An Efficient Reconfigurable Architecture by Characterizing Most Frequent Logic Functions

Iman Ahmadpour, Behnam Khaleghi, and Hossein Asadi
Department of Computer Engineering, Sharif University of Technology, Tehran

***Abstract*—Generous flexibility of *Look-Up Tables* (LUTs) in implementing arbitrary functions comes with significant performance and area overheads compared with their *Application-Specific Integrated Circuit* (ASIC) equivalent. One approach to alleviate such overheads is to use less flexible logic elements capable to implement majority of logic functions. In this paper, we first investigate the most frequently used functions in standard benchmarks and then design a set of less-flexible but area-efficient logic cells, called *Hard Logics* (HL). Since higher input functions have diverse classes, we leverage Shannon decomposition to break them into smaller ones to either reduce the HL design space complexity or attain asymmetric low input functions. A heterogeneous LUT-HL architecture and a mapping scheme are also proposed to attain maximum logic resource usage. Experimental results on MCNC benchmarks demonstrate that the proposed architecture reduces area-delay product by 13% and 36% as compared to LUT4 and LUT6 based FPGAs, respectively. Considering the same area budget, our proposed architecture improves performance by 17% and 2% as compared to LUT4 and LUT6 based FPGAs.**

I. Introduction

State-of-the-art *Field-Programmable Gate Arrays* (FPGAs) are composed of K-input *Look-Up Tables* (LUTs) as their processing building blocks. While a K-input LUT can implement any function with up to K variables, it comes with intensive area and performance overhead in designs implemented on FPGAs compared with their *Application-Specific Integrated Circuits* (ASICs) [1], [2]. The area overhead of LUTs also imposes significant power overhead as compared to its equivalent ASIC blocks.

To cope with significant area and performance overhead of LUTs, recent studies have attempted to replace LUTs by less flexible reconfigurable logic elements. This is motivated by the fact that not all K-variable functions stand with equal occurrence frequency in circuits [3]–[5]. As an example, the AND logic class by itself builds 30% to 40% of functions [3], [4]. These observations have inspired designers toward investigating new configurable elements which are able to implement a great deal of high frequent Boolean functions with less transistor count and/or with improved latency.

A number of previous studies have attempted to propose configurable elements include equipping FPGAs with dedicated hardware such as hardcore multipliers, *Digital Signal Processing* (DSP) blocks, carry chain [6], and arithmetic logic block [7]. Few other studies have tried to employ logic cells as a substitute for LUTs to improve area, power, or performance. Among these studies, COGRE [3] implements 93.4% and 50.6% of 4-input and 5-input functions using 8 and 11 SRAM cells, respectively. This study claims 10.0% to 46.3% reduction in logic area; however, this saving can be significantly deteriorated by the increase in routing area. In addition, the area and performance results have been compared with standard cell-based LUTs while transmission-gate based LUTs consume much less transistors and benefit from fast-input attribute. Another study has introduced the concept of *Extended LUT* (E-LUT) [8]. This study uses trimming input of functions to break a K-input function into two smaller ones, where one of them has fewer than K−1 variables. Without carrying out place and route experiments, the authors estimate 5% improvement in area-depth product. They also have neglected that their cells are not fully permutable anymore, which may incur routing overhead. In [4], the authors introduce Mega Cell which consists of a 4-input LUT and two other HLs which together cover more than 94% of functions. Every function will be mapped to one of the logic cells and the other pair will be power-gated. Due to limitation of the VPR toolset used in the experiments, the increased delay of longer routing wires has been neglected in this work. Another study presented in [9] has proposed *And-Invert-Cone* (AIC) that has a similar structure to And-Invert-Graph representation. The complexity of AIC is linear with its inputs and its delay increases logarithmically with the increase in inputs number. The authors have reported the area and delay improvements in terms of cluster reduction while no routing experiments has been performed and routing area is not considered. To achieve required input permutability, one can expect that the large full-crossbar connection blocks will result in significant routing overhead.

This paper presents an efficient set of configurable logic elements, referred to as HLs, to be replaced with conventional LUTs inside an FPGA. For this purpose, we use the Shannon expansion to break 5-variable functions into two asymmetric functions, wherein one of them has less than K−1 inputs. It is demonstrated that 98% of functions mapped into LUT5 can be decomposed such that at least one of its cofactors has less than K−1 variables. Our analysis also shows that 84% of LUTs can be decomposed in such a way that one of the resulted cofactors has a constant value, i.e., logical value of zero or one. For the other cofactors, we profile them to find the most frequent function classes in order to develop custom area- and/or performance-aware HLs. Using a comprehensive characterization of benchmarks circuits, we also propose a heterogeneous cluster architecture with an optimal number of each HL type, and a mapping scheme to achieve highest resource utilization. To evaluate the efficiency of the proposed architecture, we have implemented both the proposed architecture as well as previously proposed techniques such as COGRE, and E-LUT in the VPR toolset. Experimental results show that our proposed architecture improves the area-delay factor by 13%, 36%, 16% and 58% as compared with LUT4, LUT6, COGRE, and E-LUT architectures. Considering equal area budget, the proposed architecture improves the critical path delay by 17%, 2%, 14%, and 23% as compared to the aforementioned architectures, respectively.

