FPGAs with Reconfigurable Threshold Logic Gates for Improved Performance, Power and Area

Ankit Wagle†, Jinghua Yang†, Aykut Dengi, Sarma Vrudhula†
School of Computing, Informatics and Decision Systems Engineering
Arizona State University Tempe AZ 85281

***Abstract*—This paper proposes an alternative FPGA tile structure that consists of three traditional LUTs combined with a new reconfigurable threshold logic cell (TLC). The TLC requires only 7 SRAM cells and can be configured to implement one of several threshold functions. The proposed architecture is implemented in a 28nm FDSOI process, and is evaluated on standard benchmark circuits and several large complex function blocks. The results demonstrate an average reduction of 8.9% in register count, 15.4% in multiplexer count, 7% average reduction in Basic Logic Element (BLE) area, and 8.2% average reduction in BLE power, with a maximum decrease in register count up to 64%, BLE multiplexer count up to 68%, BLE Area up to 51.6% and BLE power up to 61.6% without loss in performance. We also show a reduction of 21% in the area of a tile.**

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

I. Introduction

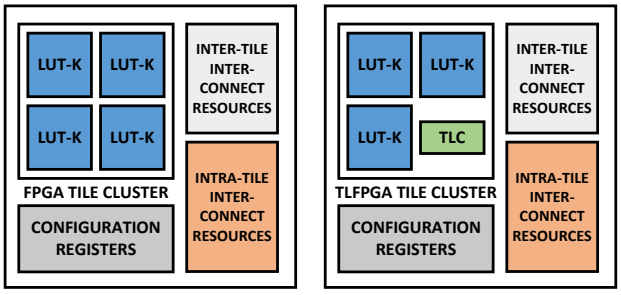
Implementation of LUTs in an FPGA comes at the heavy cost of area, power and performance. In this paper, a new FPGA named Threshold Logic FPGA (TLFPGA) is proposed. It uses a novel tile architecture (shown in Fig. 1b) consisting of a cluster of LUTs and Threshold Logic Cells (TLCs). Both LUTs and TLCs are BLEs capable of implementing Boolean functions. Compared to a standard LUT cell, the proposed TLC has smaller overall delay, area and power consumption when implementing the same function. A TLFPGA implementation shows significant improvement in area and power without sacrificing performance, as compared to a standard FPGA implementation.

The main contributions of this paper are as follows:

1) A new FPGA architecture called TLFPGA is proposed which integrates a TLC with a conventional FPGA Tile.
2) Based on the TLFPGA performance, a heuristic algorithm is proposed to perform the Threshold Logic mapping. As a result, a near optimal mapping solution with minimum number of tiles is achieved.
3) The effects of pipelining stages on the area, delay, and power are studied after mapping to the proposed architecture.

The paper is organized as follows: Section II gives a brief explanation about Threshold Logic and sequential Threshold Logic Element. Section III describes the basic architecture required to build an FPGA and the proposed TLFPGA Architecture. Section IV describes the design flow and the heuristic algorithm needed to perform the threshold logic based mapping. Section V shows experimental results of mapping, and the corresponding reduction in Basic Logic Elements (BLE) area and power. Section VI concludes the paper.

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



(a) LUT-K FPGA Tile (b) LUT-K TLFPGA Tile

Fig. 1: Tile structure for a) FPGA Tile and b) TLFPGA Tile. For TLFPGA tile, one LUT is replaced with one TLC.

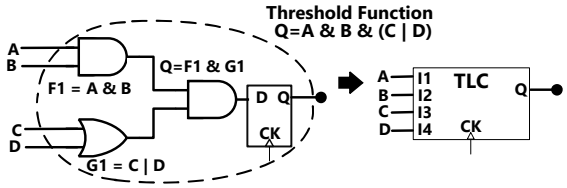


Fig. 2: Example of Threshold Cell Mapping

II. Background

A threshold function is a Boolean function that can be described by a predicate involving a weighted linear form of binary variables. A Boolean function $f(x_1,x_2,...,x_n)$ is a threshold function of there exists weights $w_1,w_2,...,w_n$ and a threshold T such that

$$f(x_1, x_2, \cdots, x_n) = \begin{cases} 1 & \text{if } \sum_{i=1}^{n} w_i x_i \geq T \\ 0 & \text{otherwise.} \end{cases} \quad (1)$$

978-1-5386-8517-4/18/$31.00 ©2018 IEEE
DOI 10.1109/FPL.2018.00051

TABLE I: Delay and Power for LUTs (without flip-flops) and TLC

BLE Type	Config Regs.	Config MUXes /XORs	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

TABLE II: Tile Area of FPGA vs. TLFPGA. Replacement of a large LUT with a small TLC helps shrink the tile size.

