CASPER: CAD Framework for a Novel Transistor-Level Programmable Fabric

Mustafa M. Shihab, Bharath Ramanidharan, Gaurav Rajavendra Reddy, Jingxian Tian,
William Swartz Jr., Carl Sechen and Yiorgos Makris
Department of Electrical and Computer Engineering, The University of Texas at Dallas, Richardson, USA
{mustafa.shihab, bharath.ramanidharan, gaurav.reddy, jingxian.tian, bill-swartz, carl.sechen, yiorgos.makris}@utdallas.edu

***Abstract*—A recently proposed TRAnsistor-level Programmable (TRAP) fabric can enable seamless on-die integration of high-density reconfigurable logic with custom ICs. However, state-of-the-art CAD tools are developed for either ASICs or FPGAs and do not support the new architecture. To this end, we present CASPER - a novel CAD framework for implementing designs on the TRAP fabric. CASPER begins with characterizing an ASIC-esque cell library in order to leverage the industry-leading logic synthesis tools for TRAP. We then systematically remodel the TimberWolf and the Versatile Place and Route (VPR) tools to facilitate TRAP-specific design placement and routing, respectively. In addition, we develop a robust programming bitstream generation tool for TRAP. Lastly, we fabricate a 65nm prototype TRAP chip and implement ten ISCAS-85/MCNC benchmark circuits on it. Our evaluation results validate the proposed CAD framework and provide a comparative overhead analysis between TRAP and FPGA.**

I. Introduction

A. Motivation

Notable technological advancement has rendered reconfigurable computing a key apparatus for various emerging applications such as image processing and bioinformatics [1]. Although the Look-Up Table- (LUT-) based FPGAs remain the prevalent choice, several studies have explored alternative reconfigurable architectures based on fine-granular programmability [2]–[4]. In particular, a recent work [5] introduced a transistor-level programmable (TRAP) fabric that has demonstrated uniquely compelling features, such as (i) a better logic density as compared to FPGAs from the same technology node, and (ii) seamless on-die integration with ASICs. Consequently, TRAP is considered as a promising technology for various novel applications of reconfigurable computing such as hardware security (i.e., design obfuscation) [6], [7]. However, before venturing into such applications, the core design of TRAP needs to be thoroughly vetted for limits and limitations. This, in turn, renders it essential to develop a robust computer-aided design (CAD) framework for fast and reliable design-space exploration for the fabric.

B. Challenges

Conventional CAD tools are predominantly designed for ASICs. In addition, most of the CAD tools for reconfigurable fabrics are proprietary and vendor-specific. The few available open-source alternatives, such as the Verilog-to-Routing (VTR) [8] framework, are exclusively designed for LUT-based FPGAs and do not support fine-granular fabrics such as TRAP.

Furthermore, the novel design philosophy of TRAP renders developing a CAD solution for the fabric a challenging task. Typically, hardware designs are implemented following either the ASIC or the FPGA CAD flow, as shown in Figure 1. However, TRAP is a programmable fabric designed to integrate with ASICs. Consequently, design implementation on TRAP requires a unique combination of features from both the ASIC and the FPGA CAD flow. First, while the design synthesis needs to be standard cell-based (like in an ASIC), but also reflects a TRAP-based implementation. Second, the design mapping must be as per TRAP's architecture, while retaining ASIC-compatible floorplanning and placement. Lastly, the design routing and the programming bitstream generation should be consistent with the fabric's design.

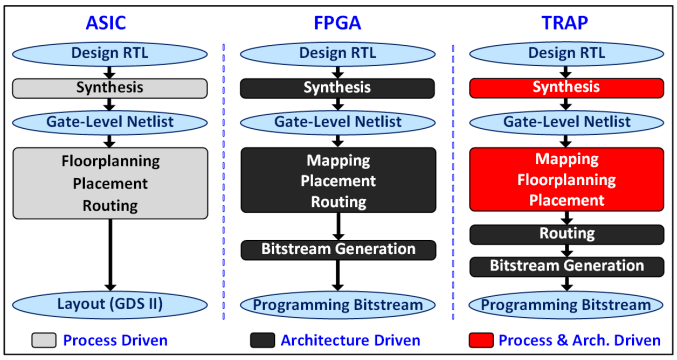


Fig. 1: TRAP leverages components from both the ASIC and the FPGA design-flow, mandating a novel CAD framework.