The rest of this paper is organized as follows. Sec. II details

TABLE I. COVERAGE RATIO OF MOST FREQUENT 4-INPUT AND 3-INPUT NPN CLASSES IN 20 LARGEST MCNC BENCHMARK CIRCUITS

NPN class	Ratio	NPN class	Ratio
ABCD	33.3%	ABC	34.0%
AB+CD	22.8%	A(B+C)	29.0%
A+BCD	18.9%	AC+B!C	14.3%
(AB+C)D	12.3%	A(!B!C+BC)	7.6%
Others	13.0%	Others	15.0%

the proposed architecture. Sec. III presents the experimental setup and results. In Sec. IV, we discuss the limitations of our work and possible improvements. Finally, Sec. V concludes the paper.

II. PROPOSED ARCHITECTURE

The main challenge of proposing a substitute for LUT-based FPGAs is the coverage ratio of proposed cells, i.e., percentage of functions an HL can implement while making it as efficient as possible. If an arbitrary function cannot be mapped into the proposed HLs, the synthesis tool will decompose it into multiple sub-functions to fit them into the proposed HLs, or in the case of heterogeneous LUT-HL architectures will map the function to a LUT. The former case eventuates in increased number of logic blocks and hence wastes local (intra-cluster) or global (inter-cluster) routing resources. In the latter case, due to the high ratio of LUTs, improvements will be negligible especially considering the probable overheads in global routing due to the unpermutability of HL inputs.

A. NPN classes and Shannon decomposition

To design efficient HL blocks, we use the concept of *Negation-Permutation-Negation* (NPN) classes to categorize very large number of functions to smaller number of classes. Two functions F and G are NPN-equivalent if they can be derived from each other by negating and/or permuting inputs, and/or negating output. Although LUT6 provides the best performance compared with other LUT-based architectures [2], it has very diverse NPN classes that cannot be covered by a small set of logic functions with limited configuration cells [10]. On the other hand, LUT4 has the best area efficiency [2], however, the small gain obtained in logic area by replacing LUTs with custom HLs may deteriorated due to routing overhead, as in [8]. Therefore, we target 5-input functions which can be mapped into LUT5 and have relatively smaller number of NPN classes. In addition, we leverage Shannon expansion theorem to further explore the possibility of simplifying 5-input functions which subsequently leads in efficient and smaller set of HLs. Based on the Shannon Boolean expansion theorem, an arbitrary function defined as $F = f(x_1, ..., x_i, ..., x_n)$ can be decomposed with regard to any of its variables x_i to form a pair of cofactors, i.e., $F = x_i \cdot f(x_1, ..., 1, ..., x_n) + \bar{x_i} \cdot f(x_1, ..., 0, ..., x_n)$. Considering the Shannon decomposition, every 5-input function can be implemented using two 4-input LUTs and a 2:1 multiplexer. Nonetheless, our investigation on MCNC benchmark suite reveals that for most of 5-input LUTs at least one of the cofactors has less than four variables, particularly it is a solitary zero or one logic. Note that every K-input function has K pairs of cofactors (one pair per each input), so in order to explore the frequency of NPN classes, we choose the pair with minimum number of variables to further limit the NPN space.

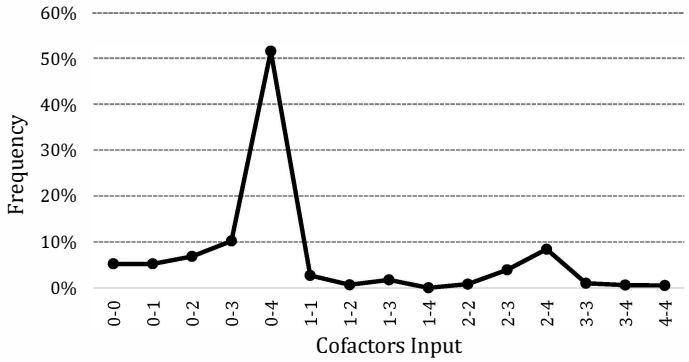
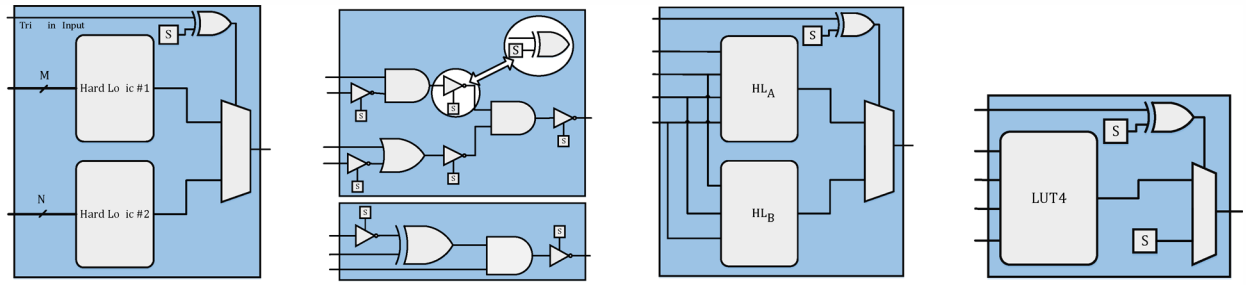