K	LUT-K FPGA (μm^2)	LUT-K TLFPGA (μm^2)
4	192	176
5	272	236
6	428	353
7	780	617

Threshold functions have been known to reduce the gate count, area, power and increase the performance of a standard Boolean logic implementation [1] [2]. The differential threshold gate, presented in [2] is used for implementing threshold logic functions. These gates have bounded fan-in.

III. Threshold Logic Cell

The circuit design of a TLC is described in Ref. [3], which can be viewed as a multi-input edge triggered flip-flop. The output of a D flip-flop is $f(x)=x$ on a clock edge, whereas the output of a TLC is a threshold function $f(x_1,x_2,...,x_n)$, on a clock edge (See Fig. 2). Note that a TLC can realize a set of Boolean functions, determined by what signals are assigned to the inputs [2]. Table I provides delay and power numbers for LUTs of various input sizes and a TLC in a 28nm FDSOI Process. TLCs are smaller and faster, and consume lower power than LUTs. The structures of a standard FPGA tile and the proposed TLFPGA tile are shown in Fig. 1. For studying the effects of LUT size on area, power, and performance, TLFPGA tiles with 4, 5, 6 and 7-input LUTs are used. Table II shows the area requirement of a standard FPGA tile versus a TLFPGA tile. In the table, LUT-K type FPGA tile consists of 4 LUT-Ks and LUT-K Type TLFPGA tile consists of 3 LUT-Ks and one TLC. Tiles are implemented using 28nm FDSOI Technology Cells using a standard ASIC flow. TLFPGA tile requires significantly smaller area than a standard FPGA tile, as there are fewer BLE configuration registers and BLE MUXes in a TLC than an LUT. Signals are assigned to the TLC using a signal assignment procedure described in [3], which helps us attain maximum speed, and makes the inputs of the Threshold Cells positionally equivalent. The programming for the TLCs in the design can be done while programming LUTs. Since an LUT has more configuration bits than a TLC, the overall time required to program the BLE(s) can be reduced.

IV. Design Flow for TLFPGA

The flow shown in Fig. 3 is used to generate the final architecture and bitstream file for the TLFPGA. This flow is a modified version of the OpenFPGA [4] flow. Threshold logic

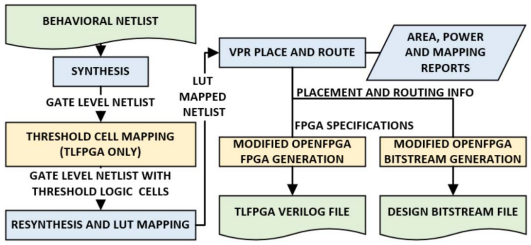


Fig. 3: Modified Design Flow for TLFPGA

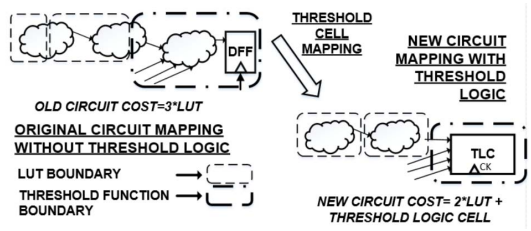


Fig. 4: Reduction in Circuit Implementation Cost in TLFPGA as compared to FPGA

technology mapping is done on a gate level netlist followed by LUT mapping using ABC [5]. The former is not included in a conventional FPGA Flow [6]. The bitstream generator is modified to support bit programming for TLCs.

Threshold Cell Mapping (TCM) in Fig. 3 is implemented by applying **Algorithm 1** on a gate level netlist. TCM replaces selected flip-flops and part of the input logic cone that feeds those flip-flops with threshold cells. This process is called logic absorption. A threshold cell is able to absorb a limited number of logic cells depending on the structure of input logic cone.

