

Design, Synthesis and Evaluation of Heterogeneous FPGA with Mixed LUTs and Macro-Gates *

Yu Hu^{1,2}, Satyaki Das², Steve Trimberger² and Lei He¹
Electrical Engineering Department, UCLA, Los Angeles, CA 90095¹
Research Laboratory, Xilinx Inc., 2100 Logic Dr. San Jose, CA 95124²

ABSTRACT

Small gates, such as AND2, XOR2 and MUX2, have been mixed with lookup tables (LUTs) inside the programmable logic block (PLB) to reduce area and power and increase performance in FPGAs. However, it is unclear whether incorporating macro-gates with wide inputs inside PLBs is beneficial. In this paper, we first propose a methodology 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 macro-gates in PLBs, we then develop a complete synthesis flow for such heterogeneous PLBs with mixed LUTs and macro-gates. The flow includes a cut-based delay and area optimized technology mapping, a mixed binary integer and linear programming based area recovery algorithm to balance the resource utilization of macro-gates and LUTs for area-efficient packing, and a SAT-based packing. We finally evaluate the proposed heterogeneous FPGA using the newly developed flow and show that mixing LUT and macro-gates, both with 6 inputs, improves performance by 16.5% and reduces logic area by 30% compared to using merely 6-input LUTs.

1. 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 macro-gate (e.g. AND gates and multiplexers) [4]. These logic elements offer a spectrum of trade-offs between functionality and costs in terms of area, power and delay. For instance, a circuit can be implemented by fewer K -input LUTs than K -input macro-gates, while a K -input macro-gate 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 study the impact of heterogeneous PLBs.

Recent work has shown when uniform LUTs are used for the PLB, a larger LUT size increases performance [5, 6], but it reduces the LUT pin utilization rate. For example, mapping IWLS'05 benchmarks [7] using 6-input LUTs and the Berkeley ABC mapper [8], we find that over 60% of the LUTs use less than five inputs. Initial studies in the literature have suggested that mixed-sized LUTs [5, 9, 10, 26], mixing LUTs and PLAs [11] inside the PLB, or PLBs with hard-wired connections [27] may improve logic density. In addition, commercial FPGAs [5] have benefited from small macro-

gates (e.g., XOR2 and MUX2) inside the PLB. However it is unclear whether incorporating macro-gates with wide inputs inside PLB is beneficial. The first contribution of this paper is to propose a methodology 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 macro-gates in PLB, we design a heterogeneous PLB consisting of both LUTs and macro-gates.

Effective and efficient synthesis tools are key enablers for the exploration of different architecture options. There are extensive studies (e.g., [12, 13, 14, 15, 16, 17]) on synthesis for homogeneous PLBs, but only limited research on synthesis for heterogeneous PLBs. [10] proposed heuristics to speedup the technology mapping for homogeneous PLBs and then extended them to consider heterogeneous PLBs with mixed LUT sizes. [18, 19] integrated Boolean Satisfiability solvers into re-synthesis to deal with macro-gates in heterogeneous PLBs. However, the high time complexity prohibits exploring complicated heterogeneous PLBs. The second contribution of this work is to develop an efficient logic synthesis flow for heterogeneous PLBs. The flow includes a cut-based delay-optimal technology mapping, a mixed binary integer and linear programming (MBILP) based area recovery algorithm to balance the resource utilization of macro-gates and LUTs for area-efficient packing, and a SAT-based packing. Using the newly developed flow, we evaluate the proposed heterogeneous PLB, and show that mixing LUT and macro-gates, both with 6 inputs, improves performance by 16.5% and reduces logic area by 30% for IWLS'05 benchmarks [7] when compared to using 6-input LUTs.

The rest of this paper is organized as follows. Section 2 presents the methodology to design heterogeneous PLBs with mixed LUTs and macro-gates. Section 3 describes the improved technology mapping and post-mapping area recovery algorithms to deal with the proposed FPGA architecture. Section 4 proposes a flexible SAT-based packing algorithm. The architecture evaluation results are given in Section 5, and the paper is concluded in Section 6. To the best of our knowledge, this paper is the first systematic study of the logic synthesis flow for FPGAs consisting of heterogeneous PLBs with wide-input macro-gates.

2. 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 macro-gate design.

2.1 Preliminaries

To rank the logic functions extracted from the training FPGA

*This paper is partially supported by NSF grant CCR-0306682. Address comments to lhe@ee.ucla.edu.

covering (MFC), we propose the following theorem to calculate the MFC very efficiently.