Fig. 1. Distribution of different cofactor pairs

B. Design of Hard Logic

The results of outlining LUT5 functions based on number of inputs of their minimum cofactors are demonstrated in Fig. 1. According to the results in this figure, in 97.7% of functions at least one of the cofactor has only two or less inputs. In addition, for ∼80% of functions at least one of the cofactors is a constant-zero or -one. This observation motivates us to design efficient HLs that implement four or less input functions instead of 5-input functions that initially were mapped into LUT5. Fig. 2(a) illustrates the general structure of a logic element decomposed into HL blocks, e.g., Hard Logic 1 and Hard Logic 2. Using the given structure of logic element based on HLs depicted in Fig. 2(a), next we design efficient HLs to embed into this structure. By employing *Boolean Matcher* tool [11], we investigate the cofactors and find out that nearly 87% of 4-input and 85% of 3-input cofactors belong to only four NPN classes as reported in Table. I.

At the first stage, we propose HL_A, as depicted in Fig. 2(b), which is able to implement three NPNs that altogether support 66% of 4-input functions. Note that negating the inputs cannot simply performed in the previous logic output since it may cause conflict in the case of multiple fanouts of previous logic. Considering $(!A \cdot !B) = !(A + B)$, if both inputs of the AND gates require negation, we provide the second inversion by the previous stage (if possible), or by exchanging the inputs of the AND with the OR gate and using their output configurable inverter. The same scenario holds for the OR gate. Evidently, while HL_A is designed for 4-input cofactors, it also can implement majority of 3-input functions since they are a subset of 4-input functions. However, considering the utilization of three and less input cofactors, we propose another area and delay efficient HL, called HL_B which is shown in Fig. 2(b). This HL helps us to further save logic block and intra-cluster multiplexer area for less input cofactors. Due to the high ratio of constant cofactors, we place HL_A and HL_B circuits in Hard Logic #1. For Hard Logic #2, we allocate a single configuration cell to employ constant logics. Our analysis using Boolean Matcher reveals that HL_A and HL_B blocks can implement 61% of LUT5 mapped functions in MCNC benchmarks. These functions are composed of a K-input cofactor where K is less than or equal to four, and a constant zero or one cofactor. Please note that for the remaining 4-input NPNs we examined other HLs, but considering their low utilization ratio, the induced complexity to implement different functions compromises the resulting gain.



(a) General structure of HL blocks (b) Proposed HL_A (top) and HL_B (bottom) cells (c) Proposed DHL (d) Used E-LUT4 Block [8]

Fig. 2. Structure of proposed Hard Logic Blocks

Investigating the remaining LUT5 functions leads us into two scenarios. The first group of functions does not contain any constant cofactor. Profiling these functions shows that most of them can be implemented by combination of HL_A and HL_B. Therefore, we introduce a *Double Hard Logic* (DHL) block as shown in Fig. 2(c). The second group includes functions that have a constant cofactor, but its other cofactor could not be implemented by neither HL_A nor HL_B. These functions will be mapped to a simple E-LUT4, as shown in Fig. 2(d). There is a minor set of functions which does not fit in any of mentioned structures. While these functions can be implemented by cascading two HL blocks, it will cause intra-cluster and global routing overhead. So, in the proposed architecture, we leave a small number of 5-input LUTs for such functions.

C. Mapping Algorithm

Using ABC tool [12], input BLIF file is mapped to 5-input LUTs. Next, by parsing the output BLIF we extract the corresponding Boolean function of each LUT. For each K-variable function, all K pairs of cofactors are extracted. For each cofactor pair, we find the target HL block with the priority of less area consumption. So, in a greedy manner we first check whether the HL_B block (with a single configuration cell for the other cofactor) can implement the cofactor. In a decreasing order of priorities, HL_A, DHL, E-LUT, and LUT5 are the next options. Algorithm 1 outlines the proposed mapping scheme.