Fig. 4 illustrates how the LUT count is reduced by TCM. The cone of logic feeding each D flip-flop is searched for an appropriate threshold function, and the corresponding logic and the flop-flop are replaced by a single TLC. The original LUT mapping in the example requires 3 LUTs to implement the subcircuit shown. After performing TCM, 2 LUTs and a TLC is required. Since the cost of implementing a TLC is significantly lower than implementing an LUT, the area and power is reduced. Algorithm 1 is used to perform the TCM for the entire circuit. When the Algorithm 1 completes, we get an enhanced netlist with TLCs mapped to the circuit. The result of the TCM algorithm is a modified netlist with the same D flip-flops and parts of their fanin cone replaced by a TLC.

The threshold function mapping for a particular register FF_i is based on the logic cells at the fanin cone of FF_i. Algorithm 1 performs the mapping at the register location FF_i by performing cuts which ensure that the function at $TLG_i \in$ TLC's function set. Each ILC_i can be transformed into a group of logic cells feeding a linear threshold function TLG_i(See Algorithm 1 line 2). The new graph is called ILC_Th_i. This representation is generated by absorbing the maximum number of logic gates feeding FF_i into the threshold function at FF_i.

Algorithm 1 Pseudo Code for Threshold Cell Mappin

- **G**: Directed cyclic graph representation of the original netlist containing nodes for N flip-flops and Logic Gates.
- $\mathbf{G_{Th}}$: Directed cyclic graph; Original netlist containing nodes for flip-flops, Simple Logic Gates and Threshold Logic Gates
- $\mathbf{FF_i}$: i^{th} Flip-Flop in the circuit
- $\mathbf{ILC_i}$: Input logic cone of FF_i
- $\mathbf{TLG_i}$: Threshold function mapped at Flip-flop FF_i for Maximum Logic Absorption
- $\mathbf{ILC\_Th_i}$: Input logic cone of threshold function at FF_i
- $\mathbf{G_i}$: Circuit representation with the threshold logic transformation till FF_i
- $\mathbf{N\_LUT}_i$: LUT Count required to implement circuit with threshold logic transformation till FF_i

Input: $G; G = G_0$
Output: G_{Th}

```
for FF_i in FF_set; i=1 to N, do
    TLG_i = Best cut at ILC_i for maximum logic absorption, with following fanout constraints:
        1) Fanout = 1: Consider logic cell for cut
        2) Fanout = 2: Replicate logic cell and consider that cell for cut
        3) Fanout ≥ 3: Do not absorb these logic cells in the cut
    G_i = TLG_i applied to G_{i-1}
    Perform low effort trial LUT mapping on G_i to get N_LUT_i
    if N_LUT_i ≤ N_LUT_{i-1} then
        G_i = G_i
    else
        G_i = G_{i-1}
    end if
end for
G_Th = G_i
return G_Th
```

For a logic cell to be eliminated, it has to be absorbed by the threshold functions at its fanouts. Logic replication [7] is also supported by the algorithm. Each time we replicate logic for absorption, we reduce the fanout of the original logic cell by 1. If the logic cell fanout goes to 0, then that logic cell can be eliminated. LUT mapping is more efficient if the logic cell fanout is 1.

Based on experiments, it has been observed that logic cells with fanout greater than 2 rarely get eliminated. To reduce the complexity of the algorithm, logic cells with fanout greater than 2 are excluded from threshold logic cuts. Also, decomposition of complex non-threshold logic cells to threshold logic cells leads to inefficient LUT mapping, and is therefore avoided. Decomposition of complex function cells into a group of threshold functions may sometimes lead to the addition of signals with a fanout of 2, or more. Addition of such signals leads to an inefficient TCM for the decomposed threshold functions.

The tile structure shown in Figure 1 consists of one TLC and three conventional LUTs. This is because commonly used cluster sizes start at 4 BLE(s) per tile [8]. However, there are no restrictions on the BLE clustering. In LUT mapping, the output of conventional LUTs can be both combinational and sequential, whereas TLC output can only be sequential. As a result, more LUTs are required than TLCs. Therefore, the proposed structure was chosen to maximize the tile-utilization and the inter-tile routing. For optimal tile utilization, this algorithm can be easily modified to generate the desired LUT to TLG ratio.

