

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, University of California at Los Angeles,
 Los Angeles, CA 90095, USA¹
 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 7% and reduces logic area by 15% compared to using merely 6-input LUTs.

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 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 by 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], [11], mixing LUTs and PLAs [12] inside the PLB, PLBs with hardwired connections [13], or mapping logic to FPGAs with both

LUTs and embedded ROMs [14], [15], may improve logic density. In addition, commercial FPGAs [5] have benefited from small macro-gates (e.g., XOR2 and MUX2) inside the PLB. For instance, Altera has included a cascade-AND gate in their architectures [16], Xilinx had the OR-CY in Virtex-II [17], and both Altera and Xilinx map some wide fanin logic to carry chains. However it is unclear whether incorporating macro-gates with wide inputs inside PLB is beneficial. Figure 1 compares the logic depths achieved by the mix of LUT-4 and small macro-gates (XOR2 and MUX2) and the mix of LUT-4 and wider macro-gates (AND4 and MUX2-1). It indicates that the heterogeneous architecture with wider macro-gates has 6% on average (up to 42%) less logic depths compared to the one with small macro-gates. 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.

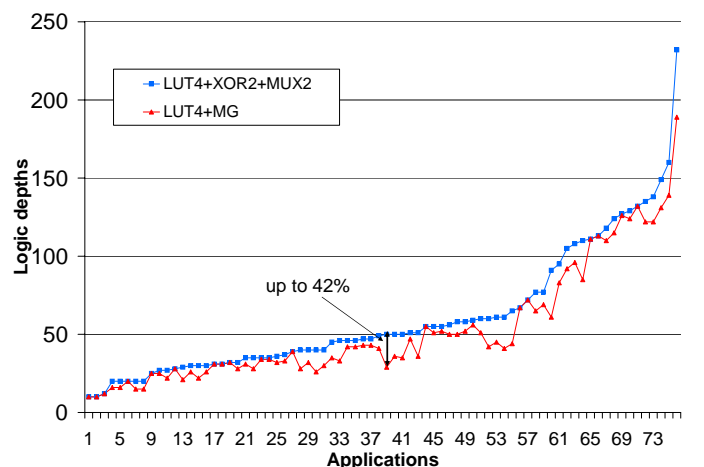


Fig. 1. Logic depths of heterogeneous architectures with small and wide macro-gates over 70 industrial applications

Effective and efficient synthesis tools are key enablers for the exploration of different architecture options. There are extensive studies (e.g., [18], [19], [20], [21], [22], [23]) 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. [24], [25] 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 7% and reduces logic area by 15% for IWLS'05 benchmarks [7] when compared to using 6-input LUTs.

The rest of this paper is organized as follows. Section II presents the methodology to design heterogeneous PLBs with mixed LUTs and macro-gates. Section IV describes the improved technology mapping and post-mapping area recovery algorithms to deal with the proposed FPGA architecture. Section V proposes a flexible SAT-based packing algorithm. The architecture evaluation results are given in Section VI, and the paper is concluded in Section VII. 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.

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 macro-gate design.

A. Preliminaries

To rank the logic functions extracted from the training FPGA application set, we need to identify two important relationships i.e., NPN-equivalence and inheritance equivalence, for every two logic functions.

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. [26]

For example, $F = ab + c$ and $G = a'c + b$ are NPN-equivalent. Although there are $2^{2^4} = 65536$ different 4-variable functions, only 222 of them are NPN independent (i.e., NPN-inequivalent to each other). In the rest of this paper, we use the term ‘‘NPN-class’’ to denote a set containing only NPN-independent elements. In this work, we limit the programmability of a macro-gate with input/output negation, which provides a good trade-off between flexibility and cost (performance and area). For an N -input macro-gate with input/output negation, $N + 1$ memory cells (i.e., configuration

bits) are needed rather than 2^N memory cells for an LUT with N inputs (LUT- N).

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 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.

In the other words, A and B are NPN-equivalent if we fix one of the inputs of A as logic zero or one, i.e., B is a *cofactor* of A . For example, $f_1(a, b, c, d) = abcd$ is inheritance equivalent to $f_2(a, b, c) = abc$ by fixing input d as one. In fact, f_1 is a 4-input AND gate, which can be used to implement a 3-input AND gate (i.e. f_2). The inheritance equivalence of the full set of NPN-class with up to N inputs can be represented by the following NPN-class diagram (NCD).

B. NPN-Class Diagram (NCD)

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: NPN-Class Diagram (NCD) is a directed acyclic graph (DAG), where each node in level N is the representative¹ 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. Figure 2 shows the NCD for all 3-input functions.

C. Utilization NPN-Class Diagram (UND)

To extract N -input macro-gate 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 NPN-Class Diagram* (UND), which is a sub-graph of NCD in addition to the weights associated with each node. UND can be defined recursively as follows.

Definition 4: (Utilization NPN-Class Diagram (UND)) For a particular application D , utilization NPN-Class diagram \mathcal{U}_D of D is a sub-graph 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 \mathcal{U}_D . If a NCD node is present in \mathcal{U}_D , all of its fanout nodes and edges should be added into \mathcal{U}_D recursively. Each node is associated with a 3-tuple $\Phi(f, n, c)$, where f is the functionality description of this NPN-Class, n and c are the implemented frequency and implementation capability (will be defined later), respectively.

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

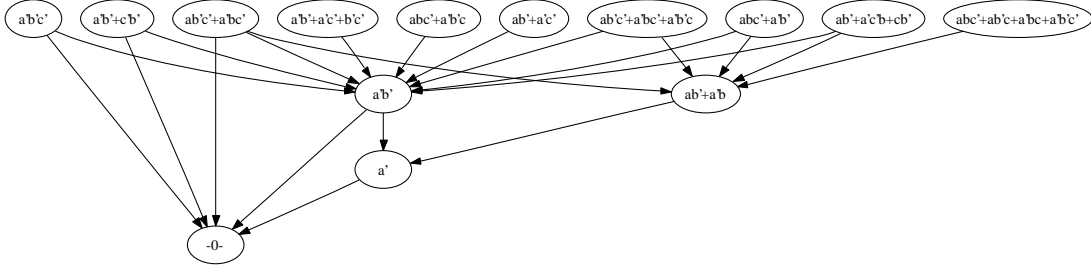


Fig. 2. NCD for 3-input logic functions

Definition 5: (Implemented Frequency (IF)) For a 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: (Implementation Capability (IC)) For a 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:

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

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

Figure 3 shows the UND for a small application that is mapped by LUT-3. There are totally 12040 functions implemented by LUTs in this application, and only three 3-input NPN-classes are presented. An interesting observation from Figure 3 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 12040 total functions). In fact, it has a child node $ab' + a'b$ whose IF is 4410 (36% of 12040 total functions), if we look into the fanout cone (shaded area) of node $ab'c + a'bc' + a'b'c$.

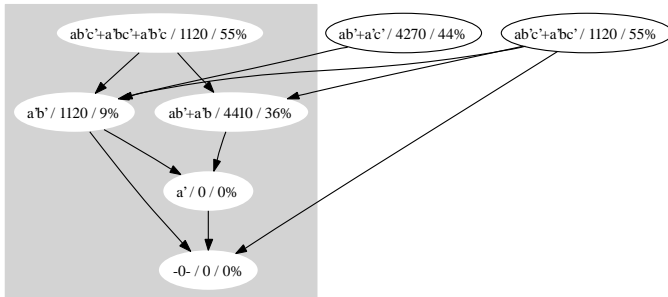


Fig. 3. An example UND

As we are interested in the minimal number of logic gates to cover all logic functions in an application, i.e., minimal functional 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.

D. Logic Function Ranking Metrics

Our objectives for ranking logic functions are (i) using fewer logic functions to cover larger subset of all functions presented in the mapped application, and (ii) keeping high utilization of input pins. In fact, there is a trade-off between those two objectives. Figure 4 shows UNDs for four toy applications. Each node is labeled by its functionality and IF. In case (a), function $ab' + a'c'$ appears once and ab appears 10 times. $ab' + a'c'$ should have a higher rank than ab if the objective is to minimize the number of NPN-classes that can cover all functions. On the other hand, if we are more interested in pin utilization, ab is a better choice, as 10 gates with function ab' only use 2/3 pins if $ab' + a'c'$ is used to implement all functions in case (a).