Algorithm 1: Proposed Mapping Algorithm

Input: Circuit BLIF file
Output: Minimum area logic block for every function

```
Map input BLIF file to LUT5 using ABC;
for every LUT L_i in new BLIF do
    f_i ← Boolean function of L_i;
    S_i ← all possible cofactor pairs of f_i;
    for every cofactor pair P_j in set S_i do
        if NPN(P_j) ∈ NPN(HL_B) then
            HL_j ← HL_B;
        else if NPN(P_j) ∈ NPN(HL_A) then
            HL_j ← HL_A;
        else if NPN(P_j) ∈ NPN(DHL) then
            HL_j ← DHL;
        else if NPN(P_j) ∈ NPN(E-LUT) then
            HL_j ← E-LUT;
        else
            HL_j ← LUT5;
    HL_i ← min{HL_j | P_j ∈ S_i};
return;
```

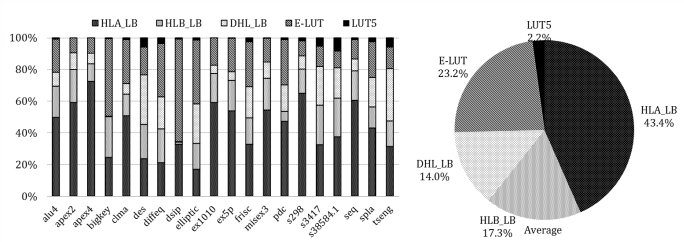


Fig. 3. Original distribution of logic blocks

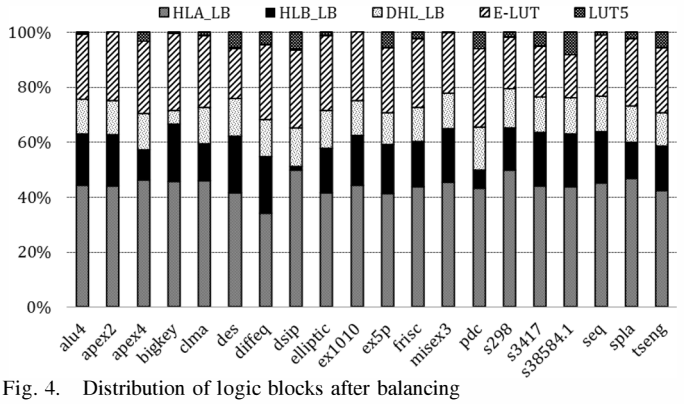


Fig. 4. Distribution of logic blocks after balancing

Fig. 3 shows the per benchmark and average distribution of the proposed logic blocks. As can be inferred from this figure, utilization percentage of each logic block varies for different applications. Since the number of each type of logic block in a cluster is specified based on their average distribution, it compromises the resource utilization in the proposed architecture. The logic block with the highest resource demand determines the final FPGA array size. For example, dedicated logic blocks for E-LUT4 will be about 23%, however, applying Algorithm 1, the *dsip* circuit requires more than 60% E-LUT4 block per cluster. Thus, it induces about 150% more clusters to provide sufficient E-LUT4 blocks. On the other hand, for this application, more than 97% of DHL blocks remains unutilized. To address this problem, we try to take advantage of overlapping functions. As an example, function $A \cdot (B \oplus C)$ initially is implemented by HL_B, due to its lower area. However, if the resource demand for HL_B exceeds its total share (i.e., 17% of logic blocks), some of these functions can be passed to any of DHL, E-LUT, and LUT5 logic blocks that are utilized below its dedicated share and have unused blocks. By applying this procedure in the final BLIF file, we efficiently balance each logic block utilization toward its average which was obtained for all 20 MCNC benchmarks. Fig. 4 shows the distribution of logic blocks after balancing step. Comparing this figure with Fig. 3 reveals the proximity

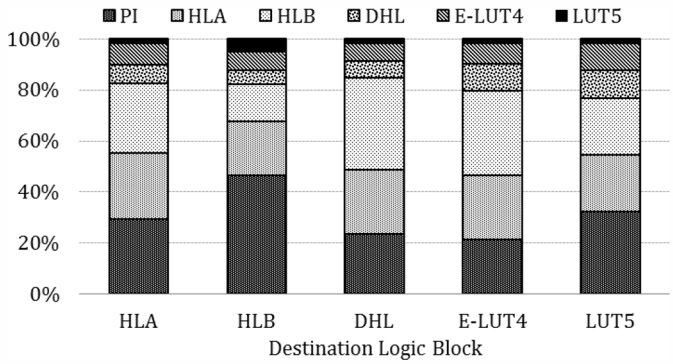


Fig. 5. Distribution of connections to different logic blocks

of logic block utilization ratio in different benchmarks after balancing.