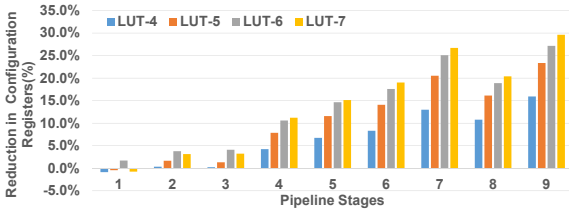


Fig. 5: Average reduction in BLE configuration register count in TLFPGA as compared to FPGA for varying LUT sizes and pipeline stages

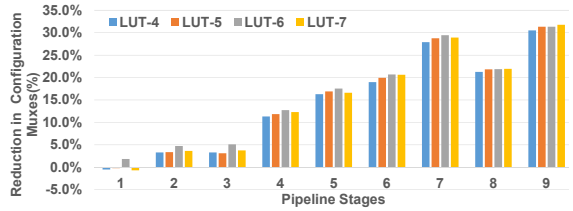


Fig. 6: Average reduction in BLE multiplexers count in TLFPGA as compared to FPGA for varying LUT sizes and pipeline stages

V. Experimental Results

For evaluating the TLFPGA architecture, ISCAS-85 and ISCAS-89 benchmark circuits are used. TCM is relevant only to sequential circuits, as the TLC is a flip-flop with embedded logic. Since ISCAS-85 circuits are strictly combinational circuits, we added additional pipeline stages by retiming the circuits. For studying the effect of pipeline stages on the TCM, 1 to 9 pipeline stages were added to all the 10 ISCAS-85 benchmark circuits, which gives us 90 circuits. The pipeline stages in ISCAS-89 circuits are not changed. The total number of circuits used for TLFPGA evaluation is 115 circuits. The logic depth of the test circuits vary between 2 to 12. Simulation corner is $Slow/Slow$ $0.9V$ VDD $125°C$.

For analyzing the effect of pipelining on various parameters, pipelined ISCAS-85 benchmark circuits are used. Fig. 5, 6, 7 and 8 show the percentage reduction in the BLE Configuration Registers (Config Regs), BLE MUXes (MUX Count),

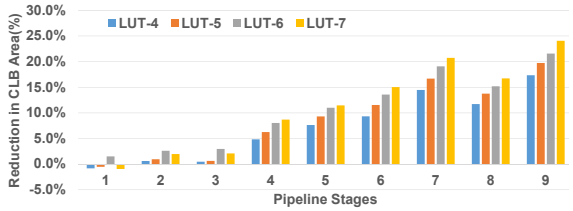


Fig. 7: Average reduction in BLE area in TLFPGA as compared to FPGA for varying LUT sizes and pipeline stages

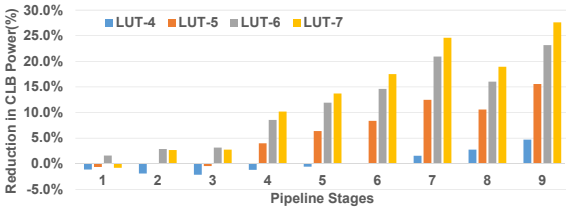


Fig. 8: Average reduction in BLE Power in TLFPGA as compared to FPGA for varying LUT sizes and pipeline stages

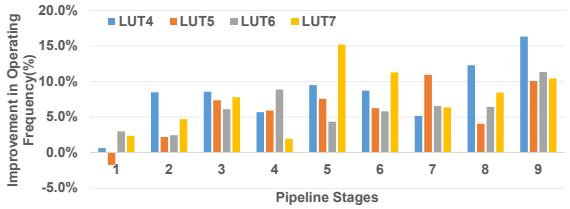


Fig. 9: Average gain in frequency in TLFPGA as compared to FPGA for varying LUT sizes and pipeline stages

BLE Area (Area) and BLE Power (Power) respectively in TLFPGA as compared to FPGA. Improvement in technology independent parameters such as Config Regs and MUX Count indicates that the TLFPGA can potentially offer benefits in other technology nodes as well. The Config Regs, MUX Count, Area and Power drop as the number of pipeline stages increases. More pipeline stages give more opportunities to the TCM to map TLCs. These experiments clearly indicate that TLFPGA is better suited for designs with deep pipelining. Since the logic depth of the circuit is not affected much during the TCM, the operating frequency in FPGA vs TLFPGA does not fluctuate much. This can be seen in Fig. 9.

Another observation that can be drawn from Fig. 5, 6, 7 and 8 is that the percentage reduction in configuration registers, MUX Count, Area and Power in TLFPGA as compared to FPGA increases with increase in the LUT size. This is because the number of configuration registers and MUX count for a BLE increase exponentially with an increase in LUT size, whereas the configuration registers and XORs in a TLC stay the same. As a result, TLFPGA provides more significant improvements when large LUTs are used.

Table III shows the average, maximum and minimum percentage reduction for configuration registers, MUX count, area and power, and percentage improvement in the operating frequency, over ISCAS-85 and ISCAS-89 circuits. Results show that the maximum improvement that we get for configuration registers, MUX count, area and power are far greater than the maximum degradation that we get when we map the circuit to TLFPGA. From the table, we can also see that all the parameters improve on an average. The average routing resources needed for the circuits are slightly reduced. These improvements can be easily carried over to more practical circuits such as filter and multiplier as well.

Though very little work has been done in the integration

TABLE III: Improvements in TLFPGA as compared to standard FPGA over ISCAS-85 and ISCAS-89 circuits

Parameter	Average	Maximum	Minimum
BLE Config Regs	8.9%	64.5%	-7.2%
BLE MUXes	15.4%	68.6%	-7.1%
BLE Area	7.0%	51.6%	-7.3%
BLE Power	8.2%	61.6%	-8.7%
Frequency	5%	165.5%	-33.1%
Routing Area	2.2%	51.5%	-58.5%
Routing Power	0.7%	75.2%	-51.7%

of threshold functions in a standard FPGA structure, a lot of work has been done on the implementation of efficient LUT structures and reconfigurable threshold logic arrays. Work on alternate LUT structures can be found in Ref. [9] and Ref. [10]. More information on commercial FPGAs can be found in Ref. [11], [12], and [13].

VI. Conclusion

Threshold logic based FPGAs proposed in this paper can be used to reduce the number of BLE configuration registers, multiplexers, area and power for most of the circuits we tested without compromising performance. The best results were obtained for deeply pipelined circuits where the percentage improvement increased with an increase in the number of pipeline stages and LUT size.

References

[1] Kai-Yeung Siu, Vwani Roychowdhury, and Thomas Kailath. *Discrete Neural Computation: A Theoretical Foundation*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1995.

[2] 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 Transactions on Very Large Scale Integration (VLSI) Systems*, 24(9):2873–2886, Sept 2016.

[3] N. Kulkarni, J. Yang, and S. Vrudhula. A fast, energy efficient, field programmable threshold-logic array. In *2014 International Conference on Field-Programmable Technology (FPT)*, pages 300–305, Dec 2014.

[4] Hao Jun Liu. Archipelago - an open source fpga with toolflow support. In *Thesis*, Berkeley, CA, 2014. UC Berkeley.

[5] Robert Brayton and Alan Mishchenko. Abc: An academic industrial-strength verification tool. In *Proceedings of the 22Nd International Conference on Computer Aided Verification*, CAV'10, pages 24–40, Berlin, Heidelberg, 2010. Springer-Verlag.

[6] Zied Farooq, Umerand Marrakchi and Habib Mehrez. *FPGA Architectures: An Overview*, pages 7–48. Springer New York, New York, NY, 2012.

[7] G. Beraudo and J. Lillis. Timing optimization of fpga placements by logic replication. In *Proceedings 2003. Design Automation Conference (IEEE Cat. No.03CH37451)*, pages 196–201, June 2003.

[8] G. Zgheib and P. Ienne. Evaluating fpga clusters under wide ranges of design parameters. In *2017 27th International Conference on Field Programmable Logic and Applications (FPL)*, pages 1–8, Sept 2017.

[9] Sayak Ray, Alan Mishchenko, Niklas Een, Robert Brayton, Stephen Jang, and Chao Chen. Mapping into lut structures. In *Proceedings of the Conference on Design, Automation and Test in Europe*, DATE '12, pages 1579–1584, San Jose, CA, USA, 2012. EDA Consortium.

[10] Wenyi Feng, Jonathan Greene, and Alan Mishchenko. Improving fpga performance with a s44 lut structure. In *Proceedings of the 2018 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, FPGA '18, pages 61–66, New York, NY, USA, 2018. ACM.

[11] Xilinx 7 Series FPGAs Configurable Logic Block User Guide.

[12] Cyclone V Device Overview.

[13] Polarfire FPGA Fabric User Guide.