FPGA Area Reduction by Multi-Output Function Based Sequential Resynthesis

Yu Hu[1], Victor Shih[2], Rupak Majumdar[2] and Lei He[1]
1. Electrical Engineering Department
2. Computer Science Department
University of California, Los Angeles *

ABSTRACT

We propose a new resynthesis algorithm for FPGA area reduction. In contrast to existing resynthesis techniques, which consider only single-output Boolean functions and the combinational portion of a circuit, we consider multi-output functions and retiming, and develop effective algorithms that incorporate recent improvements to SAT-based Boolean matching. Our experimental results show that the proposed resynthesis algorithm reduces area up to 16% compared to the best existing academic technology mapper, Berkeley ABC. Furthermore, considering multi-output functions obtains 19% more area reduction compared to considering single-output functions, and sequential resynthesis obtains 35% more area reduction compared to combinational resynthesis when both consider multi-output functions. At the same time, all results preserve the optimal logic depth produced by the ABC mapper.

Categories and Subject Descriptors

B.7.2 [**Hardware**]: Integrated circuits – Design aids

General Terms

Algorithms, Design

Keywords

Logic synthesis, retiming, FPGA

1. INTRODUCTION

Area minimization is of paramount importance for FPGA synthesis. Since area-optimal technology mapping for LUT-based FPGAs is NP-Hard [1], several heuristics have been proposed for the problem. For example, in [2, 3], critical parts of the circuit are mapped using a depth-minimizing strategy and noncritical parts are either remapped with a duplication-free mapper [2] or mapped using a cost-minimizing strategy that encourages LUT sharing [3]. More recently, iterative optimization has been employed in technology mapping [4, 5, 6], where different metrics such as area flow [4, 5] and exact local area [7] are used in different iterations to guide the area optimization.

Resynthesis [8, 9, 10, 11] is a technique that rewrites circuit structures while maintaining the functionalities of transition and output functions to reduce area. It can be performed simultaneously with technology mapping or as a post-mapping optimization. The simultaneous approaches perform logic resynthesis, such as Boolean decomposition of logic functions [12, 13], during the mapping process. Since they explore a large solution space, the simultaneous approaches tend to be time-consuming, limiting their use to small designs. To handle large designs, resynthesis is usually performed after technology mapping for area recovery. Recently, [14, 15] proposed a combinational resynthesis based on Boolean matching [16] for FPGA area reduction. The resynthesis is performed by mapping logic blocks extracted from a circuit against a library set of logic blocks (e.g., hard-wired LUTs) and replacing them with a functionally equivalent logic block if area can be reduced.

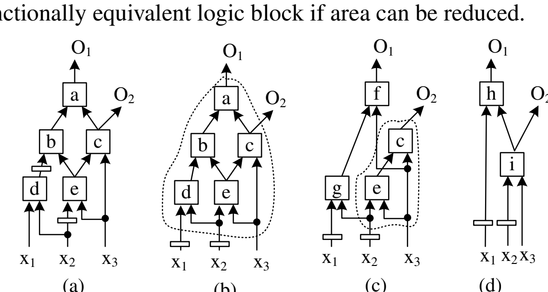


Figure 1: Motivating example for resynthesis considering retiming and multi-output logic blocks

While extremely useful, we argue in this paper that the full potential of resynthesis is yet to be realized. Existing techniques perform resynthesis only within the combinational portion of a circuit, and only consider single-output Boolean functions [14, 15]. We argue that more effective resynthesis can be achieved by additionally considering retiming and multi-output logic blocks. As a motivating example, consider the logic block in Figure 1(a), which contains five 2-input LUTs (a, b, c, d, e) and two registers. If only the combinational portion of the logic block is considered, LUT d must be preserved due to the sequential boundaries defined by the two registers. On the other hand, if retiming is considered, the logic available for resynthesis is expanded as shown in Figure 1(b). Assuming only single-output Boolean functions are considered, suppose the output O_1 driven by five LUTs (a, b, c, d, e) can be implemented with two LUTs (f and g). Logic duplication for the block containing LUT c and e is needed to guarantee the correctness of the secondary output

*This paper is partially supported by NSF grant CCR-0306682, NSF CNS 0720881 and UC MICRO supported by Actel. Address comments to lhe@ee.ucla.edu.

DAC 2008, June 8–13, 2008, Anaheim, California, USA.