D. Cluster Design

Designing a cluster of HL blocks should deal with three criteria; 1) total number of logic blocks within the cluster, 2) number of each logic block and, 3) cluster's interior connection block (input multiplexers) structure. Due to the heterogeneous nature and diversity of used logic blocks in the proposed architecture, choosing small clusters causes poor placement and heavy load on global routing due to variety of connecting pairs (e.g., HL_A to HL_B, E-LUT to DHL, etc.). For example, in a HL_A to LUT5 connection, the adjacent clusters may either lack a LUT5 because of low ratio of LUT5 blocks, or leave rough flexibility to placement algorithm. Based on this fact and by conducting experiments, we opted $N = 17$ as an optimum number of logic blocks inside a cluster. The number of each logic block type in a cluster can be obtained by their average ratio as discussed in Sec. II-C. In our experiments, the number of HL_A, HL_B, DHL, E-LUT4, and LUT5 is set to 7, 3, 2, 4, and 1, respectively.

To provide the best routability, the role of connections in input multiplexers of logic blocks becomes very important. To efficiently design multiplexers, we calculate the average of various connections between different logic blocks, which is shown in Fig. 5. Based on this figure, we infer that, for example, 27% of inputs of HL_A block originate from HL_B. Hence, we dedicate 27% of HL_A intra-cluster feedback inputs to the outputs of HL_B blocks in the same cluster.

III. Experimental Setup and Results

In order to evaluate efficiency of the proposed architecture, we implemented 20 largest MCNC benchmarks using the proposed architecture, E-LUT, COGRE, and conventional LUT-based architectures and compared their area and performance efficiency. For the E-LUT architecture, we chose its most efficient structure, i.e., two LUT4 based one, and for the COGRE, we compared with both COGRE5 and COGRE6. For the COGRE architectures, we created the genlib library of functions that COGRE5 and COGRE6 can implement, as input cell library for ABC tool. Then, we synthesized the MCNC benchmarks using the generated library and obtained the corresponding BLIF file. Delay and area parameters are extracted using HSPICE circuit level simulator using *Predictive Technology Model* (PTM) 45nm technology [13]. Finally, VPR 7.0 [14] is used for clustering, placement, and routing the benchmarks and extracting area and delay reports.

A. Architectural Parameters

1) General VPR Parameters: Since we have used 45nm technology in our simulations, for all the architectures we let VPR general parameters as the same in iFAR repository provided with VPR tool. Table II summarize the main general parameters that affect the final results.

TABLE II. General VPR Parameters

Parameter	Value	Parameter	Value
$R_{MinWidth}$ NMOS	8926	$R_{MinWidth}$ PMOS	16067
Switch block type	Wilton	Fs	3
Switch type	MUX	Switch delay	58
Segment length	4	Segment type	Unidir.

2) Area and Delay Parameters: Table III summarizes the area and delay for different logic blocks. Note that by HL_A LB we mean the logic block for HL_A which includes the HL_A cell, the SRAM for constant cofactor, and XOR gate. Such structure is depicted for the used E-LUT4 block in Fig. 2(d). Area is calculated as sum of the used minimum width transistors. Here, delay is reported as average delay of all inputs, while in simulations we used the actual delay for each input. For instance, the fastest input of LUT6 has only 15 ps delay.

3) Cluster Configuration: Table IV outlines the cluster configuration for both the proposed and baseline architectures. These values are proved to be the optimum configuration for the corresponding architectures [15]. Notice that for the COGRE architecture, we also examined larger number of logic blocks within the cluster. However, the cluster size of 6 (N=6) provides the best results. For the proposed architecture, as discussed in Sec. II, other values for N is also examined and $N = 17$ found to yield the best trade-off between area and delay. Number of cluster inputs, I, is opted from the well-known formula $I = \frac{K}{2} \times (N + 1)$ [2].

B. Area

In order to compare the architectures with respect to consumed transistor count, we use VPR with default routing options. In the default operation mode, it places the circuit on the minimum number of logical resources and in a binary-search mode and repetitively searches for minimum value of channel width (minimum routing resources) that can route the design. Reported area includes logic and routing area. Note that VPR counts the intra-cluster input multiplexers as logical resources, so usually the ratio of logic to the total area is higher than academic suppositions.