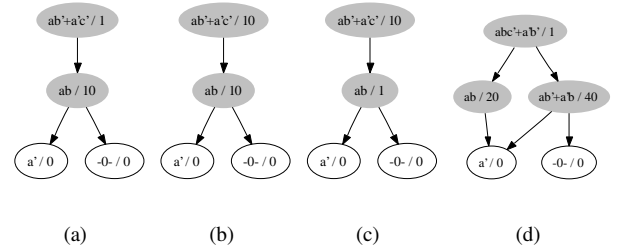


Fig. 4. Illustration for the trade off among various metrics

Intuitively, the two values, implemented frequency (IF) and implementation capability (IC), associated with each node in UND are good indicator for the ranking of a NPN-Class function, however they may not provide the best tradeoff between the above two objectives. In this work, we propose a new metric, Utilization Weighted Implementation Capability (UWIC), which is a deterministic analytical metric of NPN-Class functions and leads to a good trade-off between logic function covering and pin utilization.

Definition 7: Utilization Weighted Implementation Capability (UWIC) of NPN-Class function f is

$$UWIC_f = \sum_{\forall v \in \text{fanout cone of } f} \frac{\text{num_input}(v)}{\text{num_input}(f)} \cdot IF_v \quad (2)$$

We calculate the UWIC values for each case in Figure 4 as follows.

- 1) $UWIC_{ab} = 10 \times 1.0 = 10$, $UWIC_{ab'+a'c'} = 10 \times 2/3 + 1 \times 1.0 = 7.7$, so $Rank_{ab} > Rank_{ab'+a'c'}$.
- 2) $UWIC_{ab} = 10 \times 1.0 = 10$, $UWIC_{ab'+a'c'} = 10 \times 2/3 + 10 \times 1.0 = 16.7$, so $Rank_{ab'+a'c'} > Rank_{ab}$.

- 3) $UWIC_{ab} = 1 \times 1.0 = 1$, $UWIC_{ab'+a'c'} = 1 \times 2/3 + 10 \times 1.0 = 10.3$, so $Rank_{ab'+a'c'} > Rank_{ab}$.
- 4) $UWIC_{ab} = 20 \times 1.0 = 20$, $UWIC_{ab'+a'b} = 40 \times 1.0 = 40$, $UWIC_{abc'+a'b'} = (20 + 40) \times 2/3 + 1 \times 1.0 = 41$, so $Rank_{abc'+a'b'} > Rank_{ab'+a'b} > Rank_{ab}$.

It shows that the UWIC values are consistent with the good trade-off between logic function covering and pin utilization that we just observed.

E. UND for Wide Input Functions

To explore less-than-5-input functions, 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 by 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 on-line checking NPN-equivalence for over-5-input functions, which can be performed efficiently by the method proposed in [26]. 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.

F. Macro-Gate Design

The logic function extraction and ranking framework proposed above is implemented by the mix of Perl and C on 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.

$$\begin{aligned}
 g_1(a, b, c, d, e, f) &= abcdef \\
 g_2(a, b, c, d, e, f) &= ab'c' + bcf + bc'd + b'ce \quad (3) \\
 g_3(a, b, c, d, e, f) &= ab'cd'e + bce.f + def \\
 g_4(a, b, c, d, e, f) &= ab' + a'cd' + b'c' + e' + f'
 \end{aligned}$$

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 5. 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.

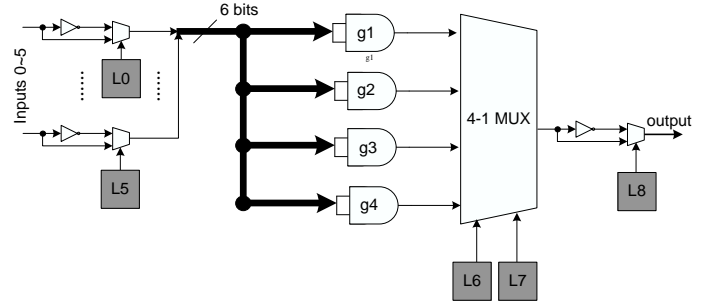


Fig. 5. Architecture of the macro-gate

III. OVERALL SYNTHESIS FLOW

For the proposed heterogeneous FPGA with mixed macro-gate and LUTs, we present the following synthesis flow as shown in Figure 6. Given a gate level netlist of an application described by an And-Inverter Graph (AIG), we first perform technology mapping to map the application into LUTs and macro-gates. The technology mapping phase includes multiple steps to minimize the delay and balance the resource utilization for LUTs and macro-gates. They will be detailed in Section IV. After that, we pack the mapped application into logic blocks. The packer is based on satisfiability and is highly flexible, as described in Section V. Finally, we perform the placement and routing. Note that we assume the 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.

IV. 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. A simple way to handle the heterogeneous architecture with both K -input LUTs and M -input macro-gates is the following 3-step algorithm. First, utilize the existing technology mapping algorithm which handles LUT based FPGAs to map the application by M -input LUTs, then replace those LUTs that can be implemented by M -input macro-gates, and finally remap the remaining sub-circuit by K -input LUTs. Although both mapping procedures in this 3-step algorithm are delay optimal, the final mapping result could be far from optimal since both mappers have no global views.

A. Functional Cut Pruning

[24] presented a general algorithm for heterogeneous FPGA technology mapping, but it is not scalable to large macro-gates. [23] extended the traditional cut based technology mapping algorithm [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 [23], the truth table of the NPN-classes that

²The macro-gate in [23] was built by three LUTs (without the logic gates) and the homogeneous PLBs were assumed.

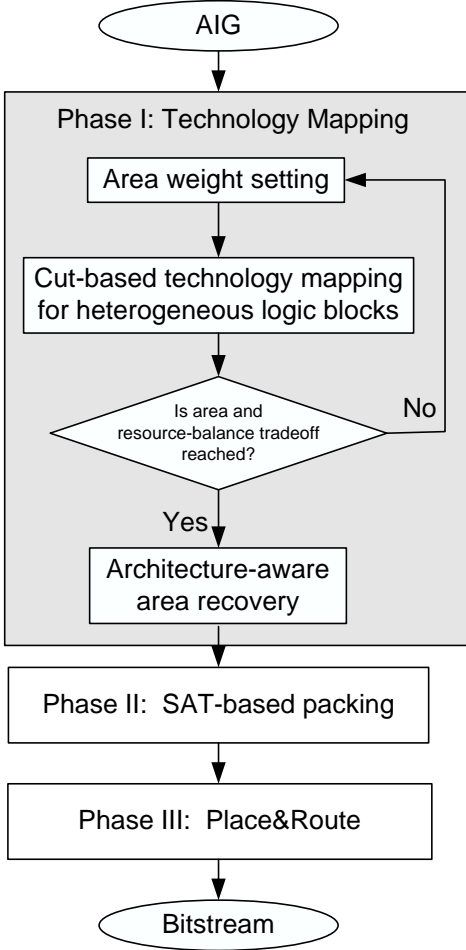


Fig. 6. Overall synthesis flow

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 [23] by labeling cuts as macro-gates or LUTs based on their logic functions. However, if $M \gg K$, i.e., the inputs of macro-gates are much more than the inputs of LUTs in the target architecture, cut enumeration is still expensive even with factor cuts [23]. In our experiments, we observe that certain P -cuts ($M > P \geq K$) can be pruned by matching their functions with the cofactors of macro-gate functions in the forward traversal process and preventing their further propagation. We pre-calculate and store all $M - 1$ to K cofactors for each function of the macro-gate. Empirically, if a P -cut cannot match any of these cofactors, its extension³ and itself will unlikely be implemented by the macro-gate.

³We use the way proposed in [10] to generate all the cuts by extending cuts in topology order.