O_2 of this logic block as shown in Figure 1(c), resulting in only one LUT reduction. On the other hand, when multi-output Boolean functions are allowed, the resynthesis may reduce the logic block by three LUTs as shown in Figure 1(d), since logic duplication is not necessary. Compared to existing techniques [14, 15], a resynthesis algorithm that considers retiming and multi-output Boolean functions explores a larger solution space and therefore obtains a solution closer to the global optimum.

We make two major contributions in this paper. First, we present a resynthesis algorithm that considers multi-output logic blocks, using an extension to the efficient SAT-based Boolean matching algorithm from [17] to multi-output Boolean functions. Our baseline for experiments is the Berkeley ABC mapper [6], the best existing academic mapper for area optimization with optimal logic depth (based on the study from [18, 7]). Our experimental results show that the new resynthesizer reduces area up to 16% compared to ABC, and it obtains 19% more area reduction compared to the resynthesizer considering single-output logic blocks.

Second, we propose a sequential resynthesis technique to explore an even larger solution space. In contrast to sequential resynthesis for ASICs [10], which requires manual decomposition of the circuit, and the sequential rewriting technique [11], which applies only to And-Inverter Graphs (AIG), our sequential resynthesizer deals with programmable devices directly without the involvement of circuit decomposition or other preprocessing. Our experimental results show that sequential resynthesis obtains 35% more area reduction compared to combinational resynthesis when both consider multi-output logic blocks. At the same time, all our algorithms preserve the optimal logic depth achieved by ABC.

The rest of this paper is organized as follows. Section 2 introduces the preliminaries. Section 3 presents the resynthesis algorithm using multi-output functions. Section 4 proposes a sequential resynthesis for further area reduction. The experimental results are discussed in Section 5 and the paper is concluded in Section 6.

2. PRELIMINARIES

A *programmable logic block (PLB)* $H(P, Q)$ consists of a network of interconnected non-programmable and programmable logic devices with a set P of *input pins* and a set Q of *output pins*. Occasionally we omit the set of input/output pins and simply use H to refer to the PLB $H(P, Q)$. In this paper, we assume that K-input LUTs (with K input pins, one output pin, and 2^K configuration bits) are the only programmable devices in a PLB, but our algorithms and methodology can be easily extended to consider other programmable devices such as programmable MUXs.

The *Boolean Matching* (BM) problem determines, given a PLB $H(P, Q)$ and a multiple output function $f^m(X) = (f_1(X), f_2(X), \cdots, f_m(X))$ over the variables X such that $|X| \leq |P|$ and $m \leq |Q|$, whether the PLB $H(P, Q)$ can implement the function $f^m(X)$. For the simple case where H is a K-LUT, any function $f^m(X)$ where $|X| \leq K$ and $m = 1$ can be implemented by the K-LUT. When H contains multiple LUTs connected with combinational logic, the BM problem becomes non-trivial.

A LUT-based *Boolean network* can be represented using one of the following two forms: (1) a directed acyclic graph (DAG) with nodes corresponding to LUTs and directed edges corresponding to wires connecting the LUTs, where the nodes in the lowest level are called *Circuit Inputs* (CIs) including *Primary Inputs* (PIs) and the outputs of registers, and the nodes in the highest level are called *Circuit Outputs* (COs) including *Primary Outputs* (POs) and the inputs to registers; or (2) a general directed graph with nodes corresponding to LUTs, directed edges corresponding to wires connecting the LUTs, and edge weights denoting the number of registers present on each edge.

A *fanin* (resp. *fanout*) *cone* of node n is a sub-network whose nodes can reach the fanin edges of n (resp. can be reached from the fanout edges of n). A *maximum fanout free cone* (MFFC) of node n is a subset of the fanin cone such that every path from a node in the subset to the PO passes through n. Informally, the MFFC of a node contains all the logic used exclusively by the node. When a node is removed or substituted, its MFFC can be removed.

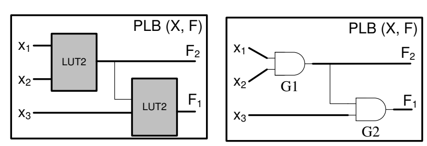
A *cut* C of node n is a set of nodes of the network such that each path from a PI to n passes through at least one node in C; node n is called the *root* of *cut* C. A cut is K-feasible if the number of nodes in it does not exceed K. A *logic block* is a sub-network which covers all nodes found on the path from the outputs (called *root nodes* of the logic block) to the cut, including the roots and excluding the cut. Depending on the number of outputs, a logic block can be *MISO* (multi-input, single-output) or *MIMO* (multi-input, multi-output).