TABLE III. Summary of Area, Delay, and Number of Configuration Bits

Module	Area	# Conf. Bits	Delay (ps)	Module	Area	# Conf. Bits	Delay (ps)
HL_A LB	254	7	71	COGRE6	392	11	123
HL_B LB	143	4	42	Two E-LUT	1438	35	102
DHL LB	288	9	71	LUT4	654	16	66
E-LUT LB	709	18	60	LUT5	1327	32	108
COGRE5	291	8	100	LUT6	2643	64	110

TABLE IV. Cluster Configuration

Parameter	Proposed	COGRE5	COGRE6	LUT4	LUT5	LUT6	E-LUT
K	5	5	6	4	5	6	6
N	17	6	6	6	6	6	6
I	46	20	23	14	17	21	21
F_cin	0.3	0.5	0.5	0.5	0.5	0.5	0.5
F_cout	0.1	0.17	0.17	0.17	0.17	0.17	0.17
F_cfb	different	0.5	0.5	0.5	0.5	0.5	0.5

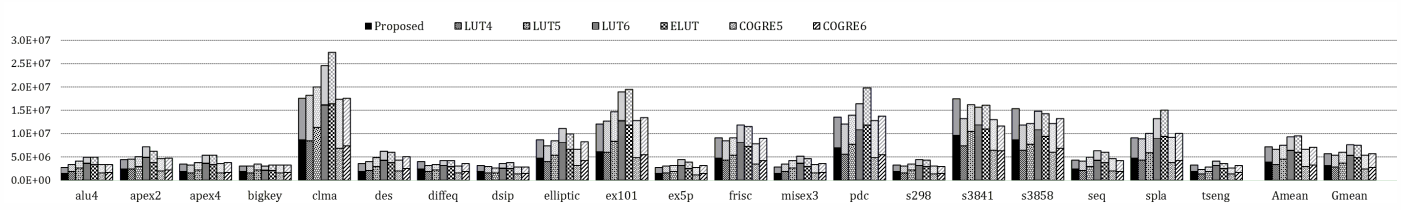


Fig. 6. Area comparison in area-constraint routing between the proposed and base-line architectures (bottom and top portions are logic and routing area, respectively)

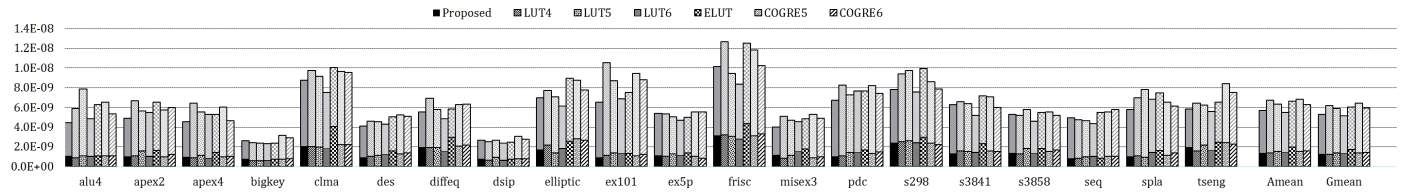


Fig. 7. Delay comparison in area-constraint routing between the proposed and base-line architectures (bottom and top portions are logic and routing delay, respectively)

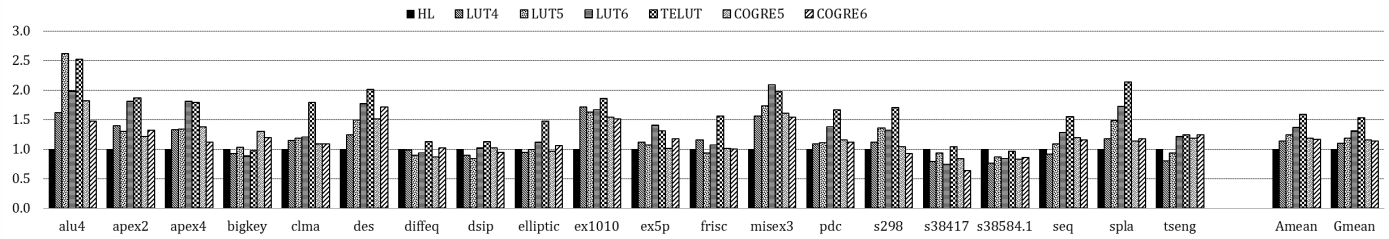


Fig. 8. Comparison of area-delay products between different architectures (normalized to the proposed HL architecture) in area-constraint routing

TABLE V. SUMMARY OF RESULTS (%)

	Area-constraint (default) routing			Equal-area routing
	Area	Delay	Area-Delay	Delay
LUT4	−4	17	13	17
LUT5	9	12	24	11
LUT6	37	−2	36	2
E-LUT	36	15	58	23
COGRE5	−3	21	18	24
COGRE6	3	12	16	14

Fig. 6 represents the logic and routing area for different architectures. As summarized in Table V, experimental results implies that the proposed architecture improves the total area by 9% and 37% as compared to LUT5 and LUT6 architectures, and imposes 4% area overhead compared with LUT4 which has been known to be the most area-efficient architecture among LUT-based architectures [2]. Comparing the logic and routing area distinctly with LUT5 which is identical with our proposed architecture in term of number of logic block inputs, the results reported in Fig. 6 show that the proposed architecture improves logic area by 21%. However, due to diversity of logic elements and connections, and due to unpermutability of the proposed logics, the proposed architecture increases routing area by 6%. In the total, the area is improved by 9% on average. On the other hand, COGRE5 increases the total number of logic blocks as compared to the proposed architecture, but due to its smaller logic blocks area, overall, it reduces the logic area by 20%. However, the increased number of logic blocks imposes routing overhead which compromises the total area gain.

