing the programmed transistor-level fabric becomes compatible with commercial LEC tools and can be used for verification against the synthesized netlist. However, while our approach readily accommodates combinational circuits, additional provisions are needed to support sequential circuits.

Sequential equivalence checking is, generally, challenging because of the state space explosion arising when comparing functionality across clock cycles. As a result, it is typically limited to cases where a correspondence between the state-holding elements is already established across the two compared models. This is often the case for common uses of LEC tools, such as proving that optimization or flattening of a circuit has not altered its functionality. In such cases, signal/instance names are usually preserved across the two models and, thus, name matching can establish state element correspondence. When mapping sequential designs on TRAP, however, the synthesized netlist and the unprogrammed TRAP fabric are very different structures. Hence, name matching is ineffective and LEC tools require additional guidance, which in our case is provided by the placement output.

Specifically, TRAP has a built-in flip-flop placed in every third column of the transistor array, which can be connected to the transistor array through a programming bit. The placement algorithm selects which of these flip-flops to use and what flip-flop in the synthesized Verilog netlist to map to each of them. Knowledge of this state-element correspondence partitions sequential equivalence checking into equivalence checking of combinational circuits surrounded by I/Os or flip-flops.

V. Evaluation

To assess effectiveness of our approach in generating appropriate HDL (*i.e.*, Verilog) models for both simulation and verification of a circuit implemented on TRAP, we applied it on the benchmarks shown in Table II. The various TRAP fabric sizes required were designed and laid out using a Global Foundries 12nm process (GF12LP). In all cases, we were able to successfully use simulation tools (*e.g.*, Modelsim) and LEC tools (*e.g.*, Synopsys' Formality) to prove correctness of the TRAP implementation vis-a-vis the synthesized netlist. The first two columns show the name and the number of logic gates in the synthesized benchmark. The third column lists the size of the TRAP fabric required, expressed in number of sets of three columns by the number rows. The fourth and fifth columns show the computation time required by Algorithms 1 and 2, which were implemented in Python 3.1. The sixth column shows the total time required to execute the two algorithms, pre-process I/O and write out the HDL. The seventh column lists the computation time required by the LEC tool. Computation times (in seconds) are for execution on a 3.7 GHz AMD EPYC 8-Core processor running a Linux OS. Lastly, the eighth and ninth columns list the number of programming bits required for each fabric size and the number of code lines in the Verilog model of TRAP wherein the directions of all transistors have been determined, with the latter corresponding roughly to the number of transistors that need to be reasoned upon by the LEC tools. The results corroborate feasibility of using commercial LEC tools for verifying the functionality of a design implemented on the TRAP fabric for the purpose of IC redaction.

TABLE II
Summary of Verification Method Evaluation

Circuit Name	Logic Gates	Fabric Size	Alg1 (sec)	Alg2 (sec)	HDL (sec)	LEC (sec)	Prog. Bits	Code Lines
C17	8	8x16	0.013	0.008	0.088	1.51	21096	5678
C432	122	8x16	0.12	0.38	.542	5.38	21096	6528
C499	306	32x8	0.46	1.25	1.8	10.83	42768	12757
C880	240	24x8	0.26	1.12	1.54	8.24	32040	10030
C1355	313	32x8	0.436	1.8	2.5	11.3	42768	13051
C1908	296	24x8	0.25	1.28	1.71	8.5	32040	10338
C2670	400	8x22	0.22	5.76	6.7	9.21	29268	10778
C3540	637	32x16	1.75	7.73	9.89	24.4	85824	26576
C5315	860	16x24	0.96	13.29	15.93	22.73	64224	23265
C7552	1177	16x28	1.33	18.2	21.1	28.24	74952	27028
C6288	1906	48x24	8.715	44.06	53.2	62.59	193536	61178
B21	15236	180x40	322.3	2330.6	2654.6	862.07	1212660	511146

VI. Conclusion

In order to maximize flexibility in implementing logic functions, transistor-level programmable fabrics such as TRAP often incorporate bidirectional pass transistors. However, these transistors pose challenges when it comes to modeling them in HDL and verifying the functionality of the programmed fabric. The unknown signal directionality of these transistors prevents them from being handled by LEC tools. To address this limitation, the presented algorithms automatically extract signal directionality of bidirectional pass transistors in TRAP. Moreover, they generate an HDL representation of the programmed fabric that can be accepted by LEC, thereby enabling formal verification between a synthesized gate-level netlist and its actual transistor-level implementation on the fabric. Effectiveness of our approach was successfully demonstrated through its application to various benchmark circuits.

References

[1] J. Tian *et al.*, "Field programmable transistor array featuring single-cycle partial/full dynamic reconfiguration," in *DATE*, 2017, pp. 1336–1341.
[2] M. Shihab *et al.*, "Design obfuscation through selective post-fabrication transistor-level programming," in *DATE*, 2019, pp. 528–533.
[3] A. Jain *et al.*, "Quo vadis signal? automated directionality extraction for post-programming verification of a transistor-level programmable fabric," in *DATE*, 2023, pp. 1–2.
[4] D. Blaauw *et al.*, "Derivation of signal flow for switch-level simulation," in *EDAC*, 1990, pp. 301–305.
[5] G. Dupenloup *et al.*, "Transistor abstraction for the functional verification of FPGAs," in *DAC*, 2006, pp. 1069–1072.
[6] X. Chen *et al.*, "High-level modeling and synthesis for embedded FPGAs," in *DATE*, 2013, pp. 1565–1570.
[7] S. Chaudhuri *et al.*, "Efficient modeling and floorplanning of embedded-FPGA fabric," in *FPL*, 2007, pp. 665–669.
[8] A. Baba-Ali and A. Farah, "An efficient algorithm for signal flow determination in digital CMOS VLSI," in *ED&TC*, 1996, pp. 288–293.