

# Short Papers

## Design and Synthesis of Programmable Logic Block With Mixed LUT and Macrogate

Yu Hu, *Student Member, IEEE*, Satyaki Das, Steve Trimberger, and Lei He, *Member, IEEE*

**Abstract**—Small gates, such as AND2, XOR2, and MUX2, have been mixed with lookup tables (LUTs) inside programmable logic blocks (PLBs) to reduce area and power and increase performance in FPGAs. However, it is unclear whether incorporating macrogates with wide inputs inside PLBs is beneficial. In this paper, we first develop a methodology to extract a small set logic functions that are able to implement a large portion of functions for given FPGA applications, and propose a heterogeneous PLB with one LUT and one macrogate for the selected logic functions. Furthermore, we develop a synthesis flow for such heterogeneous PLBs, including a cut-based delay and area optimized technology mapping, a mixed binary integer and linear programming-based postmapping area recovery to balance the utilization of macrogates and LUTs, and a SAT-based PLB architecture-aware packing. Experiments using over 70 industrial benchmark applications show that we can extract four six-input logic functions to cover more than 50% functions of these applications, and the proposed synthesis flow reduces area by 5% compared to an alternative flow without the postmapping area recovery when both have the optimal logic depth. Compared to the PLB with mixed LUT-4 and small macrogates (XOR2 and MUX2), the PLB with mixed LUT-4 and four-input macrogate reduces logic depth by 6% (and up to 42%) for the aforementioned applications.

**Index Terms**—Architecture, field programmable gate array (FPGA), technology mapping.

### I. INTRODUCTION

The popular island-style FPGA architecture [1] consists of programmable logic blocks (PLBs) embedded in routing channels. The logic element within the PLB can be a lookup table (LUT) [2], programmable logic array (PLA) [3], or macrogate (e.g., AND gates and multiplexers) [4]. These logic elements offer a spectrum of tradeoffs between functionality and cost in terms of area, power, and delay. For instance, a circuit can be implemented by fewer  $K$ -input LUTs than by  $K$ -input macrogates, while a  $K$ -input macrogate requires smaller silicon area and has lower propagation delay than a  $K$ -input LUT. The PLB is *heterogeneous* if it consists of different types of logic elements, otherwise it is *homogeneous*. In this paper, we assume that heterogeneity exists only inside a PLB, while the structures of all PLBs are identical, and we study the impact of heterogeneous PLBs.

Synthesis and architecture evaluation have been extensively studied for homogeneous PLBs with uniform LUTs, and the representative work includes [1] and [5]. Initial studies in the literature have suggested various heterogeneous PLB architectures, including mixed-sized LUTs [6]–[9], mixing LUTs and PLAs [10] inside the PLB,

Manuscript received August 24, 2007; revised January 11, 2008, May 13, 2008, and September 11, 2008. Current version published March 18, 2009. This work was supported in part by the NSF under Grant CCR-0306682. This paper was recommended by Associate Editor K. Bazargan.

Y. Hu and L. He are with the Department of Electrical Engineering, University of California at Los Angeles, Los Angeles, CA 90095 USA (e-mail: hu@ee.ucla.edu; lhe@ee.ucla.edu).

S. Das and S. Trimberger are with Xilinx, Inc., San Jose, CA 95124 USA (e-mail: satyaki.das@xilinx.com; steve.trimberger@xilinx.com).

Color versions of one or more of the figures in this paper are available online at <http://ieeexplore.ieee.org>.

Digital Object Identifier 10.1109/TCAD.2009.2014001

PLBs with hard-wired connections [11], or mapping logic to FPGAs with both LUTs and embedded ROMs [12], [13], may improve logic density.

In this paper, we focus on heterogeneous PLB with a mix of LUTs and macrogates. Commercial FPGAs [6] have applied small macrogates (e.g., XOR2 and MUX2) inside the PLB. However, it is unclear whether incorporating macrogates with wide inputs inside PLBs is beneficial. To answer this question, we need to first determine the logic functions of the macrogates, and then develop a flexible synthesis flow for architecture evaluation. The first contribution of this paper is to propose a methodology to extract a small set of logic functions that can implement a large portion of functions for given FPGA applications. Assuming that the extracted logic functions are implemented by macrogates in a PLB, we design a heterogeneous PLB consisting of both LUTs and macrogates.

Effective and efficient synthesis tools are key enablers of the exploration of different architecture options. There are extensive studies (e.g., [14]–[19]) on synthesis for homogeneous PLBs, however, only limited research on synthesis for heterogeneous PLBs. Reference [8] proposed heuristics to speedup the technology mapping for homogeneous PLBs and then extended them to consider heterogeneous PLBs with mixed LUT sizes. Reference [20] and [21] integrated Boolean Satisfiability (SAT) solvers into resynthesis to deal with macrogates in heterogeneous PLBs. However, high time complexity prohibits exploring complicated heterogeneous PLBs. The second contribution of this paper is to develop an efficient and flexible synthesis flow for heterogeneous PLBs. The flow includes a cut-based delay-optimal technology mapping, a mixed binary integer and linear programming (MBILP)-based postmapping area recovery algorithm to balance utilization of macrogates and LUTs, and a SAT-based PLB architecture-aware packing. Note that the traditional physical design flow is applicable to the proposed architecture because we assume that the structures of all PLBs are identical.

The rest of this paper is organized as follows. Section II presents a methodology to design heterogeneous PLBs with mixed LUTs and macrogates. Section III describes the synthesis algorithms to deal with the proposed FPGA architecture. This paper is concluded in Section IV. To the best of our knowledge, this paper is the first systematic study of the synthesis flow for FPGAs consisting of heterogeneous PLBs with wide-input macrogates. A detailed version of this paper can be found in our technical report [24].

### II. HETEROGENEOUS PLB DESIGN

The key step in designing a heterogeneous PLB is to extract a small set of logic functions that are able to implement a large portion of functions in given FPGA applications. In this section, we discuss how to extract such a set of logic functions by performing logic function ranking, and then present our macrogate design.

#### A. NCD

**Definition 1: (NPN Equivalence)** Two Boolean functions,  $F$  and  $G$ , belong to the same NPN-class (NPN-equivalent) if  $F$  can be derived from  $G$  by negating (N) and permuting (P) inputs and negating (N) the output [22].

**Definition 2: (Inheritance Equivalence)** For logic function  $A = F_1(a_1, \dots, a_n)$  and  $B = F_2(b_1, \dots, b_m)$ , where  $0 \leq m < n$ .  $A$  is

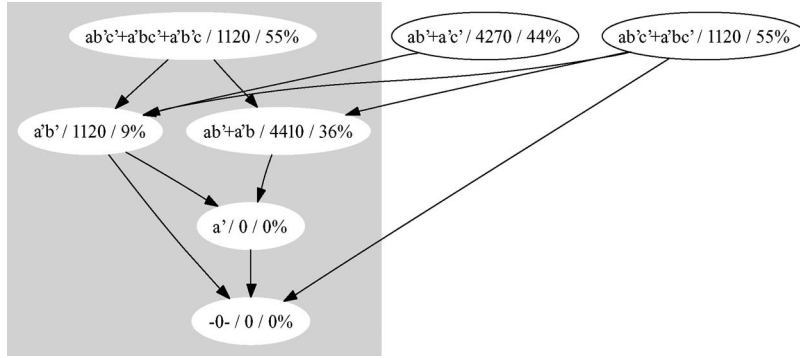


Fig. 1. Example UND.

inheritance equivalent to  $B$  iff  $\exists a_i$ , s.t.  $A(a_1, \dots, a_i = 1, \dots, a_n)$  or  $A(a_1, \dots, a_i = 0, \dots, a_n)$  and  $B$  are NPN-equivalent.

To graphically organize the logic functions and represent their NPN-Equivalence and Inheritance Equivalence relationships, we propose the following NPN-Class Diagram (NCD).

**Definition 3:** NCD is a directed acyclic graph (DAG), where each node in level  $N$  is the representative<sup>1</sup> of a  $N$ -input NPN-Class, and each edge from node  $A$  to node  $B$  indicates that function  $A$  is inheritance equivalent to  $B$ , i.e.,  $A$  can implement  $B$ 's functionality.

Note that Inheritance Equivalence is asymmetric, e.g.,  $f_1$  is inheritance equivalent to  $f_2$  but the reverse is not true. Therefore, NCD is a DAG as no edge is from lower level node to high-level node.

## B. UND

To extract  $N$ -input macrogate logic functions from the training FPGA application set, we first map those applications by exclusive LUT- $N$  architecture and then analyze all logic functions that are mapped into LUTs. Note that NCD stores all different function categories and their relationship within a DAG, which makes it extremely efficient to explore some interesting properties of an application. To represent the functions implemented by each particular application based on NCD, we present the Utilization NCD (UND), which is a subgraph of NCD in addition to the weights associated with each node. UND can be defined recursively as follows.

**Definition 4: (UND)** For a particular application  $D$ ,  $UND_{U_D}$  of  $D$  is a subgraph of NCD. If a function  $F$  is implemented by at least one of the LUTs in  $D$ , the NCD node that corresponds to the NPN-Class of  $F$  should be added into  $U_D$ . If an NCD node is present in  $U_D$ , all of its fan-out nodes and edges should be added into  $U_D$  recursively. Each node is associated with a three-tuple  $\Phi(f, n, c)$ , where  $f$  is the functionality description of this NPN-Class, and  $n$  and  $c$  are the implemented frequency (IF) and implementation capability (IC, will be defined later), respectively.

**Definition 5: (IF)** For an NPN-Class function  $f$  presented in a mapped application, the implemented frequency  $IF_f$  of logic function  $f$  is the number of LUTs which implement logic function  $f$ .

**Definition 6: (IC)** For an NPN-Class function  $f$  presented in the UND of a mapped application, the implementation capability  $IC_f$  of  $f$  is calculated by the following equation:

$$IC_f = \frac{\sum_{\forall v \in \text{fan-out cone of } f} IF_v}{\sum_{\forall u \in \text{UND}} IF_u}. \quad (1)$$

Intuitively,  $IC_f$  of NPN-Class function  $f$  indicates the portion (in terms of percentage) of logic functions in the application that can be

<sup>1</sup>The representative (canonical form) of a  $N$ -input NPN-Class can be selected based on different rules, but the NCD is always applicable.

implemented by  $f$ . Fig. 1 shows the UND for a small application that is mapped by LUT-3. There are a total of 12 040 functions implemented by LUTs in this application, and only three three-input NPN-classes are presented. An interesting observation from Fig. 1 is that the IC for function  $g = ab'c + a'bc' + a'b'c$  is 55% while its IF is only 1120 (less than 10% of 12 040 total functions). In fact, it has a child node  $ab' + a'b$  whose IF is 4410 (36% of 12 040 total functions), if we look into the fan-out cone (shadowed area) of node  $ab'c + a'bc' + a'b'c$ .

## C. UND for Wide Input Functions

To explore less-than-five-input functions, we can build a four-input NCD once and use a LUT to store the NPN-class of every logic function. For each of the training applications that are mapped by LUT-4, we can find its NPN-class from the LUT and create or label the node in UND. However, this procedure becomes prohibitively expensive since we cannot afford to preconstruct NCD for more than four-input functions by exhaustively examining NPN-equivalence for even all five-input functions ( $2^{2^5} = 4\,294\,967\,296$ ).

Practically, one can build a *partial* UND by online checking NPN-equivalence for over-five-input functions, which can be performed efficiently by the method proposed in [22]. When a new node representing a  $N$ -input ( $N \geq 5$ ) NPN-class function is inserted in the partial UND, only those  $(N - 1)$ -input functions that are inheritance equivalent to it are inserted/labeled in the partial UND, instead of performing insertion recursively. Partial UND is a good approximation of UND and all methods for UND manipulation are applicable for partial UND. In experiments, we find that the total number of all six-input NPN-classes presented in all IWLS'05 benchmarks [23] is less than 5000, and only 167 of them are present in more than 1% of all functions. Therefore, the size of partial UND can be well controlled in practice.

## D. Macrogate Design

The logic function extraction and ranking framework proposed above is implemented by a mix of Perl and C on the Berkeley ABC [5] platform. We employ IC as the metric to rank logic functions in UND.<sup>2</sup> Using IWLS'05 benchmarks [23] as the training set, the following six-input NPN-classes with the highest ranks are found as the candidates of the macrogates to be added into the FPGA PLB.

$$\begin{aligned} g_1(a, b, c, d, e, f) &= abcdef \\ g_2(a, b, c, d, e, f) &= ab'c' + bcf + bc'd + b'ce \\ g_3(a, b, c, d, e, f) &= ab'cd'e + bcef + def \\ g_4(a, b, c, d, e, f) &= ab' + a'cd' + b'c' + e' + f' \end{aligned} \quad (2)$$

<sup>2</sup>Please refer to [24] for a detailed analysis of function ranking metrics.

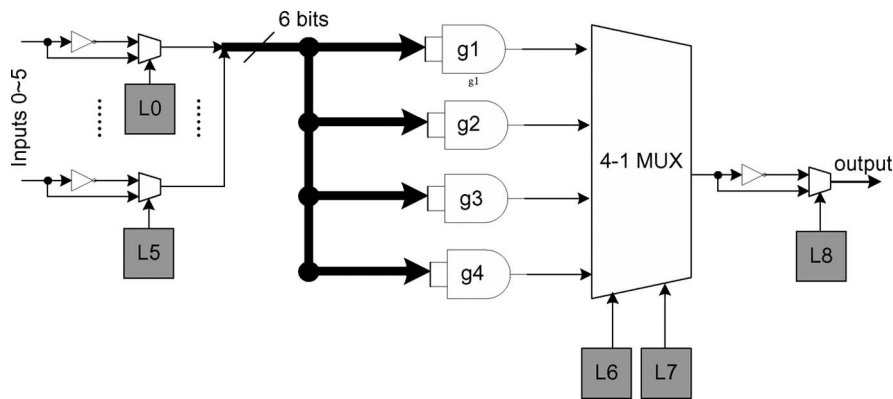


Fig. 2. Architecture of the macrogate.

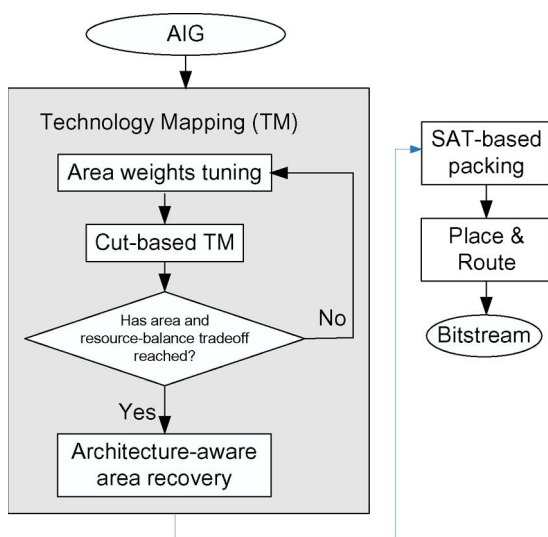


Fig. 3. Overall synthesis flow.

Combining these four gates with input/output negation, we can implement 23% six-input functions, 34% five-input functions, 60% four-input functions, 69% three-input functions, and 98% two-input functions on average for IWLS'05 benchmarks. Overall, 50% functions can be implemented by this macrogate. The structure of the macrogate is shown in Fig. 2.

### III. SYNTHESIS ALGORITHMS

#### A. Overall Synthesis Flow

For the proposed heterogeneous FPGA with mixed macrogates and LUTs, we present the following synthesis flow, as shown in Fig. 3. Given a gate level netlist of an application described by an And-Inverter Graph, we first perform technology mapping to map the application into LUTs and macrogates. The technology mapping phase includes multiple steps to minimize the delay and balance the resource utilization for LUTs and macrogates. The traditional cut-based technology mapping [8] is adapted to handle macrogates. Basically, the functionalities that can be implemented by a macrogate are precalculated and stored in a LUT, and a feasibility checking is performed to decide if a cut can be mapped to a macrogate. We also propose an efficient heuristic to speedup the cut enumeration process by pruning potentially infeasible cuts based on the cofactors of the macrogate's functionalities [24]. The details of the technology mapping and postmapping area recovery techniques will be presented

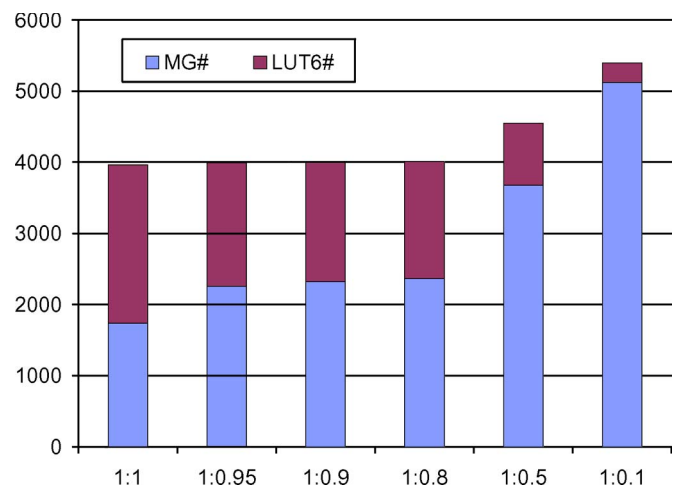


Fig. 4. Impact of area weight on LUT-MG ratio.

in Section III-B. After mapping, we pack the mapped application into logic blocks. The packer is based on satisfiability and is highly flexible, which is generally applicable to heterogeneous FPGAs. Please refer to our technical report [24] for the details of the proposed packing algorithm. Finally, we perform the placement and routing. Note that we assume the use of homogeneous logic blocks, therefore the traditional physical design algorithms, e.g., VPR [1], can be extended to handle the proposed new architecture, and they are not discussed in this paper.

#### B. Technology Mapping

The unique challenge of the technology mapping for the proposed architecture is that the utilization of different hardware resource (i.e., LUTs and macrogates) must be well balanced to reduce both logic area and interconnect delay. Given a PLB architecture, the tightest packing is obtained when the ratio between the numbers of LUTs and macrogates in the mapped circuit is equal to the available LUTs and macrogates in a PLB. For example, suppose the target architecture has one LUT and one macrogate within a PLB, i.e., the LUT-MG ratio is 1 : 1. If we can achieve the same ratio in the mapped result, a tight packing is expected.

In fact, we can tune the area weights assigned to LUTs and macrogates in the technology mapper, which shows significant impact in the LUT-MG ratio of the mapped result. Fig. 4 shows the average number of LUTs and macrogates in the mapped results for 20 IWLS'05 benchmarks with different area weight assignments for a six-input

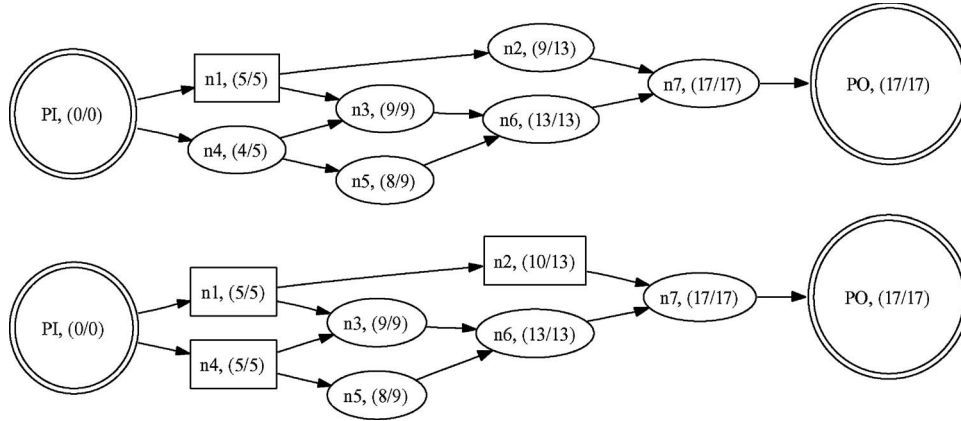


Fig. 5. Circuit before and after area recovery, where an eclipse denotes a macrogate and a rectangle denotes LUT. Arrival/required times are labeled in each node. After area recovery, the application can be packed into three PLBs compared to six PLBs before, assuming a PLB contains one LUT and one macrogate.

LUT and a six-input macrogate, i.e., 1 : 1, 1 : 0.95, 1 : 0.9, 1 : 0.8, 1 : 0.5, and 1 : 0.1. As shown in Fig. 3, to achieve a tight packing for the target architecture with LUT-MG ratio 1 : 1, we first perform a binary search to find the best area weight assignment which gives a small  $N$  (the total number of LUTs and macrogates) and a  $\alpha$  (the ratio between the number of LUTs and macrogates) close to the target LUT-MG ratio. From Fig. 4, we see that  $N$  increases dramatically if an extremely small weight is assigned to macrogates (see the bar for 1 : 0.5 and 1 : 0.1) since the mapper is biased. We find that 1 : 0.95, 1 : 0.9, and 1 : 0.8 all give good tradeoffs between  $N$  and  $\alpha$  under the target LUT-MG ratio. However, it is hard to further improve the resulting LUT-MG ratio  $\alpha$  by simply adjusting the area weight. To further balance the resource utilization, we perform an *architecture-aware area recovery* as follows.

Given the mapped result after binary search, the total number of LUTs and macrogates  $N$  and delay target  $T$  are fixed. Motivated by timing slack budgeting [25], we can reassign macrogates and LUTs in such a way that the resulting LUT-MG ratio is sufficiently close to the target LUT-MG ratio while preserving the delay target. The combinational portion of the mapped result is represented in a DAG  $G = (V, E)$ , where  $\forall v_j \in V$  is mapped to an LUT or a macrogate. Without loss of generality, we assume that the intrinsic delay of an LUT is  $\Delta D$  larger than a macrogate. In this case, the binary search should be able to find an LUT-MG ratio  $\alpha$ , which is slightly smaller than the target LUT-MG ratio  $\beta$  (i.e., the ratio between the number of LUTs and macrogates in a PLB is  $\beta$ ). To balance the number of LUTs and macrogates according to  $\beta$ , certain macrogates should be remapped as LUTs. For each node  $v_j$  that is mapped as a macrogate, if its input number is not larger than the LUT size, it can be remapped as an LUT without changing the functionality. All such nodes are stored in set  $V_m$ . A binary variable  $m_j$  indicates if node  $v_j$  can be remapped as an LUT for the given timing slack  $b_j$ . The objective is to minimize the gap between the mapped macrogate number  $\sum_{\forall v_j \in V_m(G)} m_j$  and the ideal number  $(1/(1+\beta)) \cdot N$ . The overall problem can be formulated as an MBILP as follows:

$$\min \left| \sum_{\forall v_j \in V_m(G)} m_j - \frac{1}{1+\beta} \cdot N \right| \quad (3)$$

$$\text{s. t.} \quad m_j \leq \frac{b_j}{\Delta D} \quad \forall v_i \in V_m(G) \quad (4)$$

$$m_j \in \{0, 1\} \quad \forall v_i \in V_m(G) \quad (5)$$

$$a_i + d_j + b_j \leq a_j \quad \forall e(i, j) \in E(G) \quad (6)$$

$$b_i \geq 0 \quad \forall v_i \in V(G) \quad (7)$$

$$a_i \leq T \quad \forall v_i \in V(G) \quad (8)$$

where  $a_i$  and  $d_j$  are the arrival time and intrinsic delay for node  $v_i$ , respectively.

Note that the absolute operator in the objective function can be easily transformed into linear form by introducing an auxiliary variable  $t$  as follows:

$$\min \quad t \quad (9)$$

$$\text{s. t.} \quad \sum_{\forall v_j \in V_m(G)} m_j - \frac{1}{1+\beta} \cdot N \leq t$$

$$\sum_{\forall v_j \in V_m(G)} m_j - \frac{1}{1+\beta} \cdot N \geq -t$$

$$t \geq 0. \quad (10)$$

Fig. 5 shows an example of the area recovery algorithm. A generalization of the above formulation can be found in [24] to deal with multiple types of macrogates.

### C. Experiments and Discussions

We have conducted our synthesis flow, including technology mapping, area recovery, and packing, on 70 industrial benchmark applications. Fig. 6 compares the logic depth achieved by the mix of LUT-4 and smaller macrogates (XOR2 and MUX2) and the mix of LUT-4 and wider macrogates (AND4 and MUX4). It indicates that the heterogeneous architecture with wider macrogates has 6% on average (and up to 42%) less logic depth compared to the architecture with smaller macrogates. This observation again verifies the necessity and potential effectiveness of the proposed study for the mixed LUTs and wide macrogates.

To show the effectiveness of the proposed technology mapping flow (as shown by Phase I in Fig. 3), we compare it with an alternative flow, which does not perform the area weight setting and area recovery. First, we show that the area weight cannot be arbitrarily set before technology mapping. Recall Fig. 4, which is the average number of LUTs and macrogates in the mapped results for 20 IWLS'05 benchmarks with different area weight assignments for a six-input

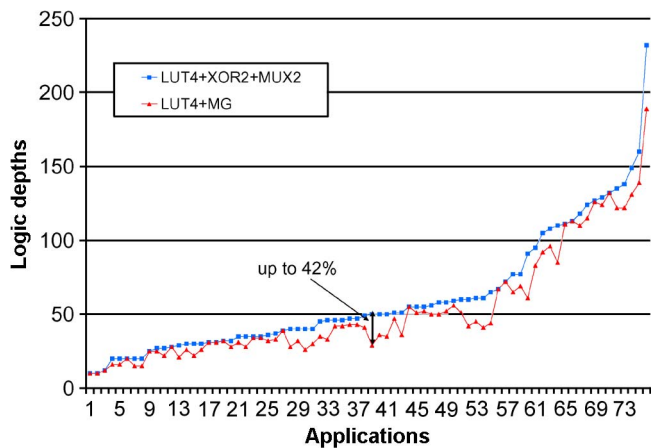


Fig. 6. Logic depths of heterogeneous architectures with small and wide macrogates over 70 industrial benchmark applications used in this paper.

LUT and a six-input macrogate. A liberal setting of the area weight may cause low utilization rate (e.g.,  $\beta = 1 : 1$  in Fig. 4) or significant area increase (e.g.,  $\beta = 1 : 0.1$  in Fig. 4). Therefore, we need to carefully select a set of area weight assignments, e.g.,  $1 : 0.95$ ,  $1 : 0.9$ , and  $1 : 0.8$  in Fig. 4, which give good tradeoffs between  $N$  and  $\alpha$  under the target architecture specification.

Moreover, the architecture-aware area recovery subphase is effective to calibrate the resource utilization in a fine granularity. We conduct experiments by mapping 20 IWLS'05 applications into an architecture with one LUT6 and one macrogate in a PLB. The number of PLBs is reduced by 5% due to the area recovery. Note that ten applications achieve perfect resource utilization (resource demand is  $1 : 1$  for LUTs and macrogates) after the area recovery, as shown in [24].

#### IV. CONCLUSION AND DISCUSSION

Targeting macrogate-based heterogeneous FPGAs, a methodology has been proposed to extract a small set of logic functions that are able to implement a large portion of functions for given FPGA applications. Assuming that the extracted logic functions are implemented by macrogates in PLBs, a flexible synthesis flow has been developed for such heterogeneous PLBs with mixed LUTs and macrogates. The flow includes a cut-based delay-optimal technology mapping, an MBILP-based postmapping area recovery to balance the utilization of macrogates and LUTs, and a SAT-based PLB architecture-aware packing. Experiments using over 70 industrial benchmark applications show that we can extract four six-input logic functions to cover more than 50% functions of these applications, and the proposed synthesis flow reduces area by 5% compared to an alternative flow without the postmapping area recovery when both have the optimal logic depth. Compared to the PLB with mixed LUT-4 and small macrogates (XOR2 and MUX2), the PLB with mixed LUT-4 and four-input macrogate reduces logic depth by 6% (and up to 42%) for the aforementioned applications.

Architecture evaluation depends on both synthesis and physical design as well as detailed area, delay, and power models for PLBs and interconnects. While the synthesis has been developed in this paper, the existing physical design flow such as VPR [1] can be adopted to evaluate the proposed PLBs, if such PLBs are used for all logic blocks. Architecture evaluation of the proposed PLB will be our future work. Particularly, we will consider the impact of not allowing full pin-permutation in the macrogate. In contrast, all input pins in an LUT are equivalent and can be freely permuted. This leads to more

routing flexibility compared to the case for the macrogate. We will also design pin-permutation-aware routing algorithms to compensate this limitation.

#### REFERENCES

- [1] V. Betz, J. Rose, and A. Marquardt, *Architecture and CAD for Deep-Submicron FPGAs*. Norwell, MA: Kluwer, Feb. 1999.
- [2] *The Programmable Gate Array Data Book*, Xilinx, San Jose, CA, 1989.
- [3] J. Cong, H. Huang, and X. Yuan, "Technology mapping and architecture evaluation for k/m-macrocell-based FPGAs," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 10, no. 1, pp. 3–23, 2005.
- [4] *Xilinx Product Datasheets*, Xilinx, San Jose, CA. [Online]. Available: <http://www.xilinx.com/literature>
- [5] *ABC: A System for Sequential Synthesis and Verification*. [Online]. Available: <http://www.eecs.berkeley.edu/~alanmi/abc/>
- [6] A. Cosoroaba and F. Rivoallon, *Achieving Higher System Performance with the Virtex-5 Family of FPGAs*. [Online]. Available: <http://www.xilinx.com/literature>
- [7] J. Cong and S. Xu, "Delay-optimal technology mapping for FPGAs with heterogeneous LUTs," in *Proc. Des. Autom. Conf.*, Jun. 1998, pp. 704–707.
- [8] J. Cong, C. Wu, and Y. Ding, "Cut ranking and pruning: Enabling a general and efficient FPGA mapping solution," in *Proc. ACM Int. Symp. Field-Programmable Gate Arrays*, 1999, pp. 29–35.
- [9] J. He, "Technology mapping and architecture of heterogeneous field-programmable gate arrays," M.S. thesis, Univ. Toronto, Toronto, ON, Canada, 1993.
- [10] A. Kaviani and S. Brown, "Hybrid FPGA architecture," in *Proc. ACM Int. Symp. Field-Programmable Gate Arrays*, 1996, pp. 1–7.
- [11] K. Chung, "Architecture and synthesis of field-programmable gate arrays with hard-wired connections," Ph.D. dissertation, Univ. Toronto, Toronto, ON, Canada, 1994.
- [12] S. Wilton, "SMAP: Heterogeneous technology mapping for area reduction in FPGAs with embedded memory arrays," in *Proc. ACM Int. Symp. Field-Programmable Gate Arrays*, 1998, pp. 171–178.
- [13] J. Cong and S. Xu, "Technology mapping for FPGAs with embedded memory blocks," in *Proc. ACM Int. Symp. Field-Programmable Gate Arrays*, 1998, pp. 179–188.
- [14] J. Cong and Y. Ding, "Flowmap: An optimal technology mapping algorithm for delay optimization in lookup-table based FPGA designs," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 13, no. 1, pp. 1–12, Jan. 1994.
- [15] J. Cong and Y. Ding, "On area/depth trade-off in LUT-based FPGA technology mapping," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 2, no. 2, pp. 137–148, Jun. 1994.
- [16] D. Chen and J. Cong, "DAOmap: A depth-optimal area optimization mapping algorithm for FPGA designs," in *Proc. Int. Conf. Comput.-Aided Des.*, 2004, pp. 752–759.
- [17] V. Manohara-rajah, S. D. Brown, and Z. G. Vranesic, "Heuristics for area minimization in LUT-based FPGA technology mapping," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 25, no. 11, pp. 2331–2340, Nov. 2006.
- [18] A. Mishchenko, S. Chatterjee, R. Brayton, and P. Pan, "Integrating logic synthesis, technology mapping, and retiming," in *Proc. Int. Workshop Logic Synthesis*, 2005, pp. 161–168.
- [19] S. Chatterjee, A. Mishchenko, and R. Brayton, "Factor cuts," in *Proc. Int. Conf. Comput.-Aided Des.*, 2006, pp. 143–150.
- [20] A. Ling, D. Singh, and S. Brown, "FPGA logic synthesis using quantified Boolean satisfiability," in *Proc. SAT*, 2005, vol. 3569, pp. 444–450.
- [21] S. Safarpour, A. Veneris, G. Baeckler, and R. Yuan, "Efficient SAT-based Boolean matching for FPGA technology mapping," in *Proc. Des. Autom. Conf.*, 2006, pp. 466–471.
- [22] D. Chai and A. Kuehlmann, "Building a better Boolean matcher and symmetry detector," in *Proc. Des. Autom. Test Conf. Eur.*, 2006, pp. 1079–1084.
- [23] *IWLS 2005 Benchmarks*. [Online]. Available: <http://iwls.org/iwls2005/benchmarks.html>
- [24] Y. Hu, S. Das, S. Trimberger, and L. He, *Design, synthesis and evaluation of heterogeneous FPGA with mixed LUTs and macrogates*, 2007. Tech. Rep. UCLA Engr 07-264. [Online]. Available: [http://www.ee.ucla.edu/~hu/pub/techReport07\\_fpga\\_arch.pdf](http://www.ee.ucla.edu/~hu/pub/techReport07_fpga_arch.pdf)
- [25] S. C. Soheil Ghiasi, E. Bozorgzadeh, and M. Sarrafzadeh, "A unified theory of timing budget management," in *Proc. Int. Conf. Comput.-Aided Des.*, Nov. 2004, pp. 653–659.