THEOREM 1. *The number of minimal functional covering (MFC) in application D is equal to the number of all primary input nodes (with no parent nodes) of UND for D .*

The proof of this theorem is straightforward.

2.3 UND for Wide Input Functions

To explore functions with fewer than 5 inputs, we can build 4-input NCD once and use a lookup table 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 lookup table and create or label the node in UND. However, this procedure becomes prohibitively expensive since we cannot afford to pre-construct NCD for more than 4-input functions by exhaustively examining NPN-equivalence for even all 5-input functions ($2^{2^5} = 4294967296$).

Practically, one can build a *partial UND* by online checking NPN-equivalence for over-5-input functions, which can be performed efficiently by the method proposed in [20]. 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 6-input NPN-classes presented in all IWLS'05 benchmarks [7] is less than 5000 and only 167 of them are present more than 1% out of all functions. Therefore, the size of partial UND can be well controlled in practice.

2.4 Macro-Gate Design

The logic function extraction and ranking framework proposed above is implemented by the mix of Perl and C on the Berkeley ABC [8] platform. Using IWLS'05 benchmarks [7] as the training set, the following 6-input NPN-classes with the highest ranks are found as the candidates of the macro-gates to be added into the FPGA PLB.

$$g_1(a, b, c, d, e, f) = abcdef \quad (2)$$

$$g_2(a, b, c, d, e, f) = ab'c' + bcf + bc'd + b'ce \quad (3)$$

$$g_3(a, b, c, d, e, f) = ab'cd'e + bcef + def \quad (4)$$

$$g_4(a, b, c, d, e, f) = ab' + a'cd' + b'c' + e' + f' \quad (5)$$

Combining these four gates with input/output negation, we can implement 23% 6-input functions, 34% 5-input functions, 60% 4-input functions, 69% 3-input functions and 98% 2-input functions on average for IWLS'05 benchmarks. Overall 50% functions can be implemented by this macro-gate. The structure of the macro-gate is shown in Figure 3. In our experiments, we incorporate this macro-gate into the FPGA PLBs and assume that there exist one LUT and one such macro-gate in each PLB.

3. TECHNOLOGY MAPPING

Given the functions of the macro-gates to be built into FPGA PLBs, a technology mapper is needed to make full use of these macro-gates in terms of performance improvement and area reduction. [18] presented a general algorithm for heterogeneous FPGA technology mapping, but it is not scalable to large macro-gates. [17] extended the traditional cut-based technology mapping algorithm

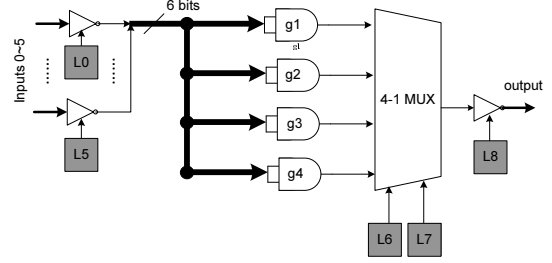


Figure 3: Architecture of the macro-gate

[10] (for LUT-only FPGAs) to achieve delay optimality for macro-gate based FPGAs² and proposed a novel concept, i.e., factor cuts, to improve the scalability of the algorithm. In [17], the truth table of the NPN-classes that can be implemented by the macro-gates (with M inputs) are pre-calculated and stored in a lookup table. Then the mapping is done in two passes. The first pass is a forward topology traversal for enumerating all M -feasible cuts and keeping only those that can be found in the lookup table storing the truth tables of macro-gates. The second pass is a backward topology traversal to select the best cut for every node under the delay target and area constraints. For technology mapping to heterogeneous PLBs with both LUTs and macro-gates, we adopt the same framework presented in [17] by labeling cuts as macro-gates or LUTs based on their logic functions.

After the cut-based mapping, an area recovery phase is needed for the proposed macro-gate based heterogeneous FPGA architecture. For the target PLB with C_L LUTs and C_M macro-gates, suppose there are N_L LUTs and N_M macro-gates in the mapped circuit, the logic area (i.e., the number of PLBs) after the ideal packing should be

$$\begin{aligned} \Phi(N, \alpha) &= \max\left(\frac{N_M}{C_M}, \frac{N_L}{C_L}\right) \\ &= \frac{N}{C_L} \cdot \max\left(\frac{\alpha}{\beta}, 1 - \alpha\right) \end{aligned}$$

where $N = N_M + N_L$, $\beta = C_M/C_L$ and $\alpha = N_M/N$.