Fig. 7 and Fig. 8 report the delay and area-delay product of different architectures, respectively, when are mapped to minimum sized device. The results reveal that the proposed architecture improves the area-delay product by 13%, 24%, and 36% compared to LUT4, LUT5, and LUT6 architectures, respectively. The summary results reported in Table V show that the improvements are due to improving in either area or delay (e.g. vs. LUT5), area (e.g. vs. LUT6), or delay (e.g. vs. LUT4).

C. Performance

Similar to the argument in Sec. III-B, comparing the performance of different architectures requires fair circumstances. For instance, mapping a circuit on LUT6 results in smaller critical path than LUT4 based architecture, but also it takes more area. In an area-equivalent condition, i.e., giving the same transistor cost that LUT6 spends to implement a design, LUT4 will represent improved performance due to enhanced flexibility of routing. Hence, in order to conduct such evaluation, for each benchmark we found the architecture with the largest total area. Then, for other architectures, we increased the channel width to reach the same area. This gives them a flexibility to conduct the routing in more efficient way and results in enhanced performance. Fig. 9 illustrates the critical path delay of MCNC benchmarks on the studied architectures in terms of logic and routing delay. Based on this figure, the proposed architecture improves the delay by 17%, 11% and 2% as opposed to LUT4, LUT5, and LUT6 architectures, respectively.

IV. DISCUSSION

State-of-the-art FPGAs target high performance applications using LUT6 in their structure as Xilinx Virtex-6 family [16]. However, there are other applications where performance is not the major criterion. On the other hands, vendors usually use a unique architecture for all FPGAs in the same family [16] and only vary their size. For example, Xilinx does not change

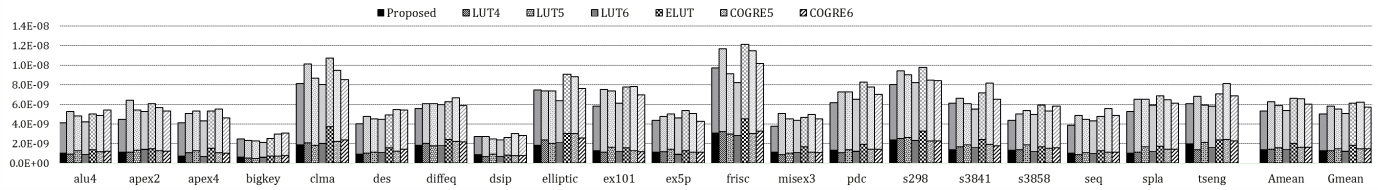


Fig. 9. Critical Path Delay Comparison

the LUT6-based architecture of Virtex-6 to LUT4-based area-efficient architecture for customers whose major constraint is either cost or power. The proposed architecture provides an efficient trade-off between performance, cost, and power. For high performance application domains, the proposed architecture provides the same performance as LUT6 with the same transistor cost. In addition, for economical applications it has only 4% area overhead compared with LUT4-based architecture, but provides considerable area-delay efficiency. Therefore, the proposed architecture can play a major role in both performance- and area-attentive applications.

Despite advantages of the proposed architecture, it lacks from few shortcomings. First, we have established our scenario based on analyzing the MCNC benchmarks. Using other benchmarks suites (such as IWLS) may result in different distribution of functions and NPN classes. Thus, for other set of benchmarks, type and/or ratio of proposed logic blocks may differ from currently suggested HLs. Nevertheless, we can classify applications into several categories, e.g., arithmetic cores, encryption/decryption modules, etc. and propose an efficient architecture for each one. Therefore, for different class of applications, different structures can be provided.

Another area that needs significant improvement is the mapping and clustering algorithm. This can be done by performing a clustering-aware logic balancing which needs conducting balancing simultaneously with clustering the logics. For example, changing a HL_B to HL_A block may evict it from a cluster (due to lack of HL_A in that cluster) or vice versa which complicates the global routing. In addition, since only 2% of functions has been mapped to LUT5, we could examine breaking such functions into multiple unutilized logic blocks (for example, can be mapped into two unused E-LUTs) and modify the clustering algorithm to place these logic parts inside the same cluster to prevent global routing overhead. This eliminates the need for pure LUT5 and makes the architecture simpler and enhances the clustering and routing process.