Based on the above discussion, we can infer that there exists no straightforward mechanism to apply any existing CAD framework for implementing designs on the TRAP fabric.

C. Contributions

To address TRAP's incompatibility with the prevailing CAD tools and to render its promising features available, in this work we present **CASPER** - a full-stack **C**AD fr**A**mework for the tran**S**istor-level **P**rogrammabl**E** fab**R**ic. Specifically, the **key** contributions of this work are as follows:

- **Design Synthesis:** We characterize a 65nm standard cell library for TRAP which enables CASPER to leverage the state-of-the-art logic synthesis tools.
- **Mapping and Placement:** We customize the TimberWolf [9] tool to map and place cells as per their implementation in TRAP and generate an ASIC-like floorplan.
- **Design Routing:** We enable TRAP-based design routing by remodeling the Versatile Place and Route (VPR) [10] tool to capture the fine-granular structure of the fabric.
- **Bitstream Generation:** We develop a Python-based tool that leverages a target design's netlist, placement, and routing files to generate the programming bits for TRAP.
- **Validation and Evaluation:** We fabricate a 65nm prototype TRAP chip and implement ten ISCAS-85 and MCNC benchmark circuits [11] on it.

Our evaluation results demonstrate a successful realization of the proposed CASPER framework and provides compara-

978-1-7281-3320-1/20/$31.00 ©2020 IEEE

Authorized licensed use limited to: Univ of Texas at Dallas. Downloaded on August 22,2021 at 23:45:49 UTC from IEEE Xplore. Restrictions apply.

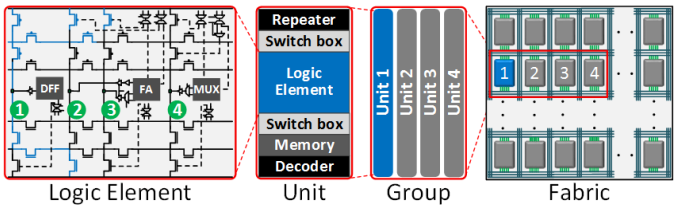


Fig. 2: Hierarchical organization of the TRAP fabric.

tive overhead analysis for TRAP and two conventional FPGAs from the same technology node.

II. Transistor-Level Programming Fabric (TRAP)

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

A. Architecture

The fabric is structured as a hierarchically arranged CMOS sea-of-transistors, as portrayed in Figure 2.

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, wherein each column consists of eight transistors. 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).

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 the fabric's input/output pads through programmable interconnects (not shown in the figure). It should be noted 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.

B. Programming Mechanism

In order to ensure compatibility and streamlined integration with the ASIC CAD-flow, TRAP utilizes a cell library with set 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 2, where a suitable bitstream switches on the required transistors (highlighted in blue) and turns off the remaining ones. The programming bits need to also activate the required multiplexers and switches in the switch boxes.

III. Proposed CAD Framework

Figure 3 portrays the proposed CASPER framework for implementing designs on the TRAP fabric. In this section, we detail the six steps of the CAD tool-flow, namely, cell library generation, logic synthesis, mapping and placement, routing, and programming bitstream generation.

