# Pattern-Based FPGA Logic Block and Clustering Algorithm

Xifan Tang, Pierre-Emmanuel Gaillardon and Giovanni De Micheli
École Polytechnique Fédérale de Lausanne (EPFL), Switzerland
Email: xifan.tang@epfl.ch

***Abstract*—In classical FPGA, LUTs and DFFs are pre-packed into BLEs and then BLEs are grouped into logic blocks. We propose a novel logic block architecture with fast combinational paths between LUTs, called pattern-based logic blocks. A new clustering algorithm is developed to release the potential of pattern-based logic blocks. Experimental results show that the novel architecture and the associated clustering algorithm lead to a 14% performance gain and a 8% wirelength reduction with a 3% area overhead compared to conventional architecture in large control-instensive benchmarks.**

## I. Introduction

*Field Programmable Gate Arrays* (FPGAs) use cluster-based logic blocks that consist of a number of *Basic Logic Elements* (BLEs) [1]. Inside the logic blocks, BLEs are fully connected by the local routing multiplexers. To increase the efficiency of logic blocks, previous works [2]–[4] focused on improving the local routing multiplexers. However, very limited works investigate the efficiency of the BLEs.

A *Basic Logic Element* (BLE) consists of a *Look-Up Table* (LUT), a *D Flip-Flop* (DFF), and a 2:1 multiplexer. It can work in either combinational or sequential mode. FPGA clustering algorithms group LUTs and DFFs into BLEs (pre-packing step) and then cluster BLEs into logic blocks (packing step) [1]. A BLE has only one fanout, which forces combinational output of LUTs to pass through the 2:1 multiplexer before reaching the local routing. This imposes strong limitations on the clustering algorithm during the pre-packing. In circuits with short critical paths, e.g., control-intensive circuits, the BLE architecture prolongs the critical path and reduces the performances significantly.

In this paper, we propose (i) a novel scalable logic block organization called pattern-based logic block. Investigating the different interconnection patterns that may exist between LUTs, we create groups of LUTs with fast combinational shortcuts. To fully unlock the performances of the new logic block structure, we introduce (ii) a pattern-based clustering algorithm, able to efficiently take advantage of the fast combinational paths. The combination of pattern-based logic block and clustering algorithm contribute to a 16% performance gain and a 8% wirelength reduction with a 3% area overhead compared to a conventional FPGA architecture at 40-nm technology node for a set of large control-intensive benchmarks.

The rest of the paper is organized as follows. Section II reviews the background of this work including conventional FPGA logic block architecture and associated clustering algorithms. Section III introduces the pattern-based logic block design, while Section IV describes the ad-hoc pattern-based clustering algorithm. Section V presents the experimental results and compares the approach to conventional architecture. Section VI concludes the paper.

## II. Background

### A. Classical Cluster-Based Logic Block

Modern FPGAs use an island-style architecture, where logic blocks are surrounded by pre-fabricated routing resources. The logic block themselves consist of *Basic Logic Elements* (BLEs) and a fully interconnected local routing [1]. Fig. 1 illustrates the architecture of a classical cluster-based logic block. A cluster-based logic block consists of a number *N* of BLEs. Each BLE contains a *k*-input LUT, a DFF and a 2:1 multiplexer. A BLE realizes fine-grain combinational or sequential operations. Its mode of operation (combinational or sequential) is controlled by the 2:1 multiplexer. Local routing, consisting of a large set of multiplexers, can route any output of BLEs to their inputs, enhancing the inner logic block routability. The logic block features *I* inputs that come from the global routing. Given *k* and *N*, setting $I = \frac{k(N+1)}{2}$ ensures that 98% BLEs are utilized on average [5]. To efficiently pack LUTs and DFFs into cluster-based logic block, clustering algorithms are of fundamental importance.