Figure 7 shows a fragment of the And-Invert graph (AIG) [23], where the dot-line edge indicates an inverter. The shadow shows a fanout free cut [10] $Cut(y)_1 = (b, c, d)$ for node y . The function of cut $Cut(y)_1$ is $b(d' + c')$. We can extend cut $Cut(y)_1$ with node z to obtain a new 4-feasible cut (b, c, d, z) for node x whose function is $Cut(x) = b(d' + c')z$. Suppose the target FPGA architecture contains mixed LUT-2 and 4-input macro-gates, and function $Cut(y)_1$ is not a cofactor of any macro-gate functions, which indicates that $Cut(x) = Cut(y)_1 \wedge z$ cannot be implemented by macro-gates if f_z is not a constant 0. In this case, we can prune cut $Cut(y)_1$ as it cannot be implemented by either a macro-gate (due to the logic function) or an LUT-2 (due to the input number). On the other hand, if z provides a controlling value for x , i.e., $f_z \equiv 0$, cut $Cut(y)_1$ can still be pruned since the function of $Cut(y)_1$ becomes an Observability Don't-Cares (ODC) term. Based on the above observation, the following empirical rule is proposed for cut pruning. Note that the other cuts, $Cut(y)_2$ and $Cut(y)_3$, rooted at y must be kept to propagate due to the existence of LUT-2.

Rule 1: (Functional Cut Pruning) For a given node v , if the function of a fanout free cut [10] C_v rooted at node v is not a cofactor of any macro-gate functions, cut C_v is pruned.

Note that this rule is extremely helpful when $M \gg K$ since a large portion of more-than- K -input cuts will be pruned. Also the functional pruning rule is orthogonal to the factor cut pruning [23] and it can be applied along with factor cut pruning to further reduce runtime.

B. Architecture-Aware Area Recovery

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

$$\min_{\forall \alpha} \Phi(N, \alpha) = \frac{N}{C_L} \cdot \frac{1}{1 + \beta} \quad (4)$$

$$\alpha^* = \frac{\beta}{1 + \beta} \quad (5)$$

For a given architecture, (4) provides a lower bound of the number of PLBs for a mapped result, where parameters N and α decide the tightness of the packing. (5) show that the tightest packing can be achieved when the LUT-MG ratios⁴ 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

⁴LUT-MG ratio is the number of LUTs divided by the number of macro-gates in an application.

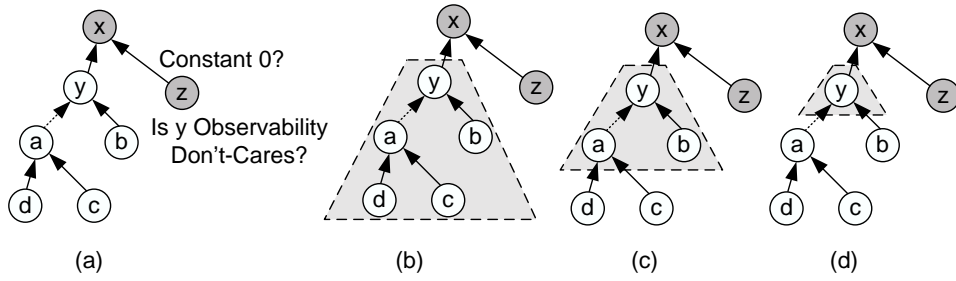


Fig. 7. Illustration of functional cut pruning. (a) All cuts rooted at y can be pruned if y is an observability don't care of x . (b) $Cut(y)_1 = b(d' + c')$. (c) $Cut(y)_2 = a'b$. (d) $Cut(y)_3 = y$

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 8 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, 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 a small N and an α close to the target LUT-MG ratio. From Figure 8, we see that the total number of LUTs and macro-gates (N in (4)) 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 resulting LUT-MG ratio, β , by simply adjusting area weight.

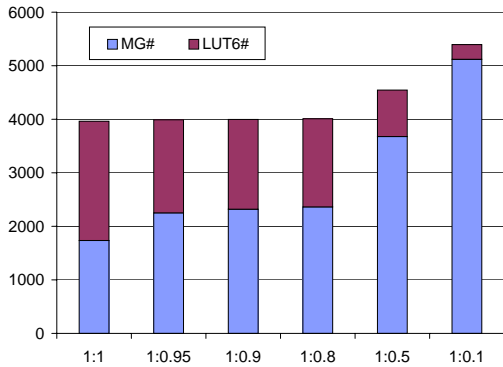


Fig. 8. Impact of area weight on LUT-MG ratio

1) *MBILP Formulation*: 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 [27], 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 an LUT or a macro-gate. Without loss of generality, we assume that

the intrinsic delay of an LUT is ΔD larger than a macro-gate. In this case, the binary search should be able to find an 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 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 re-mapped as an 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.

$$\min \quad \left| \sum_{\forall v_j \in V_m(G)} m_j - \frac{1}{1+\beta} \cdot N \right| \quad (6)$$

$$\text{s.t.} \quad m_j \leq \frac{b_j}{\Delta D}, \forall v_j \in V_m(G) \quad (7)$$

$$m_j \in \{0, 1\}, \forall v_j \in V_m(G) \quad (8)$$

$$a_i + d_j + b_j \leq a_j, \forall e(i, j) \in E(G) \quad (9)$$

$$b_i \geq 0, \forall v_i \in V(G) \quad (10)$$

$$a_i \leq T, \forall v_i \in V(G) \quad (11)$$

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.

$$\min \quad t \quad (12)$$

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

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

$$t \geq 0 \quad (13)$$

2) *Example*: Figure 9 shows a circuit which is pre-mapped by LUTs and macro-gates with the same inputs. The delay of an LUT and a macro-gate is 5 units and 4 units, respectively. The rectangular nodes represent LUTs and the elliptical nodes represent macro-gates. The numbers in each node denote the arrival time and required time of the node given that the required time in PO is equal to the clock period.

Based on the MBILP formulation presented in Subsection IV-B.1, the MBILP form for this area recovery problem can be expressed as follows. Shown in (9), arrival time constraints

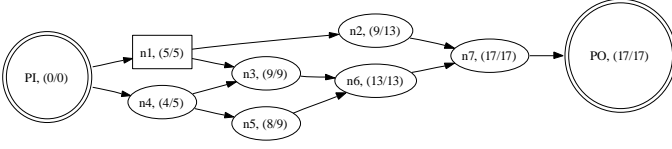


Fig. 9. The pre-mapped circuit

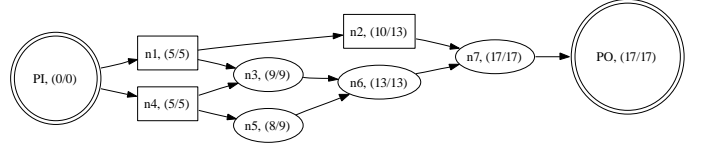


Fig. 10. The re-mapped circuit after area recovery

are listed according to all edges.

$$\begin{aligned}
 a_{PI} + 5 + b_1 &\leq a_1, & a_{PI} + 4 + b_4 &\leq a_4, & a_1 + 4 + b_2 &\leq a_2, \\
 a_1 + 4 + b_3 &\leq a_3, & a_4 + 4 + b_3 &\leq a_3, & a_4 + 4 + b_5 &\leq a_5, \\
 a_2 + 4 + b_7 &\leq a_7, & a_3 + 4 + b_6 &\leq a_6, & a_5 + 4 + b_6 &\leq a_6, \\
 a_6 + 4 + b_7 &\leq a_7, & a_7 &\leq a_{PO}
 \end{aligned} \quad (14)$$

Based on (10), the following constraints guarantee that the timing slack budget is nonnegative.

$$b_i \geq 0, \quad i = \{1, \dots, 7\} \quad (15)$$

Based on (11), the critical path delay is preserved by the following constraints.

$$a_{PO} \leq T, \quad a_{PI} = 0 \quad (16)$$

Based on (7), the following constraints leverage timing slack with resource remapping.

$$(5 - 4) \cdot m_i \leq b_i, \quad m_i = \{0, 1\}, \quad i = \{2, \dots, 7\} \quad (17)$$

Since there are equal numbers of LUTs and macro-gates in the target PLB, the ideal number of macro-gates is $N/2 = 7/2$. Therefore, the objective is to minimize the gap between the number of macro-gates ($m_2 + \dots + m_7$) and $7/2$. We write it as a linear form based on (13) as follows.

$$\begin{aligned}
 \min & \quad t \\
 \text{s.t.} & \quad m_2 + \dots + m_7 - 7/2 \leq t, \\
 & \quad m_2 + \dots + m_7 - 7/2 \geq t
 \end{aligned} \quad (18)$$

Solving the MBILP problem with objective (18) and constraints (14), (15), (16) and (17), the solutions are as follows.

$$\begin{aligned}
 a_1 &= 5, a_2 = 10, a_3 = 9, a_4 = 5, a_5 = 9, \\
 a_6 &= 13, a_7 = 17, a_{PO} = 17, a_{PI} = 0, \\
 b_3 &= b_5 = b_6 = b_7 = 0, b_2 = b_4 = 1, \\
 m_3 &= m_5 = m_6 = m_7 = 0, m_2 = m_4 = 1
 \end{aligned}$$

This solution corresponds to a set of resource reassignment by remapping node 2 and node 4 to LUT-6 and the resulting application has 3 LUT-6 and 4 macro-gates, which achieves the best balance and can be packed into 4 PLBs. Compared to the original application, logic area is reduced by 30% (from 6 PLBs to 4 PLBs). The re-mapped circuit is shown in Figure 10.