3. RESYNTHESIS WITH MIMO BLOCKS

3.1 SAT-Based BM for MIMO Functions

We first describe the MIMO resynthesis algorithm for combinational blocks. We use SAT-based Boolean matching [14, 19, 15, 17] to determine whether a Boolean function f can be implemented using a specific logic block. We favor SAT-based Boolean matching over other Boolean matching techniques for two reasons. First, its flexibility allows handling all types of heterogeneous logic blocks in FPGAs such as LUTs of different sizes, mixed LUTs and PLAs, and mixed LUTs and macro-gates [20]. Second, recent work [15, 17] has significantly reduced Boolean matching runtime, making their use practical for FPGA synthesis. We now extend these algorithms to allow multi-output logic functions and multi-output PLBs. To illustrate, consider the example shown in Figure 2. Multi-output Boolean matching determines whether the 2-output Boolean function $f^2(x_1, x_2, x_3)$, described by the truth table in Figure 2(a), can be implemented by the PLB shown in Figure 2(b), containing two LUT-2s. Let $X = \{x_1, x_2, x_3\}$ be the set of input pins. Central to SAT-based Boolean matching is *SAT encoding*, the process of formulating the matching problem for heterogeneous PLBs as a Boolean expression in conjunctive normal form (CNF) which is satisfiable if and only if the function is implementable. This is performed by the following steps.

$x_1x_2x_3$	f_1	f_2
000	0	0
001	0	0
010	0	0
011	0	0
100	0	0
101	0	0
110	0	1
111	1	1



(a) (b) (c)

Figure 2: Example for SAT encoding (a) Truth table for f, (b) Target PLB, (c) Mapped PLB

Create CNF formulas for individual elements in the PLB. A single LUT-2 has the following characteristic function:

$$
\begin{aligned}
G_{LUT}[i_1, i_2, F] \quad = \quad & (i_1 + i_2 + \overline{L_1} + F)(i_1 + i_2 + L_1 + \overline{F}) \\
& (i_1 + \overline{i_2} + \overline{L_2} + F)(i_1 + \overline{i_2} + L_2 + \overline{F}) \\
& (\overline{i_1} + i_2 + \overline{L_3} + F)(\overline{i_1} + i_2 + L_3 + \overline{F}) \\
& (\overline{i_1} + \overline{i_2} + \overline{L_4} + F)(\overline{i_1} + \overline{i_2} + L_4 + \overline{F})
\end{aligned}
$$

with i_1, i_2 denoting its inputs, F denoting its output, and $L_1, \cdots, L_4$ denoting the configuration bits of the LUT. The characteristic function of the PLB is then formulated as:

$$G = G_{LUT_1}[x_1, x_2, F_2] \cdot G_{LUT_2}[F_2, x_3, F_1] \quad (1)$$

Replicate (1) according to all possible values of the input variables X and output variables F_1 and F_2. G_{SAT} is then

$$\begin{aligned} G_{SAT} \quad = \quad & G[X/000, F_1/0, F_2/0] \cdot G[X/001, F_1/0, F_2/0] \cdot \\ & G[X/010, F_1/0, F_2/0] \cdot G[X/011, F_1/0, F_2/0] \cdot \\ & G[X/100, F_1/0, F_2/0] \cdot G[X/101, F_1/0, F_2/0] \cdot \\ & G[X/110, F_1/0, F_2/1] \cdot G[X/111, F_1/1, F_2/1] \end{aligned}$$

A satisfiable assignment to G_{SAT} implies that f can be implemented by the PLB. In this example, the SAT solver indicates that this problem is satisfiable and both LUTs are configured to 2-input AND gates, as shown in Figure 2(c).

An important consideration for Boolean matching is the problem of input/output permutation, that is, how to map the input and output function variables to the input and output pins of the target PLB. In the example, the problem is unsatisfiable if the output variables f_1 and f_2 are incorrectly mapped to the PLB output pins F_2 and F_1, respectively.

In [17] the problem of input permutations is reduced considerably by pruning permutations that are redundant due to functional and architectural symmetries. Unfortunately, this pruning technique cannot be extended to multi-output functions with regard to output permutations, as each output permutation represents an additional possible solution to the SAT problem. Note that although a circuit may exhibit architectural symmetry with respect to its output pins, additional pruning is not possible because the symmetry has already been accounted for with respect to the input pins. Thus for a circuit with m outputs, the number of mappings to be tested increases by a factor of $m!$. Although output symmetry cannot be exploited, other characteristics of the problem can be used to eliminate some mappings from consideration. For example, it can be detected that f_1 cannot be mapped to F_2 because its truth table requires at least three variables to implement, yet F_2 is determined by only two inputs.

3.2 Combinational Resynthesis Algorithm

The *combinational resynthesis* procedure takes an application mapped to K-LUTs and scans the combinational portion of the circuit, in topological order from CIs to COs.[1] Similar to [14], new logic blocks are generated by combining the logic blocks at the input LUTs, and several heuristics are employed to speed up logic block generation [7]. MIMO logic blocks are mapped against PLBs in the resynthesis library; if a mapping is found satisfiable by the Boolean matching, the logic block can be reduced to the library PLB without logic duplication. Note that the logic depth will not increase since any substitution that increases the local logic depth will be discarded beforehand.

In MIMO resynthesis, there is a risk that a substitution will introduce a combinational cycle in the circuit. Consider the MIMO logic block consisting of nodes 1, 2 and 4 (shaded portion) in Figure 3(a). Depending on the logic implemented, Boolean matching may determine that this logic block can be implemented by a two node logic block (6 and 7, in Figure 3(b)), saving one LUT. However, this substitution forms the combinational cycle $5 \rightarrow 6 \rightarrow 7 \rightarrow 3 \rightarrow 5$.

[1]Note that other graph traversal orders, such as reverse topological order or window-based traversal [21], can be used as well. We observe similar area reduction across different graph traversal strategies.

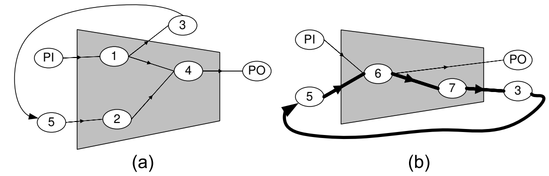


Figure 3: A MIMO extracted from IWLS'05 benchmark circuit "pci_conf_cyc_addr_dec1" (a) before resynthesis (b) after resynthesis

This problem does not occur if resynthesis is limited to MISO logic blocks. Essentially, MIMO resynthesis can introduce *new reverse paths* to the circuit. Note that in Figure 3 there is a path $3 \rightsquigarrow 5$ outside the logic block before resynthesis, and that resynthesis created a new path $5 \rightsquigarrow 3$ inside the logic block. Together, these paths form a combinational cycle. To avoid this scenario, substitutions for MIMO logic blocks are tested prior to Boolean matching. For each input/output node pair in_i and out_j, connectivity tests are performed inside and outside the logic block. If there is a path $out_j \rightsquigarrow in_i$ outside the logic block and a path $in_i \rightsquigarrow out_j$ inside the new logic block, this resynthesis step is discarded and the Boolean matching is not performed. Note that the connectivity of node pairs in the original circuit and library PLBs can be precomputed, and that the connectivity outside the logic block can be updated incrementally during the resynthesis.

4. SEQUENTIAL RESYNTHESIS

The resynthesis described in the previous section is limited to the combinational portion of the circuit. Its effectiveness is thus limited by the sequential boundaries defined by the registers in the circuit. Redefining register boundaries by retiming and optimizing the resulting circuit using combinational resynthesis may potentially lead to larger reduction than combinational resynthesis alone.

Given a logic block (without cycles) extracted from a circuit mapped by LUTs, *sequential resynthesis* performs retiming and resynthesis (for the combinational portion of the logic block) simultaneously to reduce the area (number of LUTs) while preserving functionality, i.e., the output sequence remains unchanged given the same input vector and proper initial states of all flip-flops.

Unlike existing sequential resynthesis techniques [10, 11], our sequential resynthesizer processes local retiming and resynthesis without involving any manual decomposition or preprocessing of circuits. Furthermore, it takes the reconfigurability of the programmable devices into consideration to maximize the effectiveness of the resynthesis for area reduction.

4.1 Retiming and Resynthesis Algorithm

The sequential resynthesis is performed in topological order from PIs to POs. For each node v, a sequential logic block enumeration algorithm is called and logic blocks with v as the root are extracted (logic blocks with multiple fanouts are allowed). In our sequential resynthesis framework, only acyclic logic blocks are extracted and resynthesized. Algorithm 1 shows the sequential logic block enumeration algorithm, which is an extension of the combinational cut enumeration algorithm in [22]. It takes a node v and calculates K-feasible cuts whose logic depths are less than $Depth$. Tha algorithm starts from the root node v and iteratively search backwards based on v's fanins. For a node with multiple fanins, the cuts rooted at each of those fanins are merged based [22] (line 8 in Algorithm 1). According to the structure of the logic blocks, the following three cases need to be considered in the sequential resynthesis for LUT-based FPGAs.

Algorithm 1 enumSeqCuts$(v, K, Depth, visited)$

{//enumerate K-feasible sequential cuts}
if $v \in visited$ or $Depth = 0$ **then**
  return NULL;
$visited$.insert(v);
if v is LUT **then**
  for all $n_i \in$ fanins of v **do**
    $\Phi(n_i) =$ **enumSeqCuts**$(n_i, K, Depth - 1, visited)$;
  $\Phi(v) = \{n\} \cup \Phi(n_1) \triangle \cdots \triangle \Phi(n_m)$
  $\{//A \triangle B = \{u \cup v | u \in A, v \in B, |u \cup v| \leq K\}\}$
  return $\Phi(v)$
else if v is LATCH **then**
  $n_1 = v$.fanins[0]
  if $n_1 \in$ visited **then**
    return NULL;
  return **enumSeqCuts**$(n_1, K, Depth - 1, visited)$;
else if v is PO **then**
  return **enumSeqCuts**$(v.fanins[0], K, Depth - 1, visited)$;
else
  return NULL;

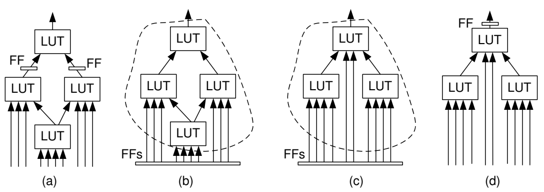


Figure 4: Seq resynthesis case I: (a) Original logic block (b) Backward retimed logic block (c) Combinationally resynthesized logic block (d) Forward retimed resynthesized logic block.

Case 1: Classical Retiming without Duplication. The simplest scenario is that a logic block is resynthesizable after performing classical retiming [23] without any logic duplication. After retiming, the weights (i.e., the number of registers) of all edges are non-negative. Figure 4 shows an example of such a case. All registers are pushed to the inputs of the logic block, and the combinational portion of the logic block is maximized and resynthesized using techniques presented in Section 3. After combinational resynthesis, the registers are pushed forward maximally and the number of registers is minimized.

Case 2: Peripheral Retiming without Duplication. For certain MIMO logic blocks, classical retiming, which requires a non-negative number of registers during retiming, cannot be performed. For example, in Figure 5(a), the registers between two LUTs cannot be moved either forward or backward by classical retiming. In this case, a peripheral retiming [10] can be used to maximize the combinational portion of the logic block as shown in Figure 5(b). Note that a peripheral retiming may result in a negative number of registers at the peripheral edges of a logic block (the one driving output out_2 in the example). The negative number of registers can be thought of as borrowing registers from the environment outside of the logic block; a forward retiming after the resynthesis will recover these negative peripheral edges, making this transformation transparent to the outside of the resynthesized logic block.

It is possible that combinational resynthesis will create an infeasible solution. In Figure 5(c), for example, there is no way to recover the negative edge using a local forward retiming. This resynthesis step is discarded to guarantee the existence of a valid forward retiming, as in Figures 5(d) and (e).

In fact, the case mentioned in Case 1 can be viewed as a special case of peripheral retiming without duplication. As proved in [10], sufficient and necessary conditions for peripheral retiming without logic duplication is as follows.

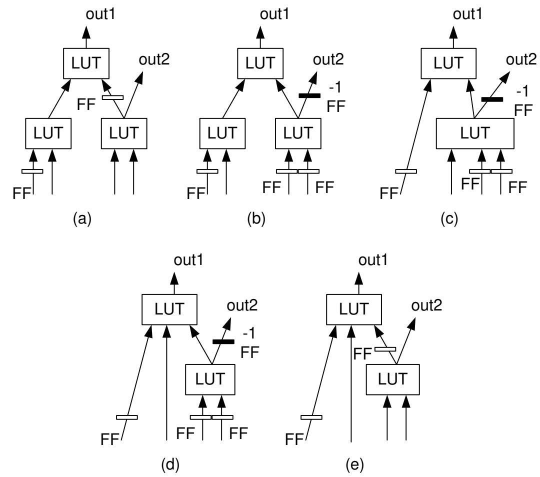


Figure 5: Seq resynthesis case II: (a) Original logic block (b) Peripheral retimed logic block (c) Combinationally resynthesized logic block (with no feasible forward retiming) (d) Combinationally resynthesized logic block (with feasible forward retiming) (e) Forward retimed resynthesized logic block.

THEOREM 1. *An acyclic graph G with m inputs and n outputs has a peripheral retiming without duplication if and only if (a) for input $i \in \{1, \cdots, m\}$ and output $j \in \{1, \cdots, n\}$ of G, all paths $(i \rightsquigarrow j)$ must have the same number of registers, and (b) there exist numbers $\{\alpha_i \in \mathbb{Z} \mid i \in \{1, \cdots, m\}\}$ and $\{\beta_j \in \mathbb{Z} \mid j \in \{1, \cdots, n\}\}$ such that for each input/output pair i and j and every path from i to j, the number of registers in the path is equal to $\alpha_i + \beta_j$.*

Theorem 1 states that a peripheral retiming is equivalent to finding α_i, β_j, a set of integer assignments for the number of registers in the peripheral edges based on the number of registers in input-to-output paths in the graph. Consider the example shown in Figure 5(a); the number of registers in each input-to-output path can be expressed by the 4×2 matrix

$$\begin{bmatrix} \alpha_1 + \beta_1 & \alpha_2 + \beta_1 & \alpha_3 + \beta_1 & \alpha_4 + \beta_1 \\ * & * & \alpha_3 + \beta_2 & \alpha_4 + \beta_2 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1 & 1 \\ * & * & 0 & 0 \end{bmatrix} \tag{2}$$

where "*" in (i, j) denotes that no path exists between input i and output j. Satisfiable assignments for α_i, β_j are $\alpha_1 = 1, \alpha_2 = 0, \alpha_3 = 1, \alpha_4 = 1, \beta_1 = 0, \beta_2 = -1$, which result in the peripheral retiming shown in Figure 5(b).

Accordingly, the necessary and sufficient conditions of classical retiming without logic duplication in Case 1 are similar to Theorem 1 except that $\alpha_i, \beta_j \in \mathbb{N}$, i.e., α_i, β_j must be non-negative integers. It is easy to show that there is no non-negative integer solution for Equation (2); therefore no feasible classical retiming exists for Figure 5(a).

As shown in [10], the computation of this assignment needs $O(e \cdot \min(m, n))$ time, where e is the number of edges and m, n are the number of inputs and outputs in the graph, respectively. Since the sequential resynthesis is performed locally in a graph of limited size, the runtime for checking the feasibility of peripheral retiming is negligible.

Case 3: Peripheral Retiming with Duplication. If the conditions in Theorem 1 are not satisfiable, logic duplication is employed to enable peripheral retiming. Without loss of generality, a single-output logic block is considered for illustration, as shown in Figure 6(a). First, the inputs are duplicated for each input with different numbers of registers in multiple paths to the output. For example, there are two paths from x_2 to output, where the numbers of registers are 1 and 0, respectively, so x_2 is duplicated to x_2^1 and x_2^0. The label in a duplicated input denotes the number of registers in its path to the output. In other words, the duplicated inputs represent the values of the input in different time frames. Figure 6(b) shows the result of input duplication, where all LUTs driven by the duplicated inputs are also duplicated. This duplication enables peripheral retiming as shown in Figure 6(c). The combinational portion can be resynthesized to Figure 6(d), where one LUT is saved. During the resynthesis, the LUTs driven by the same group of duplicated inputs should have identical configuration bits, so that they can be merged after sequential resynthesis to restore the original inputs. This can be implemented as a constraint in the SAT encoding during the Boolean matching.

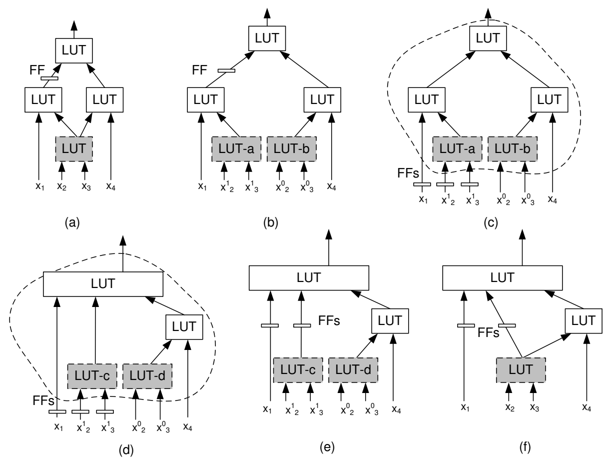


Figure 6: Seq resynthesis case III: (a) Original logic block (b) Logic block after duplication (c) Peripheral retimed logic block (d) Combinationally resynthesized logic block (e) Forward retimed resynthesized logic block (f) The final result after merging LUTs.

The overall algorithm is shown in Algorithm 2.

Algorithm 2 resynthesisSeq($Network$)

for each node n of $Network$ in topological order **do**
  $visited \leftarrow \emptyset$
  $cuts$ = **enumSeqCuts**($n, K, Depth, visited$)
  for each cut $c \in cuts$ **do**
    $b \leftarrow$logic block bounded by c
    Discard b if no area reduction is achievable
    $class \leftarrow$**classifyByTheorem1**(b)
    $b^r \leftarrow$**peripheralRetimingAndDuplication**($class$)
    $b^n \leftarrow$**SAT-BooleanMatching**(b^r)
    if b^n is feasible (see Section 4.2) **then**
      UpdateNetwork($Network$)
      break

4.2 Post Resynthesis Feasibility Checking

Feasibility Checking I: Initial State Computation. For a design including registers with asynchronous clear/preset, it is usually difficult to determine the proper clear/reset conditions after moving the registers through the logic. This is a known difficulty with retiming and similar techniques. The resynthesized circuit is *sequentially equivalent* to the original circuit if there exists an initialization sequence which brings the resynthesized circuit from an arbitrary (power-up) state to a known initial state of the original circuit. In general, it may be impossible to initialize the sequentially resynthesized circuit to the initial state of the original circuit, unless certain retiming steps are disabled. A condition to preserve feasibility of initialization has been derived in [11] in sequential rewriting for AIGs. Similarly, a necessary and sufficient condition to preserve sequential equivalence during the sequential resynthesis for LUT-based FPGAs is given in Theorem 2, which can be proven by induction. Based on Theorem 2, certain resynthesis steps must be discarded when the *sequential resynthesis invariant* is not satisfiable, which can be determined by the SAT solver.

THEOREM 2. *Express each register in terms of the cut variables in different time frames, i.e., if π_w denotes the cut variables in different time frames, then each register can be expressed as $r_i^w = f_i(\pi_w)$. The sequential resynthesis invariant for step w is $I_w(r^w) = \exists_{\pi_w} \bigwedge_i [r_i^w = f_i^w(\pi_w)]$. The sequential equivalence is preserved if and only if the sequential resynthesis invariant is satisfiable for each step.*

Feasibility Checking II: Clock Period Violation. To preserve the logic depth Φ in the original circuit during the sequential resynthesis, we use the *l-value* [?] (conceptually the same as the combinational arrival time) to keep tracking the current clock period. If the l-values of the new nodes in the resynthesized logic block are larger than the target clock period Φ, this resynthesis is simply discarded.

5. EXPERIMENTAL RESULTS

Our resynthesizer (including both combinational and sequential resynthesis algorithms) has been implemented using the OAGear package [24]. The SAT-based Boolean matcher is implemented in C++ and uses the miniSAT2.0 package [25]. The resynthesis framework accepts both blif and Verilog as inputs. The biggest 20 MCNC [26] benchmarks, including 10 combinational and 10 sequential applications, are tested. The circuits are first mapped to LUT-4 by Berkeley ABC mapper [6]. In our experiments, one traversal from PIs to POs for each circuit is performed. Logic blocks with no more than 10 inputs are extracted, and the three library PLBs shown in Figure 7 are considered, which are all possible PLB structures with no more than 10 inputs and 4 LUT-4s [14]. Our MIMO resynthesis considers outputs of all LUTs in each library PLB as the outputs of the MIMO blocks. All resynthesized results have been verified by the ABC verification tool set.