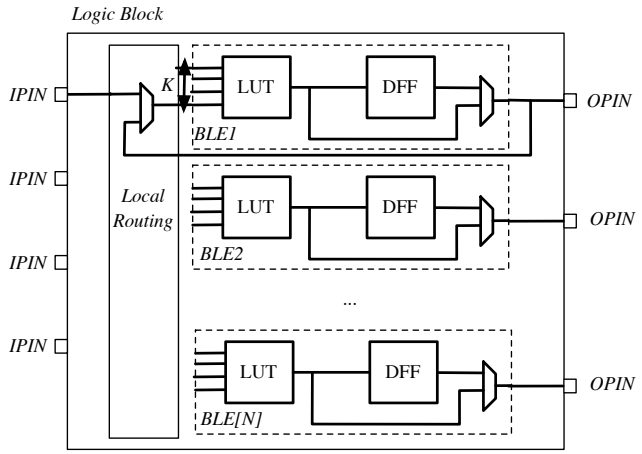


Fig. 1. Classical Cluster-based Logic Block Architecture

### B. Clustering Algorithm

Modern FPGA clustering algorithms can be grouped into two categories: seed-based and partition-based. Seed-based clustering algorithms [6]–[9] select a seed BLE with the highest criticality, pack it into a logic block and continue to absorb BLEs until the logic block cannot accommodate any more. Partition-based clustering algorithms [10] [11] depend on a graph partitioner [12] to cut the circuits into small parts and then modify the results to fit CLB capacity. AA-Pack [13] adapts optimizing techniques of seed-based packers [6]–[9] to pack heterogeneous logic blocks and supports flexible local routing architectures. AA-Pack brings novel opportunities to study the inner logic block routing. We use similar techniques in this paper, therefore we focus on introducing AA-Pack. AA-Pack [13] groups LUTs and DFFs into logic blocks in two steps. In the first step, called pre-pack, LUTs and DFFs are packed into BLEs, as shown in Fig. 2. Note that in Fig. 2(b), an additional BLE has to be created due to the limited fanout of the BLE architecture [1]. After pre-pack, timing analysis

is carried out and timing slacks are marked for each BLE, preparing for timing-driven clustering. In the second step, AA-Pack pack BLEs into logic blocks. It starts by initializing an empty logic block, then chooses a seed and uses an attraction function to select the candidate block to fill in. When two candidates blocks have the same attraction, AA-Pack selects the one with largest number of critical/near critical paths, called *PathAffects* [1], passing through. If the two candidates have the same *PathAffects*, AA-Pack selects the one with largest depth from critical path source, called $D_{source}$ [1]. In AA-Pack, each time the most "attractive" candidate is chosen, a local router is speculatively called to determine whether the candidate can be accepted. When the logic block is full, AA-Pack starts another iteration until all BLEs are packed.

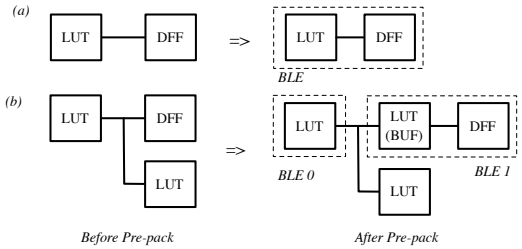


Fig. 2. Classical pre-pack

## III. Pattern-Based FPGA Logic Block

In this section, we introduce our novel pattern-based FPGA logic block architecture. Patterns are defined as groups of LUTs, among which there exist fast combinational interconnections.

### A. Combinational Interconnection Patterns

To improve the routing of combinational paths, we study the different interconnection possibilities between LUTs. We first formulate the following characteristics of LUTs:

**C1)** All the inputs of a LUT are logic equivalent, and thus are swappable.

**C2)** LUTs (actually any combinational logic gate) cannot have combinational loops, which means that the interconnections among LUTs are acyclic.

**C3)** Any two inputs of a LUT (actually any combinational logic gate) cannot share the output of a same LUT, otherwise these shared inputs can be reduced to **one**.