3) *Generalization*: The above algorithm can be generalized to handle the case that the target architecture has more than one type of macro-gate. Without loss of generality, suppose there are two types of macro-gates in the target architecture, e.g., G_1 and G_2 . The number of LUTs, G_1 s and G_2 s within a PLB is C_L , C_{g1} and C_{g2} , respectively. The intrinsic delay of LUTs,

G_1 s and G_2 s within a PLB is D_{lut} , D_{g1} and D_{g2} , respectively. As shown in Figure 6, the initial mapping is decided by the cut-based technology mapping by properly setting the area weights for LUTs, G_1 s and G_2 s. The objective is to remap the logic elements so that the number of LUTs, G_1 s and G_2 s in the resulting application is as close to $C_L : C_{g1} : C_{g2}$ as possible.

For each node (logic element) n_i , if n_i can be implemented by an LUT (i.e., fan-in number of this node is smaller or equal to LUT inputs), we associate a binary variable u_i with it, where $u_i = 1$ means that this node is remapped to an LUT, otherwise it is remapped by G_1 or G_2 . In the same manner, we associate binary variables g_{1i} and g_{2i} to indicate if node n_i is mapped to G_1 or G_2 . The following constraints guarantee that only one mapping is accepted.

$$u_i + g_{1i} + g_{2i} = 1 \quad (19)$$

To leverage the timing slack with the resource assignment solution, we have the following constraints according to the current mapping of node i . For concise presentation, we use $n_i \models G$ to denote that node i can be implemented by device G (could be an LUT or macro-gate).

Case 1 Node i has been mapped by LUT at present.

$$(D_{g1} - D_{lut}) \cdot g_{1i} \leq b_i, \quad \text{if } n_i \models G_1 \quad (20)$$

$$(D_{g2} - D_{lut}) \cdot g_{2i} \leq b_i, \quad \text{if } n_i \models G_2 \quad (21)$$

Case 2 If node i has been mapped by G_1 at present.

$$(D_{lut} - D_{g1}) \cdot u_i \leq b_i, \quad \text{if } n_i \models \text{LUT} \quad (22)$$

$$(D_{g2} - D_{g1}) \cdot g_{2i} \leq b_i, \quad \text{if } n_i \models G_2 \quad (23)$$

Case 3 If node i is mapped by G_2 at present, we have the following constraints.

$$(D_{g1} - D_{g2}) \cdot g_{1i} \leq b_i, \quad \text{if } n_i \models G_1 \quad (24)$$

$$(D_{lut} - D_{g2}) \cdot u_i \leq b_i, \quad \text{if } n_i \models \text{LUT} \quad (25)$$

All other timing constraints are the same as the above example.

The objective function is represented as follows.

$$\begin{aligned}
 \min & \quad W_u \cdot \left| \sum u_i - \frac{N \cdot C_L}{C_L + C_{g1} + C_{g2}} \right| + \\
 & \quad W_1 \cdot \left| \sum g_{1i} - \frac{N \cdot C_{g1}}{C_L + C_{g1} + C_{g2}} \right| + \\
 & \quad W_2 \cdot \left| \sum g_{2i} - \frac{N \cdot C_{g2}}{C_L + C_{g1} + C_{g2}} \right|
 \end{aligned} \quad (26)$$

which can be linearized as follows.

$$\min \quad t_u + t_1 + t_2 \quad (27)$$

$$\text{s.t.} \quad \frac{t_u}{W_u} \leq \sum u_i - \frac{N \cdot C_L}{C_L + C_{g1} + C_{g2}} \leq \frac{t_u}{W_u} \quad (28)$$

$$\frac{t_1}{W_1} \leq \sum g_{1i} - \frac{N \cdot C_{g1}}{C_L + C_{g1} + C_{g2}} \leq \frac{t_1}{W_1} \quad (29)$$

$$\frac{t_2}{W_2} \leq \sum g_{2i} - \frac{N \cdot C_{g2}}{C_L + C_{g1} + C_{g2}} \leq \frac{t_2}{W_2} \quad (30)$$

$$t_u \geq 0, t_1 \geq 0, t_2 \geq 0 \quad (31)$$

where W_1 , W_2 and W_u are the weight to indicate which resource is the most crucial one that needs to be balanced.

C. Discussions

The effectiveness of the proposed technology mapping flow, as shown by Phase I in Figure 6, can be revealed by comparing to a brute-force 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 Figure 8, which is 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. An incautious setting of the area weight may cause low utilization rate (e.g., $\beta=1:1$ in Figure 8) or significant area increase (e.g., $\beta=1:01$ in Figure 8). Thereby, we need to carefully select a set of area weight assignments, e.g., 1:0.95, 1:0.9 and 1:0.8 in Figure 8, which give good trade-offs between N and α under the target architecture specification.

Moreover, the architecture-aware area recovery sub-phase 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 macro-gate in a PLB. The number of PLBs is reduced by 5% due to the area recovery. Note that 10 applications achieve perfect resource utilization (resource demand is 1:1 for LUTs and macro-gates) after the area recovery as shown in Figure 11.

To show the effectiveness of function pruning (presented in Subsection IV-A), we map the benchmark applications to an architecture with LUT4 and macro-gates. Table I shows that the functional pruning prunes 31.59% more cuts and reduces 7.76% overall runtime while both area and delay qualities of the mapped designs are well preserved, compared to the algorithm without using FP. More improvement is expected when larger macro-gates are used.

V. 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. This problem essentially solves a clustering problem since we want to group circuit elements that are connected together in the same PLB so that wire lengths are reduced and routing congestion is avoided. Typically, a packer (e.g. *T-VPack* [1]) uses heuristics to determine what circuit elements should be clustered together. Then each of the clusters is checked to see if it can really fit into a FPGA PLB according to certain constraints. In *T-VPack*, the clustering constraints include the number of PLBs, in/out pins and clocks of each cluster. However the PLB architecture of recent commercial FPGAs is more complicated. In general, the clustering constraints that should be met for each cluster are as follows: (i) each circuit element in the cluster can be placed in certain location in the PLB, (ii) all interconnections within the cluster can be implemented in the PLB itself, and (iii) each outgoing/incoming signal is assigned to an appropriate PLB output/input pin.

However, the existing packing tools such as *T-VPack* hard-codes the architecture specification of a PLB and this check has to be written from scratch if the PLB architecture changes. This is time consuming and needs to be debugged carefully to avoid bugs.

Different from [24] and [25], we solve the problem of validating PLB packing as a local place and route problem at the PLB level. It is a placement problem since multiple compatible sites may be present for each logic element in the cluster⁵. 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 then 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, then the packing is invalid.

Now we briefly sketch the problem formulation with an example. Assume that we are attempting to fit the fragment described in Figure 12 (b) into the simplified Xilinx Spartan3-like PLB in Figure 12 (a).