Furthermore, we encountered some VPR related limitations during the experiments. First, unfortunately VPR does not support specified or partial permutation. Each of the gates in the proposed logic blocks are permutable. Moreover, there are functions with inherent permutability, e.g., $ABCD$, which is entirely permutable, or $A \cdot (B + C + D)$ in which the inputs B, C, and D can be permuted. Including such option to the VPR router enhances the routing flexibility and can improve either the delay or routing area. Second, for non-LUT-based architectures, sometimes VPR is unable to find HL-latch pairs and places the HL and the corresponding latch in different logic blocks, so it wastes a logic block and imposes logic and/or routing overhead.

V. Conclusion

In this paper, we proposed an efficient architecture that can be substituted for current LUT-based architectures in both the area-oriented and high performance applications domain. The proposed architecture improves the area-delay factor by 13%, 24%, and, 36% as compared to LUT4, LUT5, and LUT6-based architectures, respectively. Considering equal transistor count, it improves the performance by 17%, 11%, and 2% compared with LUT4, LUT5, and LUT6-based architectures, respectively.

References

[1] I. Kuon and J. Rose, *Quantifying and exploring the gap between FPGAs and ASICs*. Springer Science & Business Media, 2010.

[2] E. Ahmed and J. Rose, "The effect of lut and cluster size on deep-submicron fpga performance and density," *Very Large Scale Integration (VLSI) Systems, IEEE Trans. on*, vol. 12, no. 3, pp. 288–298, 2004.

[3] Y. Okamoto, Y. Ichinomiya, M. Amagasaki, M. Iida, and T. Sueyoshi, "Cogre: A configuration memory reduced reconfigurable logic cell architecture for area minimization," in *Field Programmable Logic and Applications (FPL), 2010 Intl. Conference on*. IEEE, 2010, pp. 304–9.

[4] A. Ahari, B. Khaleghi, Z. Ebrahimi, H. Asadi, and M. B. Tahoori, "Towards dark silicon era in fpgas using complementary hard logic design," in *Field Programmable Logic and Applications (FPL), 2014 24th International Conference on*. IEEE, 2014, pp. 1–6.

[5] P.-Y. Hsu, P.-C. Lu, and Y.-Y. Liu, "An efficient hybrid lut/sop reconfigurable architecture," in *VLSI Design Automation and Test (VLSI-DAT), 2010 International Symposium on*. IEEE, 2010, pp. 173–176.

[6] "Virtex-II platform FPGAs: Complete data sheet," Data Sheet, Xilinx, November 2007.

[7] H. Parandeh-Afshar, P. Brisk, and P. Ienne, "A novel fpga logic block for improved arithmetic performance," in *Proceedings of the 16th international ACM/SIGDA symposium on Field programmable gate arrays*. ACM, 2008, pp. 171–180.

[8] J. H. Anderson and Q. Wang, "Area-efficient fpga logic elements: Architecture and synthesis," in *Proceedings of the 16th Asia and South Pacific Design Automation Conference*. IEEE Press, 2011, pp. 369–375.

[9] H. Parandeh-Afshar, H. Benbihi, D. Novo, and P. Ienne, "Rethinking fpgas: elude the flexibility excess of luts with and-inverter cones," in *Proceedings of the ACM/SIGDA international symposium on Field Programmable Gate Arrays*. ACM, 2012, pp. 119–128.

[10] A. Kennings, K. Vorwerk, and A. Mishchenko, *Generating efficient libraries for use in FPGA resynthesis algorithms*. Department of Electrical and Computer Engineering, University of Waterloo, 2010.

[11] D. Chai and A. Kuehlmann, "Building a better boolean matcher and symmetry detector," in *Proceedings of the conference on Design, automation and test in Europe: Proceedings*. European Design and Automation Association, 2006, pp. 1079–1084.

[12] A. Mishchenko *et al.*, "Abc: A system for sequential synthesis and verification," *URL http://www. eecs. berkeley. edu/˜ alanmi/abc*, 2007 (accessed June 26, 2015).

[13] (2013 (accessed June 26, 2015)) Predictive technology model (ptm). [Online]. Available: http://ptm.asu.edu/

[14] J. Luu, J. Goeders, M. Wainberg, A. Somerville, T. Yu, K. Nasartschuk, M. Nasr, S. Wang, T. Liu, N. Ahmed *et al.*, "Vtr 7.0: next generation architecture and cad system for fpgas," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 7, no. 2, p. 6, 2014.

[15] G. Lemieux and D. Lewis, "Using sparse crossbars within lut," in *Proceedings of the 2001 ACM/SIGDA ninth international symposium on Field programmable gate arrays*. ACM, 2001, pp. 59–68.

[16] "Virtex-6 fpgas configurable logic block," User Guide, Xilinx, February 2012.