**C4)** Combining **C2** with **C3**, there should be only **one** combinational connection between two LUTs.

Thanks to the above characteristics, the number of combinational interconnection patterns between LUTs is limited. We define *X* as the size of the pattern. It corresponds to the number of LUTs involved in the pattern. Note that we limit our study to $k \geq X - 1$. In the following, we study the cases of pattern-2 and pattern-3, then we generalize to pattern-*X*.

*1) Pattern-2:* Fig. 3 illustrates all possible interconnection cases between two *k*-LUTs and demonstrates the pattern covering all possibilities. Given two *k*-LUTs (tagged 1 and 2), only two cases can be identified for their interconnections. First, a direct connection may exist between the output of one LUT and one of the inputs of the second LUT. In Fig. 3(a), the output of LUT1 is connected to an input of LUT2. From **C4**, there should be only one interconnection between LUT1 and LUT2, and by applying **C1**, we can always keep the output of LUT1 connected to the input *in0* of LUT2. Note that when using local routing in cluster-based logic block, LUT1 and LUT2 are swappable. Thus, Fig. 3(b) can be regarded as equivalent to Fig. 3(a). Second, inputs of LUT1 and LUT2 can

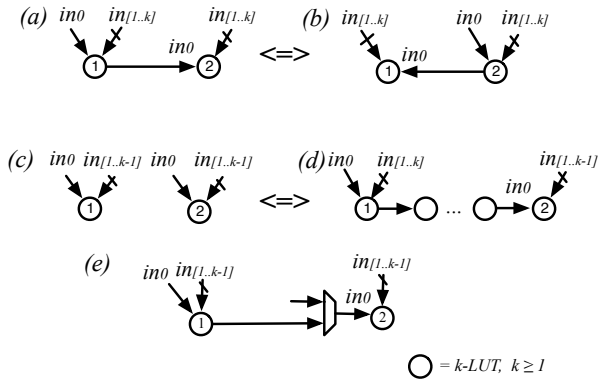


Fig. 3. All possible combinational interconnections between 2 *k*-LUTs.
(a) and (b): 2 *k*-LUTs are directly connected.
(c): 2 *k*-LUTs are independent.
(d): 2 *k*-LUTs are indirectly connected.
(e): interconnection pattern covering (a)(b)(c)(d)

be fully independent as shown in Fig. 3(c). For instance, all the LUT inputs are connected to different primary inputs, LUTs or DFFs. Fig. 3(d) presents a possibility where the output of LUT1 is connected to the input of LUT2 through other LUTs. Fig. 3(c) and (d) can be regarded as equivalent because they are all connected through the local routing. Therefore, when two LUTs are considered, only two cases (Fig. 3(a) and (c)) should be considered. Hence, we can create a universal structure able to map these different configurations by adding one multiplexer as shown in Fig. 3(e). This structure is called pattern-2, and can realize all the interconnection patterns between **2** LUTs.

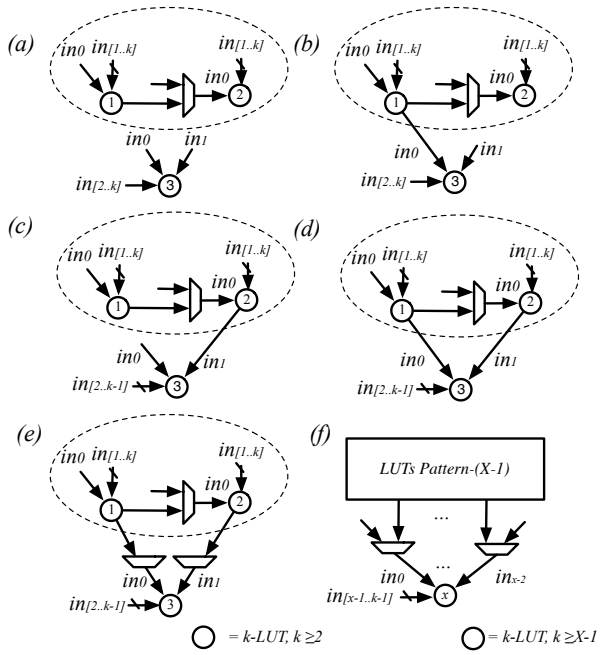


Fig. 4. All possible combinational interconnections between 3 *k*-LUTs.
(a) and (b): 3rd *k*-LUT is independent.
(c): one example of 3rd *k*-LUT connected to one of the other LUTs.
(d): 3rd *k*-LUT is connected to all the other LUTs.
(e): interconnection pattern covering (a)(b)(c)(d).
(f): interconnection pattern of M LUTs