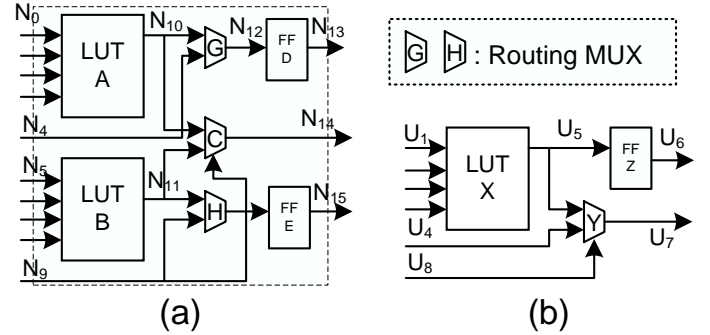


Fig. 12. (a) Architecture of a FPGA PLB, (b) Sub-circuit to be packed

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.

$$(\neg X@A) \vee (\neg X@B) \quad (32)$$

$$X@A \vee X@B \quad (33)$$

Constraint (32) implies that if X is placed at A then it cannot be placed at B and vice versa. Constraint (33) means that X has to be placed at at least one of the sites A and B . The two equations above guarantee that X would 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 on nets, as well, so

⁵For instance in Xilinx Spartan3 [4] PLB there are two LUTs.

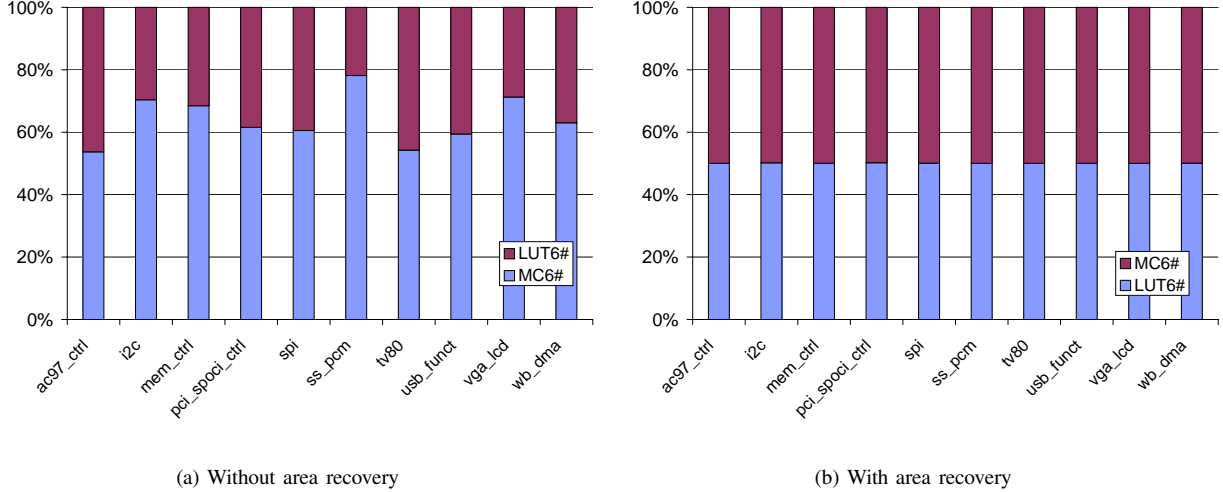


Fig. 11. Ratio of resource usage before/after area recovery

	cut#		cut#/node		runtime(s)		area		delay	
	non	pru	non	pru	non	pru	non	pru	non	pru
ac97_ctrl	145284	101330	10.7	7.4	1.57	1.34	1739	1739	4	4
aes_core	629759	379060	27.5	16.5	3.65	3.01	4143	4147	7	7
des_area	411217	189518	80	36.8	2.79	1.93	1004	1003	10	10
des_perf	5992341	3316862	64.5	35.7	30.11	20.15	15652	15653	6	6
ethernet	279832	181849	17.5	11.4	2.04	1.76	2438	2438	9	9
i2c	16187	11215	14.4	10	0.79	0.75	188	188	4	4
mem_ctrl	294274	199964	28.3	19.2	2.17	1.87	1731	1731	9	9
pci_bridge32	473859	315989	21	14	2.96	2.51	3156	3156	8	8
pci_spci_ctrl	25150	18169	26	18.8	0.84	0.81	173	173	5	5
sasc	7888	4988	9.7	6.1	0.75	0.73	87	87	3	3
simple_spi	11264	7515	10.9	7.3	0.71	0.73	121	121	4	4
spi	139042	85902	37.4	23.1	1.41	1.22	601	601	9	9
ss_pcm	3394	2275	6.7	4.5	0.75	0.63	61	61	3	3
systemcaes	582084	263954	53.1	24.1	3.85	2.35	1470	1470	11	11
systemcdes	230894	118130	63.9	32.7	2.01	1.46	580	580	8	8
tv80	351185	212358	41.5	25.1	2.49	2	1404	1404	15	15
usb_funct	358214	211159	21.4	12.6	2.41	1.99	2227	2227	9	9
usb_phy	3840	2757	7.5	5.4	0.74	0.72	89	89	3	3
vga_lcd	2994175	2333649	23.3	18.1	14.67	13.55	18145	18145	7	7
wb_conmax	1053906	793439	25.3	19.1	5.59	4.97	6966	6966	7	7
wb_dma	91294	62457	20	13.7	1.16	1.07	711	711	9	9
ave	387521.5	240195.0	10.7	12.0	2.6	2.2	2985.24	2985.43	7.14	7.14
		-31.59%		-31.50%		-7.76%				

TABLE I
EFFECTIVENESS OF FUNCTIONAL PRUNING

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

$$X@A \rightarrow U_5@N_{10}, \quad (34)$$

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 shall also be 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* (35) so that no output is driven by two inputs. We also need to add *routing constraints* (36) so that

if a net is present at an input to a routing multiplexer and the appropriate control bit is active then the output has the correct net.

$$G_{0 \rightarrow \text{out}} \rightarrow (\neg G_{1 \rightarrow \text{out}}) \quad (35)$$

$$(G_{0 \rightarrow \text{out}} \wedge U_5@N_{10}) \rightarrow U_5@N_{12} \quad (36)$$

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

$$(RT_B \wedge B_{0 \rightarrow \text{out}}) \rightarrow (\neg B_{1 \rightarrow \text{out}}) \quad (37)$$

$$(RT_B \wedge B_{0 \rightarrow \text{out}} \wedge U_8 @ N_5) \rightarrow U_8 @ N_{11} \quad (38)$$

where RT_B encodes the choice as to whether LUT B is used as a route-through. Constraint (37) 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 (38) 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 (37) 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 it helps with 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 added. For example, if LUT X is placed in site A , X 's inputs $U_1 \cdots U_4$ can be mapped to A 's inputs $N_0 \cdots N_3$ under arbitrary permutation, which can be expressed as follows.

$$\begin{aligned} X @ A \rightarrow & U_1 @ N_0 \wedge U_2 @ N_1 \wedge U_3 @ N_2 \wedge U_4 @ N_3 \vee \\ & U_1 @ N_1 \wedge U_2 @ N_0 \wedge U_3 @ N_2 \wedge U_4 @ N_3 \vee \cdots \\ & U_1 @ N_3 \wedge U_2 @ N_2 \wedge U_3 @ N_1 \wedge U_4 @ N_0 \end{aligned} \quad (39)$$

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

$$U_6 @ N_{13} \vee U_6 @ N_{14} \vee U_6 @ N_{15} \quad (40)$$

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, the satisfying assignment could 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.

The principal advantage of this method is that we can validate PLB packing for different PLB architectures without any architecture specific code. This method can be especially useful for new architectures before specialized code has been written to validate packing and also to check correctness of the specialized code during regression testing. The method is also applicable for architecture exploration as it is infeasible to write specialized valid packing checkers for all candidate architectures.

VI. ARCHITECTURE EVALUATION

A. Area and Delay Modeling

We use Cadence digital design platform *Encounter* [28] to synthesize the logic gates $g_1 \cdots g_4$ (equations 3) using Cadence 90nm GPDK technology [29]. The schematic for the synthesized macro-gates is shown in Figure 13, where the sizes of the cells are determined automatically by Encounter.