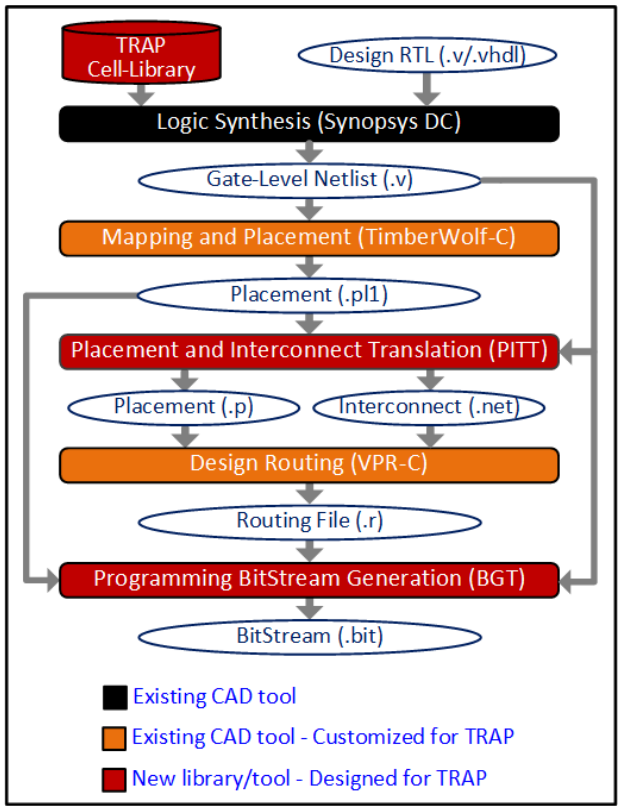


Fig. 3: Proposed CAD tool-flow for the TRAP fabric.

A. Standard Cell Library Generation

The precursory task for the proposed CAD flow is the characterization of logic cells based on their TRAP implementation. Specifically, the cell library should accurately reflect each cell's area, latency, and power characteristics when programmed on the fabric. In this work, we use the library proposed by Tian et al. [5] as our baseline and extend it for a 65nm process technology. The library contains a total of 137 cells, including custom cells for the built-in DFFs, FAs, and MUXs. Each of the cells is first verified for correct functionality through implementation on an extracted schematic of TRAP LE(s), which is then simulated using the Synopsys' HSPICE tool. We then characterize the cells with SiliconSmart ACE from Synopsys [12]. The cell library generation process is completed by creating a database file (.db) using Synopsys' Library Compiler.

B. Logic Synthesis

The design flow begins by synthesizing the register-transfer level (RTL) description of the target design. To this end, preparing the TRAP-specific cell library renders the task of logic synthesis both reliable and straight-forward, as we can leverage any of the state-of-the-art ASIC RTL synthesis tools. For example, in this work, we use the Design Compiler tool from Synopsys for this purpose.

C. Mapping and Placement

The synthesized design netlist needs to be mapped and placed based on the TRAP architecture. The mapping process should translate the logic cells into their corresponding (transistor) column-based representation. On the other hand, the cell placement should be compatible with the row-based

Authorized licensed use limited to: Univ of Texas at Dallas. Downloaded on August 22,2021 at 23:45:49 UTC from IEEE Xplore. Restrictions apply.

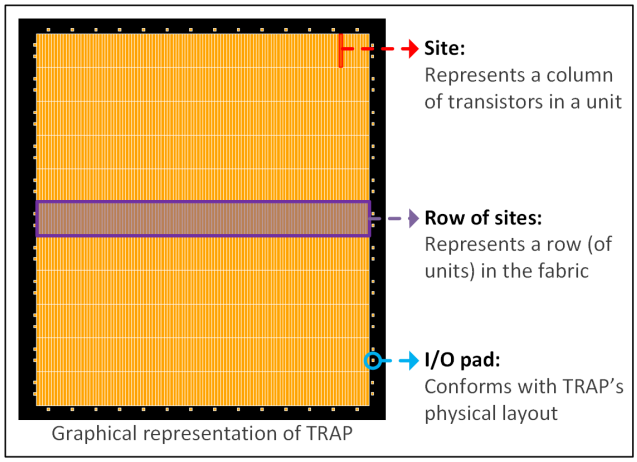


Fig. 4: TimberWolf-C is designed to capture the architecture of TRAP and execute the design placement task accordingly.

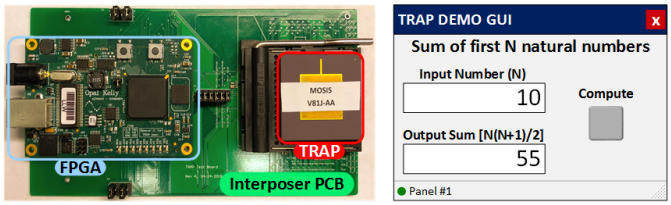
floorplanning used in ASIC designs. To achieve this, we leverage a version of the TimberWolf, which is a timing-driven placement and partitioning tool often leveraged by the CAD community for innovative research [9]. This tailored version for our CASPER framework (*TimberWolf-C*) undergoes the following customizations: (i) the dimensions of TimberWolf's *row* and *site* based grid structure is updated to render it compatible with TRAP, (ii) the internal characterization of cells and site categories is modified to ensure correct mapping and placement, as well as to verify that a target design will fit on the TRAP hardware, and (iii) TimberWolf's I/O pads placement policy is redefined to match with that of TRAP. Collectively, these modifications ensure that TimberWolf-C accurately captures the TRAP architecture, as shown in Figure 4, and performs design placement accordingly.

D. Placement and Interconnect Translation

CASPER leverages TimberWolf-C's placement information for routing a design with the Versatile Place and Route (VPR) tool. While TimberWolf generates a single output file (`.pl1`) that combines placement and interconnect information, VPR requires them in the form of two separate input files (`.p` and `.net`). We build the placement and interconnect translator for TRAP (PITT), a novel Python-based software tool, to translate the .pl1 file into corresponding .p and .net files.

Placement (.p) file generation: PITT first parses the .pl1 file to extract the name and pinlist for all the primary I/Os present in the design, and then copies that information in the .p file following the appropriate format. The tool then acquires the `clock` and `reset` nets for the sequential/state-holding cells from the synthesized netlist file. Next, it extracts the I/O pad placement information and prints it into .p file. Finally, PITT extracts the logic block (LE) information, translates it into VPR accepted format, and writes it into the .p file.

Interconnect (.net) file generation: The .net file contains the name and pinlist of all the blocks (LEs) used in the design, as well as the internal pinlists for the four sub-blocks inside each of those LEs. For an LE, the list of nets corresponds to all the pins connected to it and the net names used to make the connections are the same as they are in the Verilog file. For the sub-blocks, the nets correspond to the connection to each of the four transistor columns. Each transistor can get three values: an input, short (closed) or, open. Short (open) means the gate of the transistor is given a value that makes the transistor act as a closed (open) switch.



(a) (b)

Fig. 5: (a) Hardware testbed and (b) GUI design for validating CASPER with our prototype TRAP chip.

E. Design Routing

VPR is an open-source placement and routing software for FPGAs which takes the placement (.p) and interconnect (.net) information for a target design and generates the optimal routing result (.r) [10]. However, to render it compatible with TRAP's novel architecture, the CASPER version of VPR (VPR-C) must undergo the following customization: (i) the logic block (CLB) design is revised to make it structurally equivalent to the TRAP LEs, (ii) the programmable interconnect network is reconfigured to match with that of TRAP, and (iii) the tool is modified to replicate the I/O pad configuration of TRAP. This is critical because, unlike FPGAs, TRAP has a non-uniform distribution of I/O pads. Specifically, on the vertical sides (left and right) two pads are placed for each row of LEs, whereas on the horizontal sides (top and bottom) a single pad is shared between two columns of LEs.

F. Programming Bitstream Generation

The final step in CASPER is to generate the required bitstream for programming a target design on the TRAP fabric. Specifically, in addition to the standard tasks of programming the I/O pads, the interconnect network, and the logic blocks (LEs), the TRAP bitstream must also implement each logic cell used in the design. To this end, we develop the bitstream generator for TRAP (BGT) – a Python-based tool that leverages the design netlist, placement, and routing files to generate the required programming bitstream for the target TRAP chip. BGT performs two sub-tasks: (i) generating I/O programming (IOP) bits to program the I/O pads and connectivity with the fabric, and (ii) generating logic and interconnect programming (LIP) bits to program the actual fabric.

IV. Validation and Evaluation

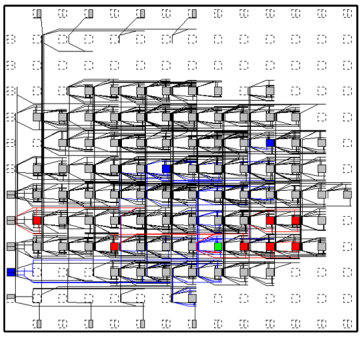
A. Silicon Prototype and Hardware Testbed

We conjecture that a complex CAD framework such as CASPER can be reliably verified only through hardware-based testing. Therefore, we fabricated a 3mmX3mm prototype TRAP chip in Global Foundries 65nm process (via MOSIS). We also developed an experimentation platform for the chip, which consists of a custom-designed interposer printed circuit board (PCB) and an Opal Kelly `XEM3010-1500P` FPGA kit (Xilinx Spartan 3) [13]. This hardware testbed for CASPER is shown in Figure 5(a). The interposer PCB houses the TRAP chip and facilitates communication with the FPGA board. The purpose of the FPGA board is to load the programming bitstream on the TRAP chip and provide required control and trigger signals. The FPGA also captures the TRAP chip's output at runtime for further analysis.

Authorized licensed use limited to: Univ of Texas at Dallas. Downloaded on August 22,2021 at 23:45:49 UTC from IEEE Xplore. Restrictions apply.

```
module sum_of_n_SLTRAP(n,clock,out);
input [7:0] n;
input clock;
output reg [7:0] out;
reg [7:0] np1;
always @(negedge clock)
  begin
    np1 = n + 1;
    out = ((n * np1)/2);
  end
endmodule
```

(a)



(b)

Fig. 6: `Sum of N natural numbers` program: (a) behavioral Verilog and (b) VPR-C's routing result.

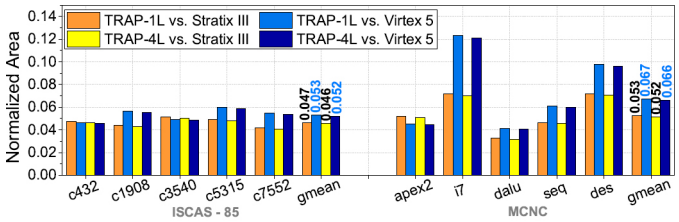
B. Validation Example

Figure 6(a) shows the behavioral RTL (Verilog) code for a "`sum of N numbers`" program that we implement in order to verify correct operation of CASPER. Specifically, the program calculates the sum of all natural numbers up to a user-provided input (N). The mathematical formula used in the program is: `S = N(N+1)/2`, where S is the calculated sum. We choose this program because, in spite of being a simple design, the program leverages three different mathematical operations, namely addition, multiplication, and division (right shift). This, in turn, renders the synthesized netlist to use a wide range of cells from the TRAP library and can allow us to test the fabric's capability to implement such cells. Furthermore, the small size of the design allows us to easily demonstrate its routing results, as shown in Figure 6(b). The generation of the routing (.r) files can confirm a successful implementation of CASPER for the most part. We then generate the bitstream for the circuit using our BGT tool and program the prototype TRAP chip. At this point, our custom GUI allows us to verify the correct operation of the programmed chip, which in turn confirms a successful implementation of CASPER. Figure 5(b) shows a sample case of the implemented program where a user input of N=10, generates the correct output of S=55.

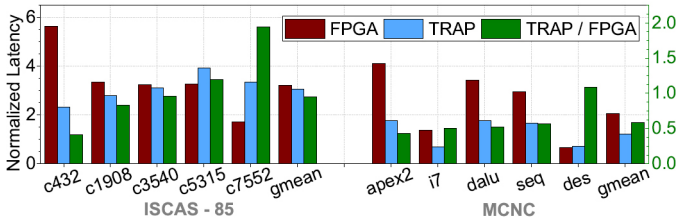
C. Comparative Evaluation against FPGAs

It is plausible to spontaneously compare TRAP with conventional FPGAs – as both are programmable fabrics. However, FPGAs typically require a separate tool-flow and lacks the seamless integration with ASIC. Nevertheless, the most critical difference between TRAP and FPGAs lies in their respective overhead over ASIC. To demonstrate this, we compared our 65nm TRAP with `Altera(Intel) Stratix III (EP3SL50F780C2)` and `Xilinx Virtex 5 (XC5VLX110)` – two of the prominent FPGAs from that technology node [14], [15]. As for benchmark design, we chose ten circuits – five each from the ISCAS-85 and the MCNC benchmark suites [11].

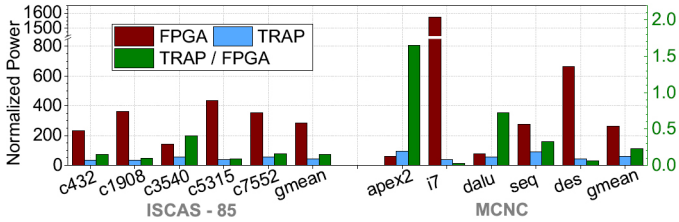
Area Overhead: In order to demonstrate TRAP's advantage in terms of design density, we compare a single-layer (`TRAP-1L`) and a four-layer (`TRAP-4L`) TRAP with the two FPGAs. As shown in Figure 7a, the difference between the two incarnations of TRAP becomes insignificant when compared to that with the FPGAs. In addtion, for both the ISCAS-85 and MCNC benchmarks, `TRAP-1L` and `TRAP-4L`'s average area overhead was 0.05X and 0.06X compared to `Stratix III` and `Virtex 5`, respectively.



(a) Area overhead.



(b) Latency overhead.



(c) Power overhead.

Fig. 7: TRAP vs. FPGA: Comparative overhead analysis.

Latency Overhead: It is often critical to implement designs with minimal latency degradation. In Figure 7b, compare `TRAP-1L` and `FPGA` (we used Virtex 5 as it has lower area overhead) in terms of latency normalized to ASIC. For the ISCAS-85, `TRAP` performs competitively with an average latency overhead of 1.15X compared to the `FPGA`. Furthermore, for the MCNC circuits, `TRAP` performs significantly better than the `FPGA` reducing the overhead to 0.54X.

Power Overhead: Efficient area and interconnect usage allow TRAP to cut down power consumption significantly. As shown in Figure 7c, for most of the benchmark circuits, `TRAP`'s power consumption overhead was closer to ASIC's, than to that of the `FPGA`. Specifically, for the ISCAS-85 and the MCNC benchmarks, `TRAP`'s power consumption overhead was 0.16X and compared to that of the `FPGA`.

Based on the above discussion, we can confirm that, for specific applications that benefit for on-die integration with an ASIC, TRAP is better-suited than conventional FPGAs.

V. Conclusion

While the recently proposed transistor-level programmable fabric has demonstrated a promising future, it remains unexplored due to incompatibility with existing CAD tools. In this work, we proposed a novel CAD framework for implementing designs on the TRAP fabric. Specifically, we characterized an ASIC-esque cell library to leverage the existing logic synthesis tools, modified the TimberWolf tool to facilitate TRAP-specific design mapping and placement, customized the VPR tool for design routing, and developed a Python-based programming bitstream generator tool for TRAP. Finally, we fabricated a prototype TRAP chip and built an FPGA-based validation testbed. Our experiments confirmed a successful implementation of the proposed framework.

Authorized licensed use limited to: Univ of Texas at Dallas. Downloaded on August 22,2021 at 23:45:49 UTC from IEEE Xplore. Restrictions apply.

REFERENCES

[1] W. Vanderbauwhede and K. Benkrid, *High-Performance Computing Using FPGAs*. Springer, 2013, vol. 3.
[2] P. Layzell, "A New Research Tool for Intrinsic Hardware Evolution," in *ICES*, 1998.
[3] A. Stoica, "Towards Evolvable Hardware Chips: Experiments with a Programmable Transistor Array," in *MicroNeuro*, 1999.
[4] J. Langeheine, K. Meier, and J. Schemmel, "Intrinsic evolution of analog electronic circuits using a cmos fpta chip," in *EUROGEN*, 2003.
[5] J. Tian, G. R. Reddy, J. Wang *et al.*, "A field programmable transistor array featuring single-cycle partial/full dynamic reconfiguration," in *DATE*, 2017.
[6] M. M. Shihab, J. Tian, G. R. Reddy *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] J. Luu, J. Goeders, M. Wainberg *et al.*, "VTR 7.0: Next Generation Architecture and CAD System for FPGAs," *ACM TRETS*, vol. 7-2, 2014.
[9] "TimberWolf Systems Inc." http://www.twolf.com/, 2019.
[10] V. Betz and J. Rose, "VPR: A New Packing, Placement and Routing Tool for FPGA Research," in *FPL*, 1997.
[11] D. Bryan, "The ISCAS'85 Benchmark Circuits and Netlist Format," *North Carolina State University*, vol. 25, p. 39, 1985.
[12] https://www.synopsys.com, 2019.
[13] "Opal Kelly XEM3010." [Online]. Available: https://opalkelly.com/products/xem3010/
[14] https://www.altera.com/, 2019.
[15] https://www.xilinx.com/, 2019.

Authorized licensed use limited to: Univ of Texas at Dallas. Downloaded on August 22,2021 at 23:45:49 UTC from IEEE Xplore. Restrictions apply.