*2) Pattern-3 to Pattern-M:* Based on the pattern-2 organization, we can extend the structure to three *k*-LUTs (tagged 1, 2 and 3). First, Fig. 4(a) shows the case where the inputs of LUT3 are fully independent from LUT1 and LUT2. Then, we can repeat the same reasoning than previously for direct connections between LUTs. Fig. 4(b)(c)(d) list all the possible cases where the inputs of LUT3 are connected to the outputs of LUT1 and LUT2. The cases where the output of LUT3 is connected to the inputs of LUT1 and LUT2 are not listed but can be regarded as equivalent to Fig. 4(b)(c)(d) by swapping LUT3 with LUT1 or LUT2. Considering all the cases in Fig. 4(a)(b)(c)(d), pattern-3 is proposed in Fig. 4(e).

On a general basis, we can extend the pattern size from 3 to *X*. Since pattern-*(X-1)* covers all possible interconnections among *(X-1)* LUTs, pattern-*X* can be achieved by considering an additional LUT (tagged *x*). The number of inputs of LUT*x* connected to pattern-*(X-1)* ranges from 0 to *(X-1)*. Hence, *(X-1)* 2:1 multiplexers can be added to each input of LUT*x* as depicted in Fig. 4(f). In a pattern-X, the number of additional 2:1 multiplexers is $X(X-1)/2$.

### B. Pattern-Based Logic Block Design

To build a logic block based on a pattern-X, the extra 2:1 multiplexers of the patterns can be included either (i) in an independent layer between local routing and BLEs, providing ultra-fast shortcuts at the cost of larger delays from logic block inputs to LUTs; or (ii) merged into the local routing. In this paper, we study the second case for simplicity. The BLE architecture remains unchanged and we simply feedback the outputs of LUTs to the local routing. Modern FPGA architectures typically use 6-input LUTs in their logic blocks. We therefore employ a pattern-7 organization. The schematic of a pattern-7 logic block is given in Fig. 5. The use of larger multiplexers leads to 0.45% area overhead. The fast combinational interconnections between LUTs are highlighted in red. Note that a pattern-based logic block can also contain multiple pattern-X. In this paper, we focus only on single pattern logic blocks to evaluate the efficiency of the approach.

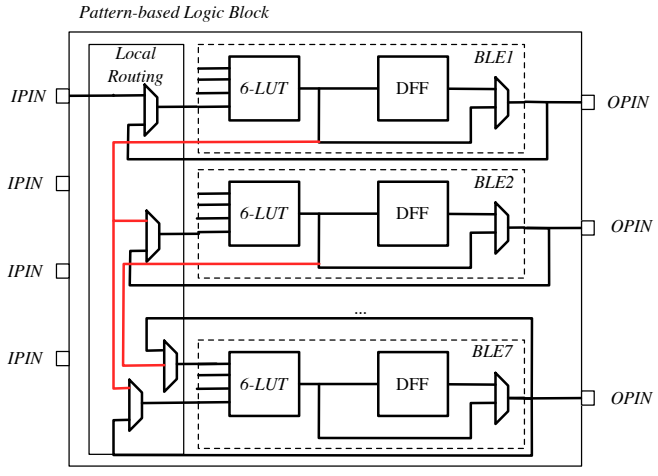


Fig. 5. Pattern-Based CLB

## IV. Pattern-Based Clustering Algorithm

To support the introduced pattern-based architecture, we developed a new clustering algorithm. While derived from AA-Pack, it aims at attracting patterns rather than single BLEs. A pattern candidate consists of a seed BLE and its unpacked predecessors. The predecessor selections is bounded by the maximum pattern size available in the cluster.

Our pattern-based algorithm adapts the attraction functions as well as *PathAffects* identification of AA-Pack. Let *lb* denotes the logic block, $p$ a pattern and $B_i$ the BLEs involved in the pattern $p$. As each time we absorb a pattern including a number of $B_i$ BLE candidates, we define the attraction function as the sum of the attraction of each candidate $B_i$:

$$\begin{aligned} attraction(lb,p) &= \sum_i attraction(lb,B_i) \\ &= \sum_i [\alpha \cdot timing\_criticality(lb,B_i) \\ &+ (1-\alpha)\cdot area\_attraction(lb,B_i)] \end{aligned} \quad (1)$$

The area attraction function is modified to increase the absorption of logic block outputs:

$$\begin{aligned} &area\_attraction(p,B_i) = \\ &\frac{1}{num\_pins(lb)}[(1-\beta)\cdot share\_input\_nets(lb,B_i)+ \\ &\beta \cdot absorbed\_output\_nets(lb,B_i)] \end{aligned} \quad (2)$$

where $share\_input\_nets(lb,B_i)$ is the number of input nets shared by $lb$ and $B_i$, and $absorbed\_output\_nets$ denotes the number of output nets of $lb$ absorbed by $B_i$. In our experiments, parameters $(\alpha,\beta)=(0.75,0.9)$ yield good performance. Similarly, we define *PathAffects(p)* as the average of the *PathAffect* of each candidate $B_i$:

$$PathAffects(p) = \frac{\sum PathAffects(lb,B_i)}{|p|} \quad (3)$$

and $D_{source}$ of a pattern as the average of the $D_{source}$ of each candidate $B_i$:

$$D_{source}(p) = \frac{\sum D_{source}(lb,B_i)}{|p|} \quad (4)$$

## V. Architectural-Level Simulations

In this section, experimental results are presented. Experimental methodology is first introduced, and followed by the discussion of the results.

### A. Methodology

Modern FPGAs use 6-input LUTs. Therefore, we consider pattern-7 as a reasonable size to investigate the new logic block architecture. Logic block architecture is set as $k=6$, $N=7$, $I=\frac{k(N+1)}{2}=24$. As for routing architecture and physical design parameters, we refer to the Altera Stratix IV GX device at 40-nm technology, available from iFAR [15]. Routing architecture uses single-driver length-4 wires [16], with $Fc(In)=0.15$ and $Fc(Out)=0.10$. Benchmark set consists of some large OpenCores projects [17]. All benchmarks pass through logic synthesis by ABC [18]. Then, they are packed either by our pattern-based packer or AA-Pack, and placed and routed by VPR 7 [19]. We evaluate the pattern-based architecture and clustering algorithm by running 3 sets of experiments: 1) the standard CAD flow shown, i.e., based on AA-Pack in Fig. 6(a) with a standard baseline architecture to serve as reference; 2) the same standard flow with the novel pattern-based architecture to evaluate the promises of the novel architecture; and 3) the pattern-based CAD flow shown in Fig. 6(b) with pattern-based architecture to evaluate the joint efforts of architecture and clustering algorithm.

### B. Experimental Results

Table I lists the results of the 3 sets of experiments. We first compare the results obtained using the standard flow, then we comment on the new flow.

TABLE I. Comparison between standard flow and pattern-based flow

| Benchmarks | | | Std. flow, Std. arch. | | | Std. flow, Pattern arch. | | | Pattern-based flow, Pattern arch. | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **Open Cores** | LUT No. | DFF No. | Area (# of trans.) | Crit. Delay (ns) | Wire-length | Area improv. | Crit. Delay improv. | Wire-length improv. | Area improv. | Crit. Delay improv. | Wire-length improv. |
| ac97_ctrl | 2790 | 2199 | 1.06E+07 | 3.21 | 27146 | +51.01% | -22.22% | +33.54% | -0.26% | -18.83% | -2.76% |
| pci_conf_cyc_addr_dec | 26 | 0 | 1.31E+05 | 2.33 | 567 | +11.57% | -22.52% | -24.87% | +11.57% | -23.69% | -19.58% |
| pci_spoci_ctrl | 243 | 60 | 9.23E+05 | 3.79 | 2427 | +6.37% | -5.50% | +16.52% | -1.25% | -11.25% | -0.08% |
| systemcdes | 503 | 190 | 2.14E+06 | 4.08 | 7191 | +10.46% | -10.32% | +2.03% | +4.38% | -11.65% | +0.97% |
| usb_phy | 100 | 98 | 4.84E+05 | 1.84 | 828 | +36.20% | +3.89% | +38.89% | +5.07% | -23.04% | -21.50% |
| des_perf | 6099 | 8746 | 5.10E+07 | 4.13 | 154738 | +33.42% | -0.95% | +7.57% | -2.13% | -8.32% | -10.44% |
| Avg. | | | **1.09E+07** | **3.23** | **32150** | **+24.84%** | **-9.60%** | **+12.28%** | **+2.90%** | **-13.83%** | **-7.63%** |

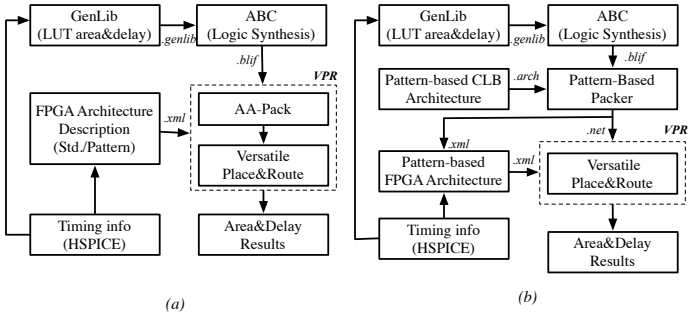


Fig. 6. EDA flow comparison: (a) Standard EDA flow (b) Pattern-based EDA flow

*1) Standard Architecture - Standard Flow vs. Pattern Architecture - Standard Flow:* In this comparison, we evaluate the potential of our novel architecture considering the area, the critical path delay and the wirelength numbers between a standard architecture and the novel pattern architecture using the same CAD flow. In OpenCores projects, pattern architecture obtains a 9.6% reduction in delay at a bigger cost in area and wirelength. This implies that pattern architecture can instruct AA-Pack to produce better performance even without utilizing the fast combinational paths. In some benchmarks, such as ac_97ctrl and pci_conf_cyc_addr_dec, pattern architecture produces very significant gain in delay and wirelength.

*2) Standard Architecture - Standard Flow vs. Pattern Architecture - Pattern Flow:* We evaluate the performance of our pattern-based flow by comparing the area, the critical path delay and the wirelength between the standard flow with standard architecture and our pattern-based flow with pattern-based architecture. Pattern-based flow increases area by 3% and shrinks delay and wirelength by 14% and 8% respectively on average. Compared to the results gathered with the standard flow, the pattern-based packer reduces the area overhead and increases further the gain in delay. Delay improvements are accounted for the fast combinational paths and for the reduction of additional LUTs to accommodate large fanouts. Critical paths of the selected OpenCores projects are short, which makes delay gain significant. The limited area loss comes from the pattern-based candidate selection, which tends to group LUTs that are intensively connected to each other instead of simply greedily absorbing the nets. Wirelength gains are accounted for (i) the novel logic block that can absorb more nets, and for (ii) the pattern-based clustering algorithm that packs the circuits with a global optimization instead of local scope on optimality.

## VI. Conclusion

In this paper, we investigate the interconnection patterns of LUTs inside standard cluster-based logic blocks and propose a novel pattern-based logic block architecture. Providing fast combinational path between LUTs, pattern-based logic block generates 0.45% area overhead when LUT size is 6. To take the advantage of fast combinational paths, a pattern-based clustering algorithm is proposed. Experimental results demonstrate that for OpenCores project benchmarks, pattern-based logic block architecture and clustering algorithm contribute to a 14% reduction in critical delay and a 8% shrink in wirelength with a limited 3% area overhead, on average, compared to standard logic block architecture.

## Acknowledgment

This work has been partly supported by the ERC senior grant NanoSys ERC-2009-AdG-246810 and the Swiss National Science Foundation under the project No. 200021-146600.

## References

[1] V. Betz *et al.*, *Architecture and CAD for Deep-Submicron FPGAs*, Kluwer Academic Publishers, 1998.

[2] D. Lewis *et al*, *The Stratix TM Routing and Logic Architecture*, FPGAs *Tech. Dig.*, 2003, pp. 12-20.

[3] K. Wang *et al.*,, *A Novel Packing Algorithm for Sparse Crossbar FPGA Architectures*, ICSICT, 2008, pp. 2345-2348.

[4] G. Ni *et al.*, *A New FPGA Packing Algorithm Based on the Modeling Method for Logic Block*, IEEE *Int'l conf.* on ASICs, 2005, pp. 877-880.

[5] E. Ahmed *et al.*, *The Effect of LUT and Cluster Size on Deep-Submicron FPGA Performance and Density*, IEEE TVLSI, Vol. 12, No. 3, 2004, pp. 288-298.

[6] V. Betz *et al.*, *Cluster-Based Logic Blocks for FPGAs: Area-Efficiency vs. Input Sharing and Size*, IEEE CICC, 1997, pp. 551-554.

[7] A. Marquardt *et al.*, *Using Cluster-Based Logic Blocks and Timing-Driven Packing to Improve FPGA Speed and Density*, FPGAs *Tech. Dig.*, 1999, pp. 37-46.

[8] E. Bozorgzadeh *et al.*, *Routability-driven Packing: Metrics and Algorithms for Cluster-Based FPGAs*, Journal of Circuits Systems and Computers, Vol. 13, No. 1, 2004, pp. 77-100.

[9] A. Singh *et al.*, *Efficient Circuit Clustering for Area and Power Reduction in FPGAs*, TODAES, Vol. 7, No. 4, 2002, pp. 643-663.

[10] W. Feng, *K-way Partitioning Based Packing for FPGA Logic Blocks without Input Bandwidth Constraint*, ICFPT, 2012, pp. 8-15.

[11] D. Chen *et al.*, *Improving Timing-Driven FPGA Packing with Physical Information*, FPL, 2007, pp. 117-123.

[12] G. Karypis *et al.*, *Multilevel K-Way Hypergraph Partitioning*, DAC, 1999, pp. 343-348.

[13] J. Luu *et al.*, *Towards Interconnect-Adaptive Packing for FPGAs*, FPGAs *Tech. Dig.*, 2014, pp. 21-30.

[14] J. Luu *et al.*, *Architecture Description and Packing for Logic Blocks with Hierarchy, Modes and Complex Interconnect*, FPGAs *Tech. Dig.*, 2011, pp. 227-236.

[15] Univ. Toronto, *Intelligent FPGA Architecture Repository*, http://www.eecg.toronto.edu/vpr/architectures/

[16] G. Lemieux, E. Lee, M. Tom, A. Yu, *Directional amd Single-Driver Wires in FPGA interconnect*, ICFPT, 2004, pp. 41-48.

[17] http://www.opencores.org

[18] University of California in Berkeley, *ABC: A System for Squential Synthesis and Verification*, Available online. http://www.eecs.berkeley.edu/~alanmi/abc/

[19] J. Rose *et al.*, *The VTR Project: Architecture and CAD for FPGAs from Verilog to Routing*, FPGAs *Tech. Dig.*, Feb. 2012, pp. 77-86.