The area of the standard cells used to implement the macro-gates and LUTs are listed in Table II, based on which we can estimate the area of the macro-gate and LUTs. A macro-gate consists of the following elements: logic gates $g_1 \cdots g_4$ ($67.37 \mu\text{m}^2$), a 4:1 MUX ($18.92 \mu\text{m}^2$), 7 programmable inverters ($15.89 \mu\text{m}^2$), 9 1-bit SRAMs for programmable bits ($40.86 \mu\text{m}^2$), 7 2:1 MUXes for programmable inverters ($47.67 \mu\text{m}^2$). Therefore the overall area of a macro-gate is $190.71 \mu\text{m}^2$. Suggested in [3], an LUT-4 can be implemented by 16 1-bit SRAMs, 4 inverters and 15 2:1MUXes which total $170.64 \mu\text{m}^2$, and an LUT-6 can be implemented by 64 1-bit SRAM, 6 inverters and 63 2:1MUXes, which total $699.56 \mu\text{m}^2$.⁶

The delay of an LUT/macro-gate includes its intrinsic delay and the delay of its input selection MUX. For LUT-4 and LUT-6, we properly size the cells and use SPICE to simulate the intrinsic delay and the input selection MUX delay based on the setting in [30]. We use the same input selection MUX delay for both LUT-6 and the macro-gate. The area and delay models used in our experiments are summarized in Table III.

TABLE II

LIST OF AREA (IN μm^2) FOR STANDARD CELLS IN THE SYNTHESIZED LOGIC GATES (FROM CADENCE 90NM GPDK TECHNOLOGY)

cell	area	cell	area
INVXL	2.27	NOR2BXL	4.54
NAND2XL	3.03	OAI32XL	6.81
NAND4XL	5.30	AND2XL	4.54
MX2X1	6.81	NOR4BXL	6.81
OR3XL	6.06	OAI21XL	4.54
OAI222XL	8.33	MX4X1	18.92
1-bit SRAM	4.54		

TABLE III

SUMMARY OF THE AREA AND DELAY MODELING FOR LOGIC ELEMENTS BASED ON CADENCE 90NM GPDK TECHNOLOGY

	macro-gate	LUT-4	LUT-6
area (in μm^2)	190.71	170.64	699.56
delay (in ps)	431.82	471.66	646.82

⁶Note that the above area modeling assumes a conservatively-designed SRAM bit cell, which makes the LUT-4 size more optimistic compared to LUT-6. The reason is as follows. SRAM bits are assumed to be built into large arrays, where the reported LUT sizes assume shared wires on all four sides. This gives an optimistic size in the data sheet. For a 16-bit LUT-sized array, the actual area can be much larger than $16 * \text{SRAM}$ size due to the unshared area on the perimeter. However, in our conclusion, we intend to compare the architecture with LUT-6 to the one with the mix of LUT-6 and macro-gates, and compare the architecture with LUT-4 to the one with the mix of LUT-4 and macro-gates, respectively (will be mentioned in Section VI-B). Therefore, the area advantage of LUT-4 in our modeling will not affect the conclusions.

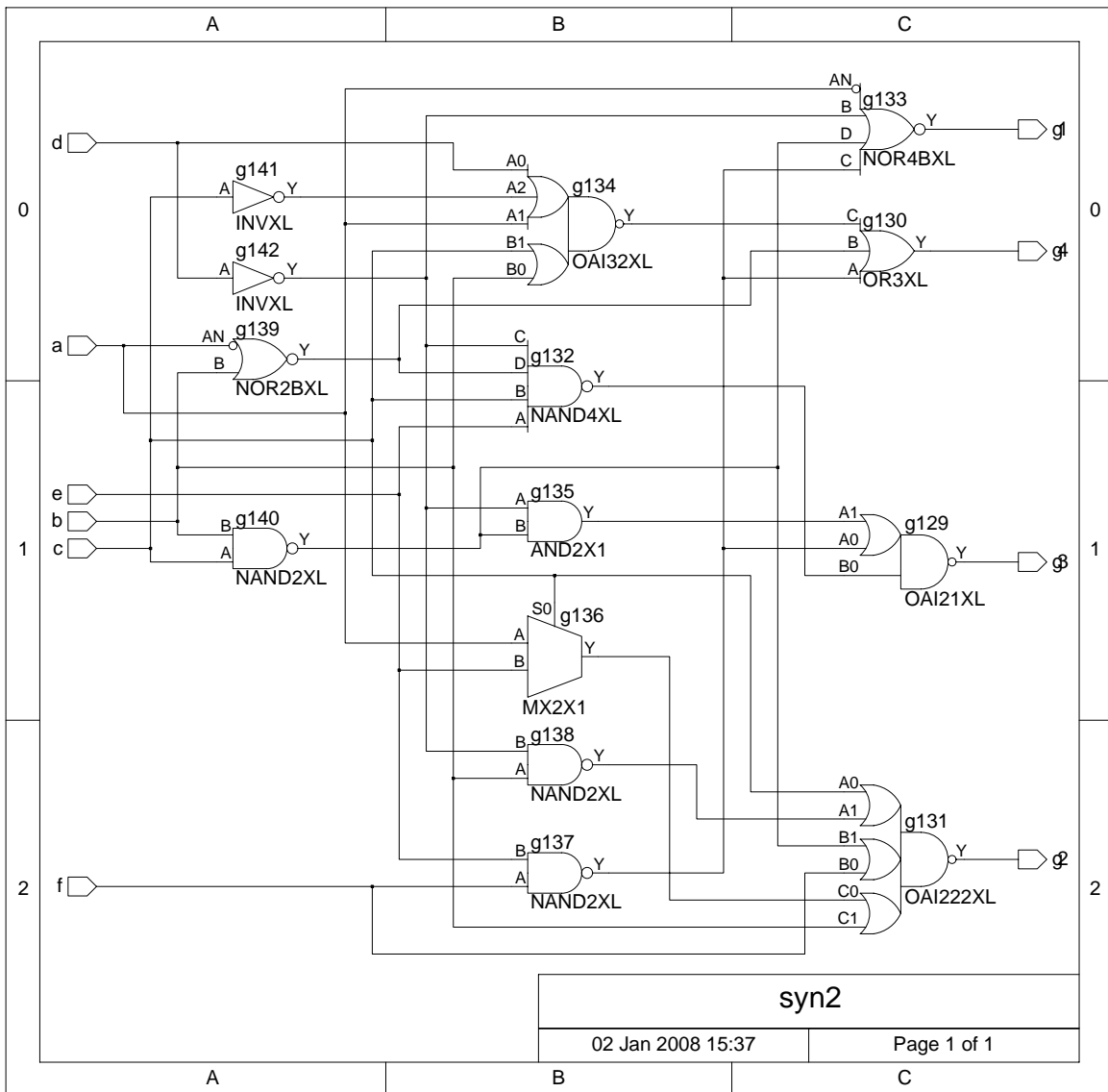


Fig. 13. Synthesized logic gates $g_1 \dots g_4$ with by Cadence Encounter under Cadence 90nm GPDK Technology

B. Experimental Results

We implement the improved mapping algorithms in 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 [31] and miniSAT [32], respectively.

We compare the following four architectures, (1) a PLB containing two LUT-4s, (2) a PLB containing one LUT-4 and one macro-gate, (3) a PLB containing two LUT-6s, and (4) a PLB containing one LUT-6 and one macro-gate. All macro-gates share the same structure as shown in Figure 5 and Figure 13. For architecture (1) and (2), we assume 8 inputs for each PLB, and for architecture (3) and (4), we assume 10 inputs for each PLB. The cluster architectures are assumed to be the same as the one shown in Figure 3.1 in [1]. We perform the CAD flow shown in Figure 6. The logic delay is estimated based on the model shown in Table III. For the two homogeneous architectures (i.e., architecture (1)

and (3)), we extract the routing delay by performing VPR [1]. Due to the absence of the physical design tools for the heterogeneous architecture with mixed macro-gates and LUTs (architecture (2) and (4)), we use the routing delay of (2) and (4) to approximate that of (1) and (3), respectively. As shown in Table V, architecture (2) and (1) share the similar logic area, but (2) has 7% fewer PLBs than (1). In addition, the logic area of architecture (4) is about 30% smaller than that of (3). Therefore such an approximate for routing delay is not over optimistic as the logic area reduction achieved by the proposed heterogeneous architectures can compensate with the limitations of the heterogeneous architectures (will be stated in Subsection VII-B).

Table IV shows the logic depth, logic delay, routing delay and overall delay for these four architectures. Compared to LUT-4 only architecture, the mix of LUT-4 and macro-gates reduces logic depth and logic delay by 9% and 13%, respectively, and reduces overall delay by 6% by considering

the routing cost. Compared to LUT-6 only architecture, the mix of LUT-6 and macro-gates reduces logic delay by 15% while increasing the logic depth by 50% (due to the fact that an LUT-6 has more implementation capacity than a macro-gate), and reduces overall delay by 7% by considering the routing cost.

For area estimation, we calculate the logic area based on the PLB number and the area modeling in Table III, and estimate the routing area by assuming the routing area is quadratic to the routing delay. Table V shows the PLB number, logic area, routing area and overall area for the four architecture. From Table V, the mix of LUT-4 and macro-gates is 33% larger than the LUT-4 only architecture, while the mix of LUT-6 and macro-gates is 15% smaller than the LUT-6 only architecture.

VII. CONCLUSIONS AND FUTURE WORK

A. Summaries

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 7% and reduces logic area by 15% compared to using 6-input LUTs.

B. Limitations and Future Work

The proposed macro-gate based architecture suffers in that it does not allow full pin-permutation (in a LUT, all input pins are equivalent and can be freely permuted, reducing the amount of flexibility in the routing architecture leading to the logic block). This means that, if the macro-gates are used, extra transistors are required in the cluster architecture to route signals to specific input pins. Some commercial FPGAs have full connectivity within clusters (CLB's), there are also commercial architectures in which this is not the case [5]. This is a major limitation of this work (it can only be used for fully populated clusters). In the future, we will investigate the area impact of using macro-gates on architectures in which the cluster is not fully populated for those architectures, and measure the logic density provided by the new technique. We will also design pin-permutation-aware routing algorithms to compensate with this limitation.

In addition, the proposed macro-gate was found to be effective for HDL designs with a generic synthesis flow. Alternate flows and architectures may make better use of a different macro-gate. This may prove a fruitful area for future investigation.

Moreover, 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 design algorithms for the placement and routing for the proposed architecture, which could give even more accurate architecture evaluation considering.

REFERENCES

- [1] V. Betz, J. Rose, and A. Marquardt, *Architecture and CAD for Deep-Submicron FPGAs*. Kluwer Academic Publishers, Feb 1999.
- [2] "The programmable gate array data book," in *Xilinx*, 1989.
- [3] J. Cong, H. Huang, and X. Yuan, "Technology mapping and architecture evaluation for k/m-macrocell-based FPGAs," *ACM Trans. Design Automation of Electronics Systems*, pp. 3–23, 2005.
- [4] "Xilinx product datasheets," in <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," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, pp. 1712–1724, 2005.
- [7] "Iwls 2005 benchmarks," in <http://iwls.org/iwls2005/benchmarks.html>.
- [8] "ABC: A system for sequential synthesis and verification," in <http://www.eecs.berkeley.edu/~alanmi/abc/>.
- [9] J. Cong and S. Xu, "Delay-optimal technology mapping for FPGAs with heterogeneous LUTs," in *Proc. Design Automation Conf.*, pp. 704–707, June 1998.
- [10] J. Cong, C. Wu, and Y. Ding, "Cut ranking and pruning: Enabling a general and efficient FPGA mapping solution," in *Proc. ACM Intl. Symp. Field-Programmable Gate Arrays*, pp. 29–35, 1999.
- [11] J. He, "Technology mapping and architecture of heterogenous field-programmable gate arrays," *Master thesis, University of Toronto*, 1993.
- [12] A. Kaviani and S. Brown, "Hybrid FPGA architecture," in *Proc. ACM Intl. Symp. Field-Programmable Gate Arrays*, pp. 1–7, 1996.
- [13] K. Chung, "Architecture and synthesis of field-programmable gate arrays with hard-wired connections," *PhD dissertation, University of Toronto*, 1994.
- [14] S. Wilton, "SMAP: Heterogeneous technology mapping for area reduction in FPGAs with embedded memory arrays," in *Proc. ACM Intl. Symp. Field-Programmable Gate Arrays*, pp. 171–178, 1998.
- [15] J. Cong and S. Xu, "Technology mapping for FPGAs with embedded memory blocks," in *Proc. ACM Intl. Symp. Field-Programmable Gate Arrays*, pp. 179–188, 1998.
- [16] Altera Corporation, "Stratix programmable logic device family data sheet," Aug 2002.
- [17] Xilinx Corporation, "Virtex-II 1.5v platform FPGA complete data sheet," July 2002.
- [18] J. Cong and Y. Ding, "Flowmap: An optimal technology mapping algorithm for delay optimization in lookup-table based FPGA designs," in *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, pp. 1–12, 1994.
- [19] J. Cong and Y. Ding, "On area/depth trade-off in LUT-based FPGA technology mapping," in *IEEE Trans. VLSI Syst.*, pp. 137–148, 1994.
- [20] D. Chen and J. Cong, "DAOmap: A depth-optimal area optimization mapping algorithm for FPGA designs," in *Proc. Intl. Conf. Computer-Aided Design*, pp. 752–759, 2004.
- [21] V. Manohara-rajah, S. D. Brown, and Z. G. Vranesic, "Heuristics for area minimization in LUT-based FPGA technology mapping," in *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, pp. 2331–2340, 2006.
- [22] A. Mishchenko, S. Chatterjee, R. Brayton, and P. Pan, "Integrating logic synthesis, technology mapping, and retiming," in *International Workshop on Logic Synthesis*, pp. 161–168, 2005.
- [23] S. Chatterjee, A. Mishchenko, and R. Brayton, "Factor cuts," in *Proc. Intl. Conf. Computer-Aided Design*, pp. 143–150, 2006.
- [24] A. Ling, D. Singh, and S. Brown, "FPGA logic synthesis using quantified boolean satisfiability," in *SAT 2005 Springer LNCS Vol 3569*, pp. 444–450.
- [25] S. Safarpour, A. Veneris, G. Baeckler, and R. Yuan, "Efficient SAT-based boolean matching for FPGA technology mapping," in *Proc. Design Automation Conf.*, pp. 466–471, 2006.
- [26] D. Chai and A. Kuehlmann, "Building a better boolean matcher and symmetry detector," in *Proc. Design Automation and Test Conf. in Europe*, pp. 1079–1084, 2006.

TABLE IV
DELAY EVALUATION FOR DIFFERENT ARCHITECTURES

circuit	Logic depth				Logic delay				Routing delay				Overall delay			
	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	1888	1768	1947	1511	2324	2324	2059	2059	4212	4092	4006	3570
aes_core	8	7	6	9	3776	3224	3894	3235	5819	5819	7383	7383	9595	9043	11277	10618
des_area	10	10	7	10	4720	4520	4543	3884	7707	7707	5714	5714	12427	12227	10257	9598
des_perf	6	6	4	7	2832	2792	2596	2373	9090	9090	6748	6748	11922	11882	9344	9121
ethernet	10	9	7	9	4720	4048	4543	3882	6071	6071	4238	4238	10791	10119	8781	8120
i2c	5	4	4	4	2360	1768	2596	1941	1878	1878	1595	1595	4238	3646	4191	3536
mem_ctrl	11	9	8	9	5192	4008	5192	3882	5894	5894	4513	4513	11086	9902	9705	8395
pci_bridge32	9	8	7	8	4248	3576	4543	3667	5081	5081	3996	3996	9329	8657	8539	7663
pci_sporci_ctrl	6	5	5	5	2832	2200	3245	2373	2453	2453	2509	2509	5285	4653	5754	4882
sasc	3	3	2	4	1416	1376	1298	1294	1670	1670	1265	1265	3086	3046	2563	2559
simple_spi	4	4	3	4	1888	1768	1947	1511	1252	1252	1457	1457	3140	3020	3404	2968
spi	10	9	7	9	4720	4088	4543	3669	4662	4662	4280	4280	9382	8750	8823	7949
ss_pcm	3	3	2	3	1416	1296	1298	1079	734	734	1180	1180	2150	2030	2478	2259
systemcaes	11	11	8	13	5192	4992	5192	5176	5705	5705	5797	5797	10897	10697	10989	10973
systemcdes	9	8	5	9	4248	3696	3245	3235	3238	3238	3149	3149	7486	6934	6394	6384
tv80	17	15	11	15	8024	6800	7139	6689	7323	7323	6542	6542	15347	14123	13681	13231
usb_funct	9	9	6	9	4248	4248	3894	3237	2705	2705	3452	3452	6953	6953	7346	6689
usb_phy	4	3	2	3	1888	1336	1298	1294	1155	1155	974	974	3043	2491	2272	2268
vga_lcd	7	7	6	7	3304	3224	3894	3022	12521	12521	15104	15104	15825	15745	18998	18126
wb_conmax	9	7	6	7	4248	3064	3894	3235	20039	20039	13413	13413	24287	23103	17307	16648
wb_dma	10	9	6	9	4720	3968	3894	3667	8674	8674	6730	6730	13394	12642	10624	10397
GEO	7.11	6.48	4.95	6.79	3358	2918	3210	2734	4033	4033	3712	3712	7687	7223	7126	6633
Ratio1	100%	91%	70%	95%	100%	87%	96%	81%					100%	94%	93%	86%
Ratio2			100%	150%			100%	85%							100%	93%

TABLE V
AREA EVALUATION FOR DIFFERENT ARCHITECTURES

circuit	PLB#				Logic area				Routing area				Overall area			
	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	2005	1739	1449	1523	6.84E+05	6.36E+05	2.03E+06	1.36E+06	1.04E+06	1.10E+06	2.27E+06	2.53E+06	1.72E+06	1.74E+06	4.30E+06	3.89E+06
aes_core	4479	4147	1956	2932	1.53E+06	1.52E+06	2.74E+06	2.62E+06	3.63E+06	4.94E+06	9.84E+06	1.37E+07	5.16E+06	6.46E+06	1.26E+07	1.63E+07
des_area	1076	1004	484	674	3.67E+05	3.67E+05	6.77E+05	6.03E+05	9.79E+05	1.07E+06	1.07E+06	1.31E+06	1.35E+06	1.44E+06	1.75E+06	1.91E+06
des_perf	16185	15652	3436	4863	5.52E+06	5.73E+06	4.81E+06	4.35E+06	5.69E+07	6.07E+07	3.25E+07	3.52E+07	6.24E+07	6.64E+07	3.73E+07	3.95E+07
ethernet	2523	2438	1976	2215	8.61E+05	8.92E+05	2.77E+06	1.98E+06	1.42E+06	2.01E+06	2.41E+06	2.36E+06	2.29E+06	2.90E+06	5.17E+06	4.34E+06
i2c	196	188	132	144	6.69E+04	6.88E+04	1.85E+05	1.29E+05	4.24E+04	7.77E+04	7.00E+04	8.70E+04	1.09E+05	1.46E+05	2.55E+05	2.16E+05
mem_ctrl	1843	1731	1446	1413	6.29E+05	6.33E+05	2.02E+06	1.26E+06	8.11E+05	1.37E+06	1.53E+06	1.71E+06	1.44E+06	2.00E+06	3.55E+06	2.97E+06
pci_bridge32	3298	3156	2706	2911	1.13E+06	1.15E+06	3.79E+06	2.60E+06	1.61E+06	2.33E+06	2.93E+06	3.09E+06	2.74E+06	3.49E+06	6.72E+06	5.70E+06
pci_sporci_ctrl	186	173	125	137	6.35E+04	6.33E+04	1.76E+05	1.23E+05	4.76E+04	7.87E+04	1.05E+05	1.37E+05	1.11E+05	1.42E+05	2.81E+05	2.60E+05
sasc	103	87	73	85	3.52E+04	3.18E+04	1.03E+05	7.61E+04	4.89E+04	4.69E+04	9.77E+04	7.27E+04	8.40E+04	7.87E+04	2.01E+05	1.49E+05
simple_spi	147	121	100	113	5.03E+04	4.43E+04	1.40E+05	1.01E+05	2.21E+04	2.22E+04	7.83E+04	9.40E+04	7.25E+04	6.65E+04	2.18E+05	1.95E+05
spi	657	601	456	440	2.24E+05	2.20E+05	6.39E+05	3.94E+05	2.19E+05	2.86E+05	5.67E+05	5.36E+05	4.43E+05	5.06E+05	1.21E+06	9.30E+05
ss_pcm	62	61	51	48	2.12E+04	2.23E+04	7.14E+04	4.30E+04	5.69E+03	7.17E+03	5.89E+04	5.13E+04	2.69E+04	2.95E+04	1.30E+05	9.43E+04
systemcaes	1474	1470	963	1158	5.03E+05	5.38E+05	1.35E+06	1.04E+06	6.07E+05	7.03E+05	1.68E+06	1.30E+06	1.11E+06	1.24E+06	3.03E+06	2.34E+06
systemcdes	562	580	304	351	1.92E+05	2.12E+05	4.25E+05	3.14E+05	1.12E+05	1.63E+05	4.00E+05	2.98E+05	3.03E+05	3.75E+05	8.26E+05	6.12E+05
tv80	1481	1404	1066	1129	5.06E+05	5.14E+05	1.49E+06	1.01E+06	4.21E+05	5.96E+05	1.25E+06	9.66E+05	9.27E+05	1.11E+06	2.74E+06	1.98E+06
usb_funct	2453	2227	1704	1775	8.37E+05	8.15E+05	2.38E+06	1.59E+06	3.39E+05	3.30E+05	1.87E+06	1.81E+06	1.18E+06	1.15E+06	4.26E+06	3.39E+06
usb_phy	94	89	72	60	3.21E+04	3.26E+04	1.01E+05	5.37E+04	1.20E+04	2.43E+04	5.71E+04	3.04E+04	4.41E+04	5.69E+04	1.59E+05	8.41E+04
vga_lcd	20407	18145	14395	16561	6.96E+06	6.64E+06	2.01E+07	1.48E+07	1.00E+08	1.00E+08	3.03E+08	3.70E+08	1.07E+08	1.07E+08	3.23E+08	3.85E+08
wb_conmax	7290	6966	5501	5895	2.49E+06	2.55E+06	7.70E+06	5.27E+06	5.54E+07	1.09E+08	9.13E+07	9.07E+07	5.79E+07	1.12E+08	9.90E+07	9.60E+07
wb_dma	734	711	564	579	2.51E+05	2.60E+05	7.90E+05	5.18E+05	8.47E+05	1.24E+06	2.36E+06	1.75E+06	1.10E+06	1.50E+06	3.15E+06	2.26E+06
GEO	1025	958	662	733	3.50E+05	3.50E+05	9.28E+05	6.56E+05	5.05E+05	6.69E+05	1.24E+06	1.21E+06	9.80E+05	1.16E+06	2.40E+06	2.05E+06
Ratio1	100%	93%			100%	100%	265%	188%	100%	133%	246%	240%	100%	118%	245%	209%
Ratio2			100%	111%			100%	71%			100%	97%			100%	85%

- [27] S. C. Soheil Ghiasi, Elaheh Bozorgzadeh and M. Sarrafzadeh, "A unified theory of timing budget management," in *Proc. Intl. Conf. Computer-Aided Design*, pp. 653–659, November 2004.
- [28] "Encounter digital ic design platform," in http://www.cadence.com/products/digital_ic/index.aspx.
- [29] "Cadence design tools and 90nm gpdic technology," in <http://www.ee.ucla.edu/~dejan/ee115c/>.
- [30] L. Cheng, F. Li, Y. Lin, P. Wong, and L. He, "Device and architecture cooptimization for FPGA power reduction," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, pp. 1211 – 1221, 2007.
- [31] "Mosek," in <http://www.mosek.com>.
- [32] "MiniSAT," in <http://www.cs.chalmers.se/Cs/Research/FormalMethods/MiniSat/MiniSat.html>.