Since C_L and β are constant for a given architecture, it is easy to show

$$\min_{\alpha} \Phi(N, \alpha) = \frac{N}{C_L} \cdot \frac{1}{1 + \beta} \quad (6)$$

$$\alpha^* = \frac{\beta}{1 + \beta} \quad (7)$$

For a given architecture, (6) provides a lower bound of the number of PLBs for a mapped result, where parameters N and α decide the tightness of the packing. (7) shows that the tightest packing can be achieved when the LUT-MG ratios³ of the given architecture ($\frac{\beta}{1+\beta}$) and the mapped result (α) are equal.

For example, suppose the target architecture has one LUT and one macro-gate 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 adjust the area weight assigned to LUTs and macro-gates in the technology mapper, which shows significant impact in the LUT-MG ratio of the mapped result. Figure 4 shows the average number of LUTs and macro-gates in the mapped results for 20 IWLS'05 benchmarks with different area weight assignments for a 6-input LUT and a 6-input macro-gate, i.e., 1:1, 1:0.95, 1:0.9,

²The macro-gate in [17] was built by three LUTs (without the logic gates) and the homogeneous PLBs were assumed.

³LUT-MG ratio is the number of LUTs divided by the number of macro-gates in an application.

1:0.8, 1:0.5 and 1:0.1. The LUT-MG ratio can be clearly seen in this figure. 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 small N and α close to the target LUT-MG ratio. From Figure 4, we see that the total number of LUTs and macro-gates (N in (6)) increases dramatically if an extremely small weight is assigned to macro-gates (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 trade-offs between N and α under the target LUT-MG ratio but it is hard to further improve the resulting LUT-MG ratio, β , by simply adjusting area weight.

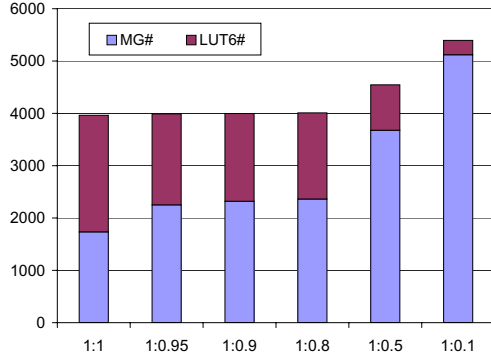


Figure 4: Impact of area weight on LUT-MG ratio

Given the mapped result after binary search, the total number of LUTs and macro-gates, N , and delay target, T , are fixed. Motivated by the timing slack budgeting [21], we can reassign macro-gates 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 a LUT or a macro-gate. Without loss of generality, we assume that the intrinsic delay of a LUT is ΔD larger than a macro-gate. In this case, the binary search should be able to find a LUT-MG ratio, α , which is slightly smaller than the target LUT-MG ratio, $\beta/(1 + \beta)$. To balance the number of LUTs and macro-gates according to β , certain macro-gates should be re-mapped as LUTs. For each node v_j that is mapped as a macro-gate, if its input number is not larger than the LUT size, it can be re-mapped as a 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 re-mapped as a LUT for the given timing slack b_j . The objective is to minimize the gap between the mapped macro-gate number $\sum_{\forall v_j \in V_m(G)} m_j$ and the ideal number $\frac{1}{1+\beta} \cdot N$. The overall problem can be formulated as a mixed binary integer and linear programming (MBILP) as follows.

$$\min \quad \left| \sum_{\forall v_j \in V_m(G)} m_j - \frac{1}{1+\beta} \cdot N \right| \quad (8)$$

$$\text{s.t.} \quad m_j \leq \frac{b_j}{\Delta D}, \forall v_i \in V_m(G) \quad (9)$$

$$m_j \in \{0, 1\}, \forall v_i \in V_m(G) \quad (10)$$

$$a_i + d_j + b_j \leq a_j, \forall e(i, j) \in E(G) \quad (11)$$

$$b_i \geq 0, \forall v_i \in V(G) \quad (12)$$

$$a_i \leq T, \forall v_i \in V(G) \quad (13)$$

where a_i and d_j are the arrival time and intrinsic delay for node v_i , respectively. All the other variables have been explained above. Note that the absolute operator in the objective function can be easily transformed into linear form by introducing an auxiliary variable t

as follows.

$$\min \quad t \quad (14)$$

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

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

$$t \geq 0 \quad (15)$$

4. SAT BASED FLEXIBLE PLB PACKER

Given the mapped application, the following step is to pack the logic elements, such as LUTs, FFs and macro-gates, into FPGA PLBs. The typical existing packing tools such as T-VPack [1] hard-codes the architecture specification of a PLB, a step requiring rewriting from scratch if the PLB architecture changes. This is time consuming and needs to be debugged carefully to avoid bugs. The flow proposed in [18] and [19] tries to solve technology mapping and packing simultaneously. It is extremely flexible but impractical for typical applications due to its high computational cost.

In contrast to the existing work, we solve the problem of validating PLB packing as a local place and route problem at the PLB level. This can be considered a placement problem since multiple compatible sites may be present for each logic element in the cluster⁴. Also, for the chosen placement, all interconnections within the cluster must be routed in the PLB and all external connections must be connected to appropriate PLB input/output pins. We use a satisfiability (SAT) solver to carry out this PLB packing validation. The tool takes a netlist description of the target PLB, the user netlist, and the netlist elements clustered into the PLB to generate a set of Boolean equations such that each solution to these equations represents a valid packing. Then a SAT-solver can be used to find a solution to this set of equations. If a solution exists, the PLB packing is valid and an assignment of the netlist elements to locations in the PLB and PLB input or output pins to nets is produced. If no solution exists, the packing is invalid.

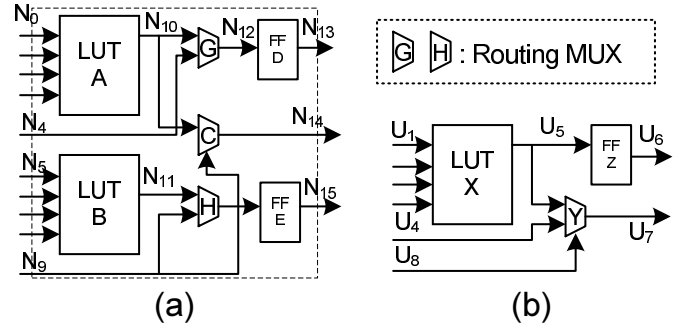


Figure 5: (a) Architecture of a FPGA PLB, (b) Sub-circuit to be packed

Now we briefly sketch the problem formulation with an example. Assume that we are attempting to fit the fragment described in Figure 5 (b) into the simplified Xilinx Spartan3-like PLB in Figure 5 (a). We have variables $X@A$, $X@B$ encoding placement of LUT X at either the site A or B . Now we generate the *exclusivity constraint* and *presence constraint* as follows.

$$(\neg X@A) \vee (\neg X@B) \quad (16)$$

$$X@A \vee X@B \quad (17)$$

Constraint (16) implies that if X is placed at A then it cannot be placed at B and vice versa. Constraint (17) means that X must be

⁴For instance in Xilinx Spartan3 [4] PLB there are two LUTs.

placed at at least one of the sites A and B . The two equations above guarantee that X will be placed at exactly one location. Similar constraints would be generated for the flip-flop Z and the possible locations D and E .

Nets are treated like logic element sites, which are encoded as Boolean variables, $U_i@N_j$ ($\forall i \in \{1 \dots 8\}, j \in \{0 \dots 15\}$). The mutual exclusion conditions apply to nets as well, so that distinct nets in the user netlist cannot be present at any one net in the PLB.

We also have *input and output constraints* for all logic elements. One such output constraint would be

$$X@A \rightarrow U_5@N_{10}, \quad (18)$$

which guarantees that if X is placed at location A in the PLB, then the N_{10} net in the PLB carries the U_5 signal. Input constraints are also generated in an analogous fashion.

For routing multiplexers (like G and H) which are controlled by configuration logic we have variables that determine which input drives the output. So for G , we would have the variables, $G_{0 \rightarrow \text{out}}$ and $G_{1 \rightarrow \text{out}}$, and we need to add *exclusive driver constraints* (19) so that no output is driven by two inputs. We also need to add *routing constraints* (20) so that if a net is present at an input to a routing multiplexer and the appropriate control bit is active, the output has the correct net.

$$G_{0 \rightarrow \text{out}} \rightarrow (\neg G_{1 \rightarrow \text{out}}) \quad (19)$$

$$(G_{0 \rightarrow \text{out}} \wedge U_5@N_{10}) \rightarrow U_5@N_{12} \quad (20)$$

Finally, extra variables are needed to encode the choice that some logic elements can act as route-throughs. An example of such an element is a lookup table. It can be programmed to pass one of its inputs out to the output and thus act like a routing multiplexer if it is not being used. This increases the routing flexibility if we need it. Corresponding to these extra variables, *route-through constraints* are generated. For example, some of the constraints generated for taking into consideration that LUT B might be used as a route-through are

$$(RT_B \wedge B_{0 \rightarrow \text{out}}) \rightarrow (\neg B_{1 \rightarrow \text{out}}) \quad (21)$$

$$(RT_B \wedge B_{0 \rightarrow \text{out}} \wedge U_8@N_5) \rightarrow U_8@N_{11} \quad (22)$$

where RT_B encodes the choice as to whether LUT B is used as a route-through. Constraint (21) maintains that inputs 0 and 1 cannot drive the output simultaneously. There will be other constraints corresponding to mutual exclusion of every pair of inputs. Constraint (22) specifies that if input 0 does drive the output and if net N_5 is at input 0, then it would also appear at the output. Note that constraint (21) is just a representative constraint and other constraints related to the other input pins and to other possible nets will also be generated as needed.

Certain input pins of logic elements are swappable, i.e., they may be interchanged if doing so helps in producing a valid PLB. Variables are introduced to encode the choice of equivalent pins that have been swapped and *equivalent pin constraints* corresponding to these are added.

Finally, *boundary constraints* forcing input (or output) nets to be assigned to at least one of the input (or output) pins are added. An example of such a constraint is

$$U_6@N_{13} \vee U_6@N_{14} \vee U_6@N_{15} \quad (23)$$

Once all the constraints are automatically generated from a topological description of the PLB graph and the logic elements in the user's netlist that we wish to pack into a PLB, we use a SAT-solver to check if they can all be satisfied simultaneously. If they are all satisfied, we can generate a valid placement and route from the solution

that the SAT solver gives us. Translating the solution to an assignment of netlist elements to PLB locations, PLB input pins to nets and control bits in the routing multiplexers is easy. We construct the PLB based on all the variables that are TRUE in the satisfying assignment. In our example, one satisfying assignment would have the following variables be TRUE: $U_1@N_0, U_2@N_1, \dots, X@A, RT_B$, where $U_1@N_0$ means that the net U_1 is at the PLB input N_0 , $X@A$ means that LUT X is placed at location A and RT_B means that LUT B is used as a route-through.

5. ARCHITECTURE EVALUATION

5.1 Area and Delay Modeling

Suggested in [3], a LUT-4 can be implemented by 16 1-bit SRAM, 4 inverters (INV1) and 15 2:1MUXes which are $16 \times 6 + 4 \times 2 + 15 \times 4 = 164$ transistors, and a LUT-6 can be implemented by 64 1-bit SRAM, 6 inverters and 63 2:1MUXes, which are $64 \times 6 + 6 \times 2 + 63 \times 4 = 648$ transistors. The macro-gate in Figure 3 can be implemented by 9 1-bit SRAM, 7 inverters, 3 2:1MUXes and 84 transistors (for implementing the macro-gate⁵), which are $9 \times 6 + 7 \times 2 + 3 \times 4 + 84 = 164$ transistors. The logic delay is modeled by the square root of the logic area. The area and delay models used in our experiments are summarized in Table 2.

	LUT-4	macro-gate	LUT-6
transistor number	164	164	648
normalized area	1.00	1.00	3.95
normalized delay	1.00	1.00	1.98

Table 2: Summary of the area and delay models

5.2 Experimental Results

We implement the improved mapping algorithms in the Berkeley ABC platform [8] with C language and the SAT-based packer with LISP language, and test them on IWLS'05 benchmarks [7]. The MBILP and SAT problem are solved by mosek [24] and miniSAT [25], respectively.

In our technology mapping, the combinational portion of an application is first mapped to LUTs and macro-gates, and the sequential elements are added back to the mapped sub-circuit. The mapped application is then packed by our SAT-based packer. After the packing, the logic cost is evaluated based on the aforementioned area and delay model. Note that the placement and routing are not performed because this work is focused on logic delay and area evaluation for the proposed macro-gate based heterogeneous FPGA architecture. The interconnect cost of such architecture is not considered here and will instead be evaluated in our future work.

Table 1 compares four architectures, i.e., a PLB containing a LUT-4 (column "LUT4"), the mix of a LUT-4 and a macro-gate in Figure 3 (column "LUT4+MG"), a LUT-6 (column "LUT6"), and the mix of a LUT-6 and a macro-gate (column "LUT6+MG"), respectively. The logic depth, number of PLBs, estimated delay and area⁶ of the circuits with different architectures are shown in the table. Compared to LUT-4 only architecture, the mix of LUT-4 and macro-gates reduces both logic depth and logic delay by 9.2%, and reduces logic area by 12.9%. Compared to LUT-6 only architecture, the mix of LUT-6 and macro-gates reduces logic delay and logic area by 16.5% and 30%, respectively, while it increases

⁵The embedded logic gates $g_1 \dots g_4$ of the macro-gate in Figure 3 are synthesized by SIS 1.2 [22] with batch command *script.rugged* followed by *read_library mcnc.genlib; map -m 0*, and their schematics are omitted due to the space limit

⁶Given areas in Table 2, the area ratio between these four PLBs is 1.00:(1.00+1.00):3.98:(3.98+1.00).

circuit	Logic depth				Estimated delay (normalized)				PLB#				Estimated area (normalized)			
	LUT4	LUT4 MG	LUT6	LUT6 MG	LUT4	LUT4 MG	LUT6	LUT6 MG	LUT4	LUT4 MG	LUT6	LUT6 MG	LUT4	LUT4 MG	LUT6	LUT6 MG
ac97_ctrl	4	4	3	4	4	4	6	4	4010	1739	2898	1523	4599	3733	13132	8424
aes_core	8	7	6	9	8	7	12	10	8959	4147	3912	2932	10275	8903	17727	16218
des_area	10	10	7	10	10	10	14	11	2153	1004	968	674	2469	2154	4386	3728
des_perf	6	6	4	7	6	6	8	7	32371	15652	6872	4863	37125	33601	31140	26900
ethernet	10	9	7	9	10	9	14	12	5047	2438	3953	2215	5788	5234	17913	12252
i2c	5	4	4	4	5	4	8	6	392	188	265	144	450	404	1201	794
mem_ctrl	11	9	8	9	11	9	16	12	3687	1731	2892	1413	4228	3715	13105	7813
pci_bridge32	9	8	7	8	9	8	14	11	6596	3156	5413	2911	7565	6774	24529	16102
pci_spoci_ctrl	6	5	5	5	6	5	10	7	372	173	251	137	427	370	1137	755
sasc	3	3	2	4	3	3	4	4	206	87	147	85	236	186	666	470
simple_spi	4	4	3	4	4	4	6	4	295	121	200	113	338	259	906	625
spi	10	9	7	9	10	9	14	11	1315	601	913	440	1508	1289	4137	2431
ss_pcm	3	3	2	3	3	3	4	3	124	61	102	48	142	131	462	266
systemcaes	11	11	8	13	11	11	16	16	2948	1470	1926	1158	3381	3155	8728	6405
systemcdes	9	8	5	9	9	8	10	10	1125	580	608	351	1290	1245	2755	1942
tv80	17	15	11	15	17	15	22	20	2963	1404	2132	1129	3398	3013	9661	6245
usb_funct	9	9	6	9	9	9	12	10	4906	2227	3408	1775	5626	4781	15443	9816
usb_phy	4	3	2	3	4	3	4	4	188	89	145	60	216	190	657	332
vga_lcd	7	7	6	7	7	7	12	9	40814	18145	28791	16561	46808	38954	130466	91604
wb_conmax	9	7	6	7	9	7	12	10	14580	6966	11003	5895	16721	14954	49860	32608
wb_dma	10	9	6	9	10	9	12	11	1469	711	1129	579	1685	1525	5116	3200
average	7.86	7.14	5.48	7.48	7.86	7.14	10.95	9.14	6406	2985	3711	2142	7346	6408	16816	11849

Table 1: Delay and area comparisons among different architectures

the logic depth by 36.5%. The average logic depth of mixing LUT-6 and macro-gate is larger than that for mixing LUT-4 macro-gates because our technology mapper sets the delay weight of a LUT-6 and a macro-gate as 1.98:1.00 based on Table 2. The experimental results show that the logic area is reduced by 5% due to the MBILP based area recovery algorithm (Subsection 3).

6. CONCLUSIONS AND FUTURE WORK

Targeting macro-gate 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 macro-gates in PLBs, a complete synthesis flow has been developed for such heterogeneous PLBs with mixed LUTs and macro-gates. The flow includes a cut-based delay-optimal technology mapping, a mixed binary integer and linear programming based area recovery algorithm to balance the resource utilization of macro-gates and LUTs for area-efficient packing, and a SAT-based packing. The proposed heterogeneous FPGA design has been evaluated using the newly developed flow. The experimental results show that mixing LUT and macro-gates, both with 6 inputs, improves performance by 16.5% and reduces logic area by 30% compared to using 6-input LUTs.

In the future, we will investigate more complicated macro-gate based heterogeneous PLBs architectures by the proposed synthesis flow, and consider area, delay and power optimization during the technology mapping and packing processes. We will also connect our synthesis flow with the back-end tools such as placement and routing for even more accurate architecture evaluation considering both logic and routing cost.

7. REFERENCES

- [1] V. Betz, J. Rose, and A. Marquardt, *Architecture and CAD for Deep-Submicron FPGAs*. Kluwer Academic Publishers, 1999.
- [2] "The programmable gate array data book," *Xilinx Inc.*, 1989.
- [3] J. Cong, et al, "Technology mapping and architecture evaluation for k/m-macrocell-based FPGAs," *TOADES*, 2005.
- [4] "Xilinx datasheets," <http://www.xilinx.com/literature>.
- [5] A. Cosoroaba and F. Rivoallon, "Achieving higher system performance with the virtex-5 family of FPGAs," in <http://www.xilinx.com/literature>.
- [6] F. Li and L. He, "Power modeling and characteristics of field programmable gate arrays," *TCAD*, 2005.

- [7] <http://iwls.org/iwls2005/benchmarks.html>.
- [8] <http://www.eecs.berkeley.edu/~alanmi/abc/>.
- [9] J. Cong and S. Xu, "Delay-optimal technology mapping for FPGAs with heterogeneous LUTs," *DAC*, 1998.
- [10] J. Cong, et al, "Cut ranking and pruning: Enabling a general and efficient fpga mapping solution," *FPGA*, 1999.
- [11] A. Kaviani and S. Brown, "Hybrid FPGA architecture," *FPGA*, 1996.
- [12] J. Cong and Y. Ding, "Flowmap: An optimal technology mapping algorithm for delay optimization in lookup-table based FPGA designs," *TCAD*, 1994.
- [13] J. Cong and Y. Ding, "On area/depth trade-off in LUT-based FPGA technology mapping," *TVLSI*, 1994.
- [14] D. Chen and J. Cong, "Daomap: A depth-optimal area optimization mapping algorithm for FPGA designs," *ICCAD*, 2004.
- [15] V. Manohara-rajah, et al, "Heuristics for area minimization in LUT-based FPGA technology mapping," *IWLS*, 2004.
- [16] A. Mishchenko, et al, "Integrating logic synthesis, technology mapping, and retiming," *IWLS*, 2005.
- [17] S. Chatterjee, et al, "Factor cuts," *ICCAD*, 2006.
- [18] A. Ling, et al, "Fpga logic synthesis using quantified boolean satisfiability," *SAT 2005*, pp. 444-450.
- [19] S. Safarpour, et al, "Efficient satbased boolean matching for fpga technology mapping," *DAC*, 2006.
- [20] D. Chai and A. Kuehlmann, "Building a better boolean matcher and symmetry detector," *DATE*, 2006.
- [21] S. C. Soheil Ghiasi, et al, "A unified theory of timing budget management," *DAC* 2004.
- [22] E. M. Sentovich, et al, "SIS: A system for sequential circuit synthesis," *Department of Electrical Engineering and Computer Science, Berkeley, CA 94720*, 1992.
- [23] J. M. Rabaey, et al, *Digital Integrated Circuits: A Design Perspective (Second Edition)*. Prentice-Hall, 2003.
- [24] <http://www.mosek.com>.
- [25] <http://www.cs.chalmers.se/Cs/Research/FormalMethods/MiniSat/MiniSat.html>.
- [26] Jianshe He, "Technology Mapping and Architecture of Heterogenous Field-Programmable Gate Arrays," M.A.Sc. Thesis, University of Toronto, 1993.
- [27] Kevin Chung, "Architecture and Synthesis of Field-Programmable Gate Arrays with Hard-wired Connections," Ph.D. Thesis, University of Toronto, 1994.