Embedding Binary Perceptrons in FPGA to improve Area, Power and Performance

Ankit Wagle, Elham Azari, Sarma Vrudhula
School of Computing, Informatics and Decision Systems Engineering
Arizona State University, Tempe AZ 85281
{awagle1, eazari, vrudhula}@asu.edu

***Abstract*—For the flexibility of implementing any given Boolean function(s), the FPGA uses re-configurable building blocks called LUTs. The price for this reconfigurability is a large number of registers and multiplexers required to construct the FPGA. While researchers have been working on complex LUT structures to reduce the area and power for several years, most of these implementations come at the cost of performance penalty. This paper demonstrates simultaneous improvement in area, power, and performance in an FPGA by using special logic cells called Threshold Logic Cells (TLCs) (also known as binary perceptrons). The TLCs are capable of implementing a complex threshold function, which if implemented using conventional gates would require several levels of logic gates. The TLCs only require 7 SRAM cells and are significantly faster than the conventional LUTs. The implementation of the proposed FPGA architecture has been done using 28nm FDSOI standard cells and has been evaluated using ISCAS-85, ISCAS-89, and a few large industrial designs. Experiments demonstrate that the proposed architecture can be used to get an average reduction of 18.1% in configuration registers, 18.1% reduction in multiplexer count, 12.3% in Basic Logic Element (BLE) area, 16.3% in BLE power, 5.9% improvement in operating frequency, with a slight reduction in track count, routing area and routing power. The improvements are also demonstrated on the physically designed version of the architecture.**

***Index Terms*—Threshold Logic, FPGA, Reconfigurable, FD-SOI, 28nm, PNAND, Low Power, Low Area, High Performance, Digital Perceptron**

I. Introduction

Traditionally, the attractiveness of FPGAs was primarily due to their reconfigurability in the field, which provided the flexibility of software and the high performance and energy-efficiency of hardware, when compared with microprocessor-based systems. FPGAs are also ideal for rapid prototyping of designs [1]. However, when compared to ASICs, FPGAs are still an order of magnitude inferior in performance, power, and area (PPA) [2]. Nevertheless, the extremely high costs of ASICs, especially, in sub-16/14nm technologies [3] [4], combined with shrinking time-to-market [5], and the more recent drive to implement compute-intensive applications like CNNs/DNNs on FPGAs [6]–[9] have all placed added emphasis on improving the energy-efficiency of FPGAs.

Improvements in the PPA of FPGAs to date have been achieved through the use of new design mapping algorithms, and by modifying the architecture of the basic computational block of an FPGA – the *lookup table* (LUT). Advances in technology mapping [10], [11], have taken place over several decades, and have been incorporated into most of the commercial FPGAs [12]–[14]. Comparatively less has been done on exploring alternate architectures for the LUT.

The most common approach to examining alternate architectures for an LUT is based on varying the size of their support set and/or changing the set of functions each can realize. Ahmed et al. [15] suggest the use of LUT4[1] maximizes area efficiency whereas an LUT6 results in the best performance. Feng et al. [16] introduced a new LUT7 structure with reduced functionality, by hard-wiring two LUT4 circuits, to achieve a 9.5% reduction in the area with a 1.6% reduction in performance. Anderson et al. [17] describe an extended LUT5 structure that resulted in a 9% reduction in the area with a 5% performance penalty.

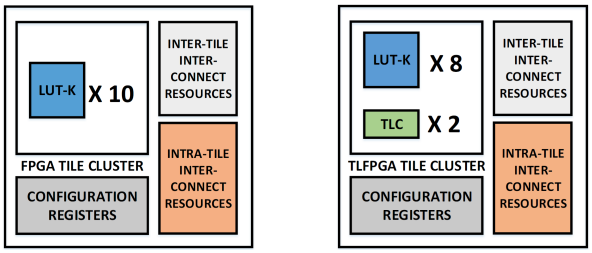
Research in new circuit architectures for LUTs and programmable interconnects employing emerging device technologies such as RRAMs, STT-MTJs and DWTs have also been reported [18]–[21]. Although these technologies are still under development and not yet viable commercially, existing literature provides compelling evidence that they have the potential for realizing ultra-compact and energy-efficient FPGAs.

In this paper, we describe a new architecture for an FPGA, referred to as *threshold logic FPGA* (TLFPGA), that integrates a conventional LUT with a CMOS digital implementation of a binary perceptron, also known as a threshold logic cell (TLC). Kulkarni et al. [22] have reported a design of a 2D array of similar cells, but their architecture, strictly speaking, is not a general purpose FPGA, as their array can only realize combinational netlists, and is practical only for netlists without feedforward signals. In contrast, the design described herein, combined with a new logic mapping algorithm that exploits the presence of both a conventional LUTs and TLCs within the basic logic element, achieves significant improvements in all the metrics of PPA.

The architecture of a TLFPGA favors highly pipelined circuits, i.e., the improvements in PPA increase with more pipeline stages – a characteristic that is not generally found when using conventional logic structures, where increasing performance is often achieved at the expense of area and/or power. To ensure an accurate evaluation of the proposed architecture, the FPGAs and TLFPGAs for different sizes of LUTs were designed down to the layout level, using the public domain FPGA design tool VPR [23], OpenFPGA [24], and

[1]LUTn refers to an LUT whose support set is n.

978-1-7281-2350-9/19/$31.00 ©2019 IEEE



(a) (b)

Fig. 1: Sample Tile structure for (a) FPGA and (b) TLFPGA. Two of the LUTs in (a) have been replaced with TLCs in (b).

commercial tools from Cadence Inc., including Genus® for synthesis and Innovus® for placement and routing [25]. All estimates for performance and power are based on simulation of netlists with all parasitics extracted from the layouts. The circuit benchmarks that were mapped on to the FPGAs and TLFPGAs include the traditional set of ISCAS-85 [26] as well as larger and more complex function blocks from OpenCores.

The paper is organized as follows: Section II describes the TLFPGA architecture and compares LUTn for $n = 4, 5, 6, 7$ with a TLC7 in terms of area, delay and power. Section III describes the design flow that results in a complete layout of the FPGA and TLFPGA. Sections IV and V describes logic mapping algorithm that looks for subcircuits which are threshold functions that can be implemented by a TLC7. Extensive experimental results using standard benchmark circuits and large complex function blocks are presented in Section VI. Section VII concludes this paper.

II. TLFPGA Architecture

Figure 1 illustrates the structure of a standard FPGA Tile and the threshold logic tile within a TLFPGA. All the 10 BLEs in an FPGA tile are LUTs, whereas the 10 BLEs in the TLFPGA tile consists of 8 LUTs and 2 TLCs. This arrangement will be justified shortly based on empirical evaluations of various configurations. Each tile has configuration registers and Muxes which are used to program the BLEs.

A TLC is a digital CMOS circuit implementation of a binary perceptron, whose function is referred to as a threshold function or a linearly separable function [27], [28]. A unate Boolean function $f(x_1, x_2, \cdots, x_n)$ is called a threshold function if there exist weights w_i for $i = 1, 2, \cdots, n$ and a threshold T[2] such that

$$f(x_1, x_2, \cdots x_n) = 1 \;\leftrightarrow\; \sum_{i=1}^{n} w_i x_i \geq T, \qquad (1)$$

where $\sum$ denotes the arithmetic sum. Thus a threshold function can be represented as $(\boldsymbol{W}, T) = [w_1, w_2, \cdots, w_n; T]$. An example of a simple threshold function is $f(a, b, c, d) = abc \vee abd$ with $[w_1, w_2, w_3, w_4; T] = [2, 2, 1, 1; 2]$. A more complex threshold function is $g(a, b, c, d, e) = ab \vee acd \vee bcd \vee ace \vee bce \vee ade \vee bde$, with $[w_1, w_2, w_3, w_4, x_5; T] = [2, 2, 1, 1, 1; 4]$. Although threshold functions can be implemented as a static CMOS logic circuits, implementations that are based on evaluating the defining predicate in Equation 1 by comparing some physical quantity such as charge or conductance can result in substantial reductions in gate count, area and power, as well as improving the performance when compared to standard logic implementations [28].

[2] W.L.O.G. the weights w_i and threshold T can be integers [27].

Operation of a TLC: Figure 2 depicts the transistor-level structure of the TLC. It consists of a sense amplifier, a latch, a left input network, and a right input network. The circuit operates in two phases. In the reset phase (CLK = 0), N5 and N6 are discharged, turning off all discharge paths from N1 and N2 to ground. The outputs N1 and N2 transition to 1 through M1 and M4.

Evaluation takes place when $CLK : 0 \rightarrow 1$. Assuming that the inputs have arrived, and that the left input network has higher conductivity than the right input network. In the evaluation phase, M13 and M14 are turned OFF, and both N5 and N6 will rise to 1. Without the loss of generality, assume as a result, node N5 rises before node N6, and turns M7 on. Prior to evaluation, N1 and N2 were both 1. Hence, M5 is active when M7 turns on. This discharges N1 through M5 and M7. The discharge of N1 stops the further discharge of N2 by turning off M6 and turning on M3. Consequently, the final values of the outputs are N1 = 0, N2 = 1, which resets the output latch. If the right input network had high conductivity, the result would have been N1 = 1, N2 = 0, which results in setting the latch. Note the feedback involving M9 and M10. These are strictly unnecessary but are included to ensure that once the clock transition completes, further changes on the inputs will not affect the outputs.

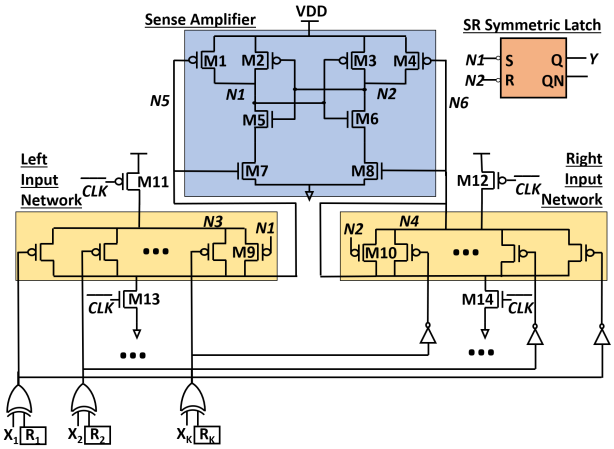


Fig. 2: Threshold Logic Cell (TLC) Structure. The sense amplifier detects the difference in conductivity of the left and right input networks and sets the outputs $N1$ and $N2$ accordingly.

The signals applied to the left and right input networks are complementary, to ensure that there will always be a difference of at least one active transistor between the two networks. A configuration register $R_i = 0$ if input X_i is to appear in

positive polarity, and $R_i = 1$ if X_i is to be complemented. The use of complementary signals in the two input networks ensures a strict inequality between the conductivity of the left and the right input networks and prevents the sense amplifier from being in a metastable condition.

For the TLC to properly realize a threshold function, the predicate shown in Equation 1 has to be converted to a strict inequality, and the variables in Equation 1 have to mapped to its inputs. Thus, Equation 1 is replaced with $\sum_{i=1}^{n} 2w_i x_i > 2T - 1$. As the signals driving the input networks are complementary, to realize this inequality the same number of literals representing a signal must appear in both networks. For example, consider $f(a,b,c) = a \vee bc \equiv 2a + b + c \geq 2 \equiv 4a + 2b + 2c > 3$. This is rewritten as $2a + b + c + 1 > 2(1-a) + (1-b) + (1-c)$. Therefore the signals assigned to left input network in Figure 2 would be $X_1 = a$, $X_2 = a$, $X_3 = b$, $X_4 = c$, $X_5 = 1$, $X_6 = 0$, and $X_7 = 0$, and $R_i = 1$, for $i = 1, 2, \cdots 7$.

Note that this particular signal assignment only requires the seven XORs and configuration registers to program the TLCs and does not require any additional control mechanism. Furthermore, signal replication does not affect the track count in the TLFPGA. The configuration registers of a TLC can be programmed alongside the configuration registers of other LUTs in the FPGA.

The benefits of the TLC have already been validated using 65nm Technology in Yang et al. [29]. A TLC-based Wallace multiplier ASIC was compared against a functionally equivalent, conventional standard cell implementation. Using both simulation and chip measurement results, a 33% improvement in dynamic power, 24% lower core area, 45% lower wirelength and 50% lower leakage was reported for a TLC-based Wallace multiplier, without any performance degradation. Significant improvements in other designs, using the TLC, have also been demonstrated in 65nm in Kulkarni et al. [30].

TLC vs LUT: Table I shows a comparison of a TLC7 with LUTs of various sizes. Both cells were implemented in 28nm FDSOI with power and delay estimates obtained from parasitic extracted netlists. At the individual cell level, a TLC has a substantially improved delay and power when compared to all standard size LUTs. In addition, the number of configuration registers and Muxes are also substantially reduced.

TABLE I: Delay and power for LUTs (with DFFs) and TLC. Compared to LUT-4, a TLC is 2X faster and 31% lower power.

BLE Type	Config. Regs	MUX/XOR	Delay (ps)	Power (μW)
LUT-4	16	15	220	33.2
LUT-5	32	31	226	64.0
LUT-6	64	63	294	125.0
LUT-7	128	127	331	248.0
TLC	**7**	**7**	**109**	**22.8**

A TLC can only realize a limited set of functions. This limitation can be overcome by proper logic absorption (see Figure 3). When a part of the logic cone feeding a flip-flop happens to be a threshold function that can be realized by a TLC7, that part along with the flipflop can be replaced by a single TLC7 cell. Threshold functions occur frequently in many practical circuits. Thus, a key step in mapping circuits to a TLFPGA is to identify threshold functions that are part of the logic cone feeding flipflops.

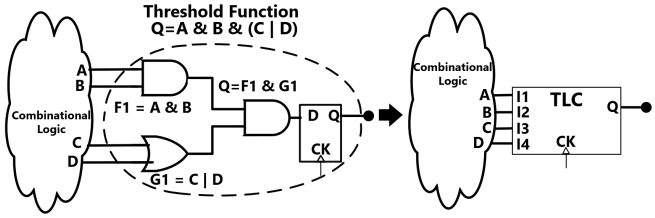


Fig. 3: Logic absorbtion. Part of the logic cone driving the DFF is a threshold function $abc \vee abd$. That logic and the DFF are replaced by a single TLC.

TLC Tile vs LUT Tile: A cluster of 8 LUTs and 2 TLCs was chosen for a TLFPGA tile because most of the logic cones in the benchmark circuits had threshold functions with support set of four variables. This means each feeder circuit output (see Figure 3) could be mapped to an LUT. Hence, clustering four LUTs with a single TLC would allow the routing between the LUTs and the inputs of the TLC to remain within a single tile, thereby reducing the inter-tile routing. This leads to an improvement in the overall performance of the circuit. Since a cluster size of 5 is too small, a cluster size of 10 BLEs was chosen. Figure 4 depicts a typical cluster.

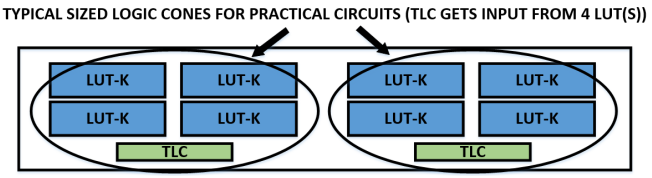


Fig. 4: Cluster size of 10 for TLFPGA; A TLC typically gets inputs from four variables. The number of LUTS in each TLC is set to four to ensure sufficient LUTs to feed the TLCs within a single tile.

TLFPGA tiles with 4, 5, 6, and 7-input LUTs were designed to examine the effect of LUT size on area, power, and performance. Note that a threshold logic tile contains fewer BLE configuration registers and BLE Muxes as compared to a standard LUT tile, and is therefore significantly smaller than a standard LUT tile. The area comparison between an FPGA and a TLFPGA tile is shown in Table II. At the individual cell level and at the tile level, the TLFPGA shows substantial improvement in the area. Later in Section VI, it is shown that the use of the TLFPGA architecture improves the overall PPA by reducing both the logic and routing resources during technology mapping.

III. Top Flow for TLFPGA

Figure 5 shows a modified version of the design steps in OpenFPGA [24] that is used to generate the bit-stream file for the TLFPGA. The main modifications made to this flow for designing with a TLFPGA are *threshold cell mapping (TCM)*

TABLE II: Tile area of FPGA vs. TLFPGA. Replacement of a large LUT with a small TLC helps shrink the tile size. NOTE: These numbers do not include the area of inter-tile routing resources, as they are subject to change based on the number of tiles.

K	LUT-K FPGA (μm^2)	LUT-K TLFPGA (μm^2)	Improvement (%)
4	192	176	8.3%
5	272	236	13.2%
6	428	353	17.5%
7	780	617	20.9%

algorithm and the *post-TCM improvements* which are shown in yellow boxes and explained in the following two sections.

The OpenFPGA's bit-stream generator was updated to support programming for TLCs. The FPGA generator was also modified to generate the Verilog output for TLFPGA. The input to this flow is a given behavioral description of a circuit. This netlist is synthesized to generate a gate-level netlist. The TCM is then performed on the flattened gate-level netlist (section IV) followed by LUT mapping using ABC [31]. The TCM is TLFPGA-specific and is not included in a conventional FPGA Flow [32]. The resulting circuit is passed to the VPR [23] tool (Versatile Packing, Placement and Routing) for placement and routing. The routing and placement information generated by VPR is used by the modified OpenFPGA to generate the final TLFPGA Verilog. Open-FPGA also uses this information to generate a bit-stream configuration file, required to program the TLFPGA.

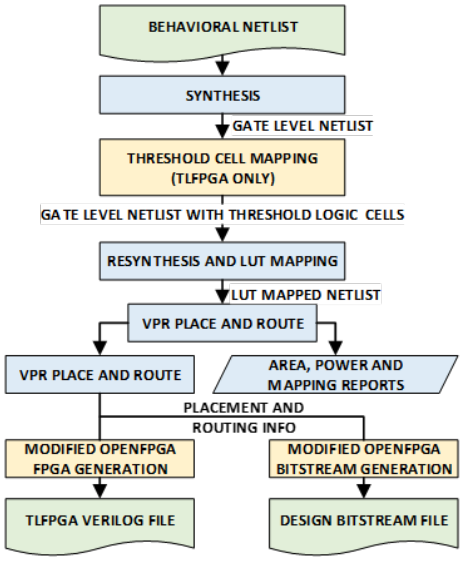


Fig. 5: Modified OpenFPGA Flow for TLFPGA

IV. Threshold Cell Mapping

This section describes the algorithm required to map a circuit to the TLFPGA architecture. The algorithm takes a synthesized netlist and replaces selected parts of the circuit with TLCs to create a netlist with a combination of TLCs and logic cells. Kulkarni et al. [30] describe an algorithm for incorporating threshold logic gates in ASICs but their algorithm is not applicable here due to the additional constraints imposed by the architecture of the TLFPGA. The distinction is similar to the distinction between technology mapping for ASICs versus FPGAs. In the following, we present a brief outline of the threshold cell mapping algorithm (TCM).

The key step in TCM, referred to as *TLextract*, is to identify a subcircuit whose function is threshold within a logic cone driving a given flip-flop. We employ a simple heuristic to perform *TLextract*.

First, the set of all threshold functions that can be realized by a TLC7 are enumerated based on the restrictions placed by the signal assignment method described in Sec. II.

There are 22 such functions with a support set of at most 5. This allows the logic cone starting from the input of the flip-flop to be searched for the cut size of at most five, extracted and then tested whether the function of the circuit is a threshold function among those that can be realized by a TLC7. Because the size is limited and the search is limited to unate subcircuits, exact testing by verifying the definition in Equation 1 is feasible and fast. Note that the *TLextract* is guaranteed to return a valid result. The trivial case would be a single logic gate that drives the input of the flip-flop.

Whether or not the threshold subcircuit is replaced by a TLC7 is determined by a *trial-mapping*. For a given flip-flop, this procedure creates two copies of the subcircuit containing the flip-flop and its input logic cone. In the first copy, trial-mapping replaces the flip-flop, and the cone that was returned by *TLextract* with a TLC7. The part of the circuit which was not replaced with a TLC7, is mapped to the LUTs. In the second copy, all the cells from the original netlist are mapped to the LUTs. These two alternatives are compared based on logic depth and LUT count.

TLextract can be performed independently on flip-flops whose logic cones do not intersect. Two flip-flops whose logic cones intersect are said to be dependent. This is viewed as a transitive relation, and therefore is an equivalence relation. Hence the flip-flops are easily partitioned into groups, each group consisting of dependent flip-flops and distinct groups being independent. Within a group, *TLextract* is performed iteratively, in the descending order of their timing criticality.

Given a TLFPGA architecture with a limited number of LUTs, TLCs and routing resources, the mapping algorithm performs the trial mapping against the following constraints to enhance the resource utilization of the TLFPGA:

Architecture based mapping constraint: The synthesis algorithm tries to maintain the ratio 4 LUTs to 1 TLC during TCM using this constraint. This constraint is applied to maximize the BLE utilization in the mapped tiles. Given a circuit with a requirement of N LUTs, if the TLC7 can be mapped to S flip-flops and if $S > N/4$, then this constraint allows only the top $N/4$ timing critical flip-flops to be mapped to TLC7(s) in the final circuit. By matching the TLC/LUT ratio of the *tile* with the *circuit's* ratio, the tiles can be packed with a higher density such that there are no unused BLEs in the mapped tiles.

Area and power reduction constraint: Reduction in the number of mapped LUTs reduces the required logic area and

power. If the number of mapped LUT-K(s) drop by X, when we map Y number of TLC(s), then the change in the number of configuration registers is given using the following equation:

$Config\_Reg\_New = Config\_Reg\_Old - X*(2^K) + Y*7$

In the example shown in Figure 6, the original LUT mapping requires $3 * (2^K)$ configuration registers, but after TCM, this requirement drops to $2 * (2^K) + 7$. This constraint allows the TCM to map the TLC7 to a flip-flop only if the mapping leads to a reduction in the number of LUTs.

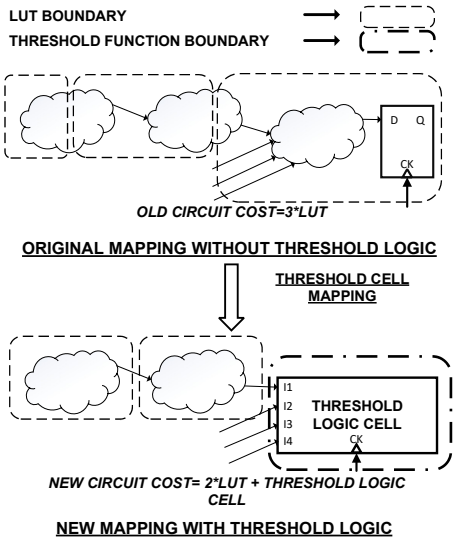


Fig. 6: Reduction in circuit implementation cost in a TLFPGA as compared to an FPGA. The cost of mapping the circuit to a TLFPGA is lower than the cost of mapping the same circuit to an FPGA.

Performance constraint: To ensure that the performance is not degraded after the TCM, the performance constraint allows flip-flops and their input logic cones to be replaced with TLCs only if it does not increase the logic depth.

Cell fanout constraint: Any cell in the given circuit, with an output net having a fanout of 2 or more needs to be mapped to the output of an LUT/TLC. Logic-replication helps remove this mapping requirement by dropping the cell's effective fanout to 1. However, logic-replication, when performed in excess can significantly increase the LUT count. Therefore, the cell fanout constraint allows the logic replication only for cells with a fanout of 2.

Complex cell constraint: A complex cell (e.g. Half-adder) can be decomposed into simpler threshold functions, but the decomposed structure might contain cells which have a fanout of 2 or more. To avoid generating unnecessary fanouts during TCM, the complex cell constraint blocks the decomposition of complex cells during TCM, so that they can be directly mapped to the LUTs.

V. Post-TCM Improvements

This section discusses how the results from TCM are improved after the initial set of mapping and placement information is collected.

Once VPR performs an initial placement of the design, a timing-based simulated annealing algorithm is used to rearrange the TLCs in the TLFPGA. TLCs on the timing-critical paths are given a higher priority for favorable placement locations on the TLFPGA.

Most heterogeneous FPGA architectures face the *congestion* problem. Over-use of a particular type of LUT/TLC leads to its scarcity. This makes it harder for the technology-mapper to find the necessary resources. This is an undesirable effect as it makes the design spread over a larger area. This problem is observed less in the TLFPGA due to two reasons. First, TLCs are small and require very less area overhead, in case the TLCs are not used. Second, the registers that the TLCs are mapped to, are chosen based on the placement congestion. In a TLFPGA region with scarcity of TLCs, there is a choice to map the threshold function directly to an existing LUT resource. This feature helps the placer find enough TLCs during placement without compromising on the placement locations. This is an extremely important feature as it significantly reduces the negative impacts of having a heterogeneous architecture. In typical heterogeneous FPGAs, under/over-use of a particular type of LUT/TLC leads to PPA penalties. For the TLFPGA, since the input count of an LUT and a TLC is similar, the mapping can be performed in a way such that the LUTs and the TLCs can be used interchangeably, without affecting the PPA much.

VI. Experimental Results

This section compares the design metrics of the TLFPGA with conventional FPGA architectures using VPR as well as the layout extracted netlists.

A. Benchmark Setup

The TLFPGA architecture was evaluated using ISCAS-85 and Open-Cores [33] benchmark circuits using the ST 28nm FDSOI Technology at $Slow/Slow$ $0.9V$ VDD $125°C$ simulation corner. In order to map the circuits to the TLCs, the benchmark circuits need to be sequential.

For studying the effects of uniform pipelining on the TLFPGA, registers were added to the ISCAS-85 circuits using re-timing. Ranging from a single pipeline stage to nine pipeline stages, the 10 combinational ISCAS-85 circuits were modified to generate a total of 90 sequential circuits. For the Open-Cores circuits, the number of pipeline stages was increased with the help of re-timing, until the logic depth of the critical paths reached a value between 8 and 12.

B. VPR Results for ISCAS-85 Circuits

The results extracted using VPR contain fully placed and routed circuits, based on the model parasitics of the FPGA and the TLFPGA. Models used for VPR are based on the tile structures placed and routed using Cadence Innovus®.

Figure 7 shows the percentage improvement in the BLE-parameters of a TLFPGA as compared to an FPGA. Improvements in technology-independent parameters, such as configuration registers and Muxes (Figures 7(a) and 7(b)) indicate that the TLFPGA can potentially offer benefits at other technology nodes as well. As the number of pipeline

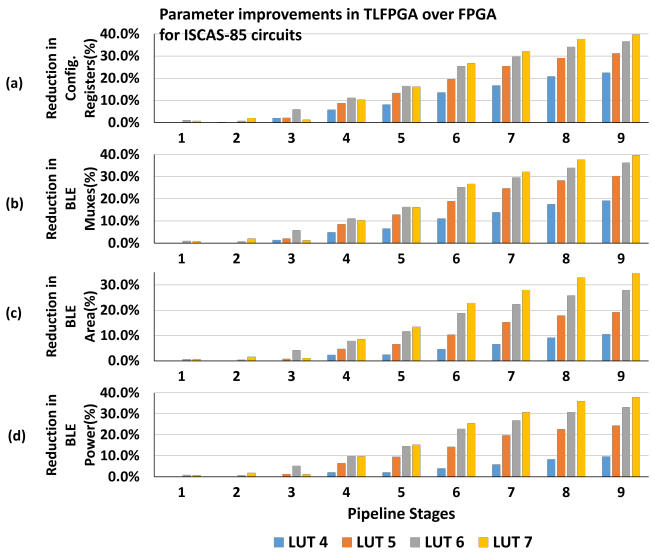


Fig. 7: Average percentage reduction in the number of (a) BLE configuration registers, (b) multiplexers, (c) area and (d) power in TLFPGA as compared to FPGA for ISCAS-85 circuits (higher is better).

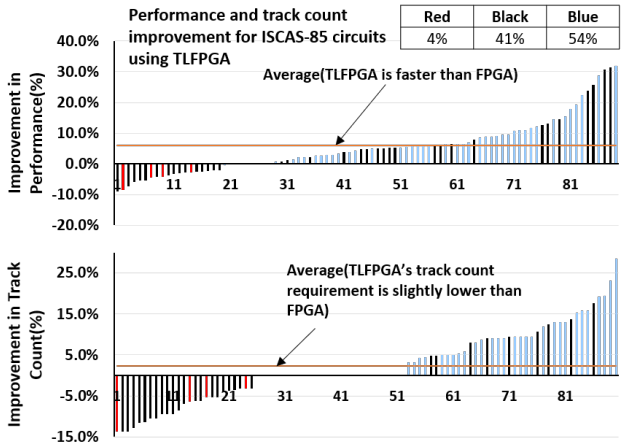


Fig. 8: Improvements in performance and track count for LUT-6 TLFPGA over LUT-6 FPGA for all circuits (higher is better). Circuits are arranged in the ascending order of their improvements. Color coding indicates improvement in:: Blue: Both performance and track count, Black: Either performance or track count, Red: Neither performance nor track count.

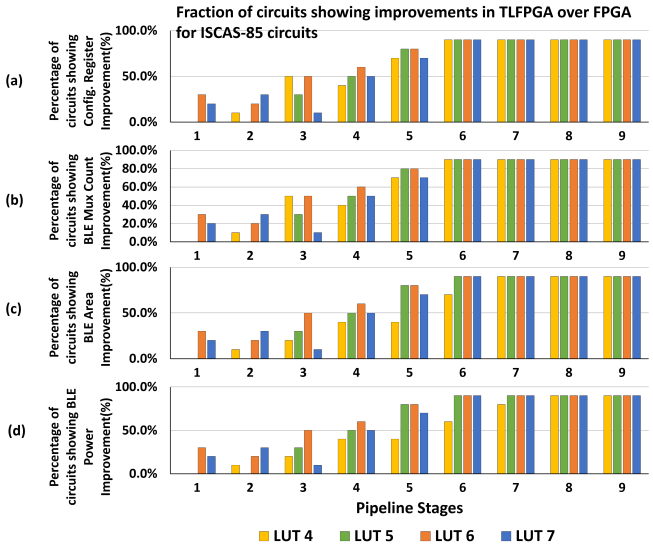


Fig. 9: Fraction of circuits that showed improvements in the number of (a) BLE configuration registers, (b) multiplexers, (c) area and (d) power in TLFPGA as compared to FPGA for ISCAS-85 circuits (higher is better).

stages increases, improvements in area and power of the BLEs used in the TLFPGA increase when compared to an FPGA. Increasing the number of stages increases the number of flip-flops, providing a greater opportunity for replacement with TLCs and logic absorption, and leading to greater improvements. These experiments clearly indicate that the TLFPGA gets the best results when used for deeply pipelined circuits.

Figure 7 also shows that for a fixed number of pipeline stages, the advantages in the BLE parameters in a TLFPGA increases with larger sizes of LUTs as compared to an FPGA. This is because there is an exponential rise in the number of configuration registers and Muxes as the LUT size increases, whereas the number of configuration registers and XORs in a TLC are unchanged. Hence, reducing the larger LUTs during technology mapping drastically drops the area and power requirements, resulting in significantly higher improvements when the TLFPGA is used.

Figure 8 shows the performance improvement as well as the improvement in the track count. Simultaneous improvement in performance and track count over the benchmark circuits is also shown in this figure. While 54% of the circuits demonstrated an improvement in both the performance and the track count, only 4% of the circuits had a degradation in both of these parameters. 41% of the circuits traded off one parameter for the other.

Figure 9 shows the count of ISCAS-85 circuits for which the BLE-parameters improved in TLFPGA as compared to FPGA. As the number of pipeline stages increases, it can be seen that the number of circuits benefiting from the TLFPGA architecture increases as well. As the number of pipeline stages increases, the number of flip-flops in the design increases, which creates more opportunities to map the TLCs, that eventually leads to better parameter improvements.

Figure 10 also shows that the routing area and power slightly improve when TLFPGA is used. This is because a reduction in the total number of BLEs reduces the routing requirements.

C. VPR Results for OpenCores Circuits

The ISCAS-85 circuits were chosen for the experiments because it is easy to add uniform pipeline stages to these circuits. However, the ISCAS-85 circuits may not best represent the industrial designs presently available. Therefore, for further evaluation of the TLFPGA architecture, the benchmark circuits from OpenCores were used. These circuits were deeply-pipelined to boost the improvements gained from using the TLFPGA. The typical logic depth for these circuits is 8-

TABLE III: Improvements in TLFPGA as compared to standard FPGA for OpenCores circuits for LUT-6 50x50 TLFPGA; Results are extracted using VPR. Models used for VPR are based on the tile structures placed and routed using Cadence Innovus®.

Circuit	LUT	Config. registers	Mux Count	Area	Power	Critical Path	Freq	Track count
64 bit Adder	18.1%	15.2%	15.1%	9.8%	13.1%	8.3%	9.0%	0.0%
64 bit Comparator	3.5%	3.0%	3.0%	2.1%	2.7%	15.2%	17.9%	0.0%
32 bit Multiplier	44.9%	40.1%	40.0%	31.5%	36.7%	22.9%	29.7%	0.0%
Hartley FFT	12.1%	10.9%	10.9%	8.7%	10.0%	7.8%	8.5%	0.0%
Mod-3 Calculator	12.7%	11.2%	11.2%	8.5%	10.1%	10.9%	12.3%	0.0%
16 bit Divider	20.6%	18.3%	18.2%	14.0%	16.6%	4.4%	4.6%	0.0%
32 bit Filter	5.6%	4.8%	4.7%	3.3%	4.2%	17.3%	21.0%	3.1%

TABLE IV: Comparison of power (mW) for the physical layout of 8x8 LUT-7 FPGA vs. LUT-7 TLFPGA with a track-width of 96 in X and Y directions. Using TCM, additional power reduction is expected. However, this is not included during the static power analysis.

	FPGA				TLFPGA				Drop %			
	Internal	Switching	Leak	Total	Internal	Switching	Leak	Total	Internal	Switching	Leak	Total
CLB	1.72	5.47	0.75	7.94	1.32	4.73	0.62	6.67	23.4%	13.4%	17.9%	16.0%
Switchbox	1.12	3.56	0.50	5.19	1.08	3.44	0.50	5.02	3.9%	3.5%	0.1%	3.3%
Crossbar	5.00	2.79	2.38	10.17	4.99	2.68	2.38	10.06	0.1%	3.9%	0.0%	1.1%
IO Config	0.02	0.06	0.01	0.10	0.02	0.06	0.01	0.10	3.7%	-2.5%	-5.6%	-1.4%
Total	7.87	11.88	3.65	23.41	7.42	10.92	3.52	21.85	5.7%	8.1%	3.7%	6.6%

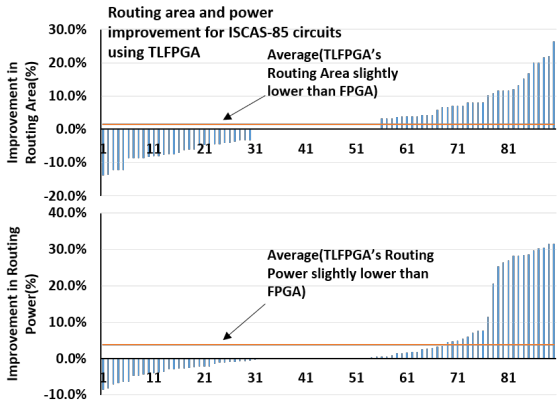


Fig. 10: Improvements in the routing area and power for LUT-6 TLFPGA over LUT-6 FPGA for all circuits (higher is better). Circuits are arranged in the ascending order of their improvements.

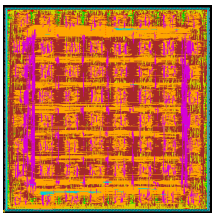
12 logic gates. Table III shows the improvements in various parameters for the OpenCores circuits.

TABLE V: Comparison of area and power of a tile in 8x8 LUT-7 FPGA vs. a tile in LUT-7 TLFPGA tile with a track-width of 96 in X and Y directions. Power is reported using static power analysis. Physical layout of tiles include the cells required for intra-tile routing.

Parameter	FPGA Tile	TLFPGA Tile	% Reduction
Area	$1054.6\mu m^2$	$870.5\mu m^2$	17.5%
Internal power	13.7uW	11.23uW	18.0%
Switching power	8.821uW	7.418uW	15.9%
Leakage power	10.43uW	8.701uW	16.6%
Total power	32.95uW	27.35uW	17.0%

D. Results from fully extracted layouts with parasitics

The results in this subsection are generated using the parasitic-extracted netlists from Cadence Innovus®. With a fixed track count of 96 and a cluster size of 10, the physical layouts of an 8x8 FPGA and TLFPGA (see Figure 11a) were generated. The synthesis and layout of the required blocks were done using Cadence Genus®and Innovus®.



(a)

Design	FPGA Power (mW)	TLFPGA Power (mW)	Improve (%)
C432	5.9	5.01	15.1%
C499	6.4	5.63	11.9%
C880	15.6	13.7	12.18%

(b)

Fig. 11: (a) Physical layout of an 8x8 prototype TLFPA generated using Cadence Innovus®; Verilog generated using modified OpenFPGA flow (b) Dynamic power reduction in small designs, when mapped to the physical post-layout version of FPGA and TLFPGA.

Tables IV and V show the power and area comparison of a LUT-7 FPGA tile versus LUT-7 TLFPGA tile. These tables report the area and power numbers of the full tile, which includes inter-tile and intra-tile routing resources as well. It can be seen that both the area and power of the TLFPGA tile are lower than the FPGA tile.

Tables VI and VII show the instance count and area comparison of a LUT-7 FPGA tile versus LUT-7 TLFPGA tile. The area of the physical layout is dependent on both the cell type and the drive strengths of the cells. For evaluation purposes, the combinational loops that arise from the FPGA's and TLFPGA's routing paths are set to false paths and the place-and-route tool adds minimum drive-strength buffers against signal slew-rate constraints. Although this method is not ideal for a real FPGA design, it lets us generate and evaluate the FPGA architectures using a fully automated flow very quickly (Kim et al. [34]). Due to the lack of accurate timing constraints on the inter-tile timing in the OpenFPGA flow, we limit the size of the TLFPGA to 8x8, to avoid major timing violations. The area of the FPGA and TLFPGA are almost the same at the end of the place-and-route flow, due to the lack of timing constraints for the global routing blocks, which makes it harder for the tools to set the drive strength of all the gates. Based on the static power analysis, a power

drop is observed in the TLFPGA as compared to the FPGA. This is because the components in the tiles used in a TLFPGA consume lower power, as compared to the components of a standard FPGA. Additional power drops come from the TCM algorithm described in Sec. IV. These results are not meant to show the final area and power results, but instead to show the potential improvements that we can gain from using the TLFPGA architecture in a real environment. Depending on the implementation, the overall benefits are subject to change. For example, if the routing architecture is implemented differently, then the overall PPA improvements will change.

TABLE VI: Comparison of instance count for 8x8 LUT-7 FPGA vs. LUT-7 TLFPGA physical layout with a track-width of 96 in X and Y directions.

	FPGA		TLFPGA		
	Count	Weight	Count	Weight	Drop
CLB	51968	15.5%	40576	12.5%	21.9%
IO Config	2688	0.8%	2688	0.8%	0.0%
Crossbar	216896	64.5%	216896	66.8%	0.0%
Switchbox	64507	19.2%	64478	19.9%	0.0%
Total	336059	100.0%	324638	100.0%	3.4%

TABLE VII: Comparison of area for 8x8 LUT-7 FPGA vs. LUT-7 TLFPGA physical layout with a track-width of 96 in X and Y directions.

	FPGA		TLFPGA		
	Area (μm^2)	Weight	Area (μm^2)	Weight	Drop
CLB	66658	17.2%	53728	14.3%	19.4%
IO Config	9651	2.5%	9651	2.6%	0.0%
Crossbar	253620	65.4%	253620	67.6%	0.0%
Switchbox	57985	14.9%	57966	15.5%	0.0%
Total	387916	100.0%	374966	100.0%	3.3%

Post-PnR Verilog of a small 8x8 FPGA and TLFPGA is used to map a few small circuits and compare the differences in the various parameters discussed above. Using a 300MHz clock frequency, Table 11b demonstrates that all the three test circuits show an improvement in overall dynamic power.

VII. Conclusion

This paper describes a new architecture for an FPGA, referred to as *threshold logic FPGA* (TLFPGA), that integrates a conventional LUT with a CMOS digital implementation of a binary perceptron, also known as a threshold logic cell (TLC). We show that by using the TLFPGA architecture, especially for deeply pipelined circuits, it is possible to gain significant area, power and performance improvements, by strategically mapping the circuits to the TLCs. Evaluation was done using the ISCAS circuits, as well as larger benchmarks from Open-Cores. The benefits of using the TLFPGA are also demonstrated, by mapping circuits to the physically designed versions of an FPGA and a TLFPGA. Since the TLCs are binary perceptrons, this architecture is well suited for Binary and Quantized Neural Networks.

VIII. Acknowledgment

We gratefully acknowledge the National Science Foundation for supporting this work under NSF PFI award no. #1701241.

References

[1] Synopsys. https://www.synopsys.com/implementation-and-signoff/fpga-based-design/fpga-based-prototyping.html.
[2] I. Kuon and J. Rose. Measuring the Gap Between FPGAs and ASICs. *IEEE TCAD*, Feb 2007.
[3] Taiwan Semiconductor Manufacturing Company(TSMC). https://www.tsmc.com/english/dedicatedfoundry/technology/16nm.htm.
[4] GlobalFoundries(GF). https://www.globalfoundries.com/sites/default/files/product-briefs/product-brief-14lpp-14nm-finfet-technology.pdf.
[5] Einfochips. https://www.einfochips.com.
[6] Y. Zhou and J. Jiang. An FPGA-based accelerator implementation for deep convolutional neural networks. In *IEEE ICCSNT*, Dec 2015.
[7] N. Raspa and G. Natale and M. Bacis and M. D. Santambrogio. A framework with cloud integration for cnn acceleration on fpga devices. In *IEEE IPDPSW*, May 2018.
[8] Y. Ma, Y. Cao, S. Vrudhula, and J. Seo. An automatic RTL compiler for high-throughput FPGA implementation of diverse deep convolutional neural networks. In *IEEE FPL*, Sep. 2017.
[9] Y. Ma, Y. Cao, S. Vrudhula, and J. Seo. Optimizing the Convolution Operation to Accelerate Deep Neural Networks on FPGA. *IEEE TVLSI*, July 2018.
[10] J. Cong and Y. Ding. FlowMap: an optimal technology mapping algorithm for delay optimization in lookup-table based FPGA designs. *IEEE TCAD*, Jan 1994.
[11] L. Amarú, P. Gaillardon, and G. De Micheli. Majority-inverter graph: A new paradigm for logic optimization. *IEEE TCAD*, May 2016.
[12] Xilinx 7 Series FPGAs Configurable Logic Block User Guide.
[13] Cyclone V Device Overview.
[14] Polarfire FPGA Fabric User Guide.
[15] E. Ahmed and J. Rose. The effect of LUT and cluster size on deep-submicron FPGA performance and density. *IEEE TVLSI*, March 2004.
[16] W. Feng, J. Greene, and A. Mishchenko. Improving FPGA Performance with a S44 LUT Structure. In *ACM/SIGDA ISFPGA*. ACM, 2018.
[17] J. Anderson, Q. Wang, and C. Ravishankar. Raising FPGA Logic Density Through Synthesis-Inspired Architecture. *IEEE TVLSI*, March 2012.
[18] X. Tang, G. Kim, P. Gaillardon, and G. De Micheli. A Study on the Programming Structures for RRAM-Based FPGA Architectures. *IEEE Transactions on Circuits and Systems I: Regular Papers*, April 2016.
[19] T. N. Kumar, H. A. F. Almurib, and F. Lombardi. A novel design of a memristor-based look-up table (LUT) for FPGA. In *IEEE APCCAS*, Nov 2014.
[20] S. Williams and M. Lin. Architecture and Circuit Design of an All-Spintronic FPGA. In *ACM/SIGDA FPGA*, 2018.
[21] N. S. Nukala, N. Kulkarni, and S. Vrudhula. Spintronic Threshold Logic Array (STLA) - a compact, low leakage, non-volatile gate array architecture. In *IEEE/ACM NANOARCH*, July 2012.
[22] N. Kulkarni, J. Yang, and S. Vrudhula. A fast, energy efficient, field programmable threshold-logic array. In *IEEE FPT*, Dec 2014.
[23] V. Betz and J. Rose. VPR: A New Packing, Placement and Routing Tool for FPGA Research. In *IEEE FPL*. Springer-Verlag, 1997.
[24] H. Liu. Archipelago - An Open Source FPGA with Toolflow Support. In *Thesis*. UC Berkeley, 2014.
[25] Cadence. http://www.cadence.com.
[26] F. Brglez and H. Fujiwara. A neutral netlist of 10 combinational circuits and a target translator in fortran. 06 1985.
[27] S. Muroga. *Threshold Logic and its Applications*. 1971.
[28] K. Siu, V. Roychowdhury, and T. Kailath. *Discrete Neural Computation: A Theoretical Foundation*. Prentice-Hall, Inc., 1995.
[29] Jinghua Yang, Joseph Davis, Niranjan Kulkarni, Jae sun Seo, and Sarma Vrudhula. Dynamic and Leakage Power Reduction of ASICs Using Configurable Threshold Logic Gates. In *Proc. IEEE Custom Integrated Circuits Conf. (CICC)*, San Jose, CA, Sept. 2015.
[30] N. Kulkarni, J. Yang, J. S. Seo, and S. Vrudhula. Reducing Power, Leakage, and Area of Standard-Cell ASICs Using Threshold Logic Flip-Flops. *IEEE TVLSI*, 24(9), Sept 2016.
[31] R. Brayton and A. Mishchenko. ABC: An Academic Industrial-strength Verification Tool. In *IEEE CAV*. Springer-Verlag, 2010.
[32] U. Farooq, Z. Marrakchi, and H. Mehrez. *FPGA Architectures: An Overview*. Springer New York, 2012.
[33] OpenCores. https://opencores.org/.
[34] J. Kim and J. H. Anderson. Synthesizable fpga fabrics targetable by the verilog-to-routing (vtr) cad flow. In *IEEE FPL*, Sep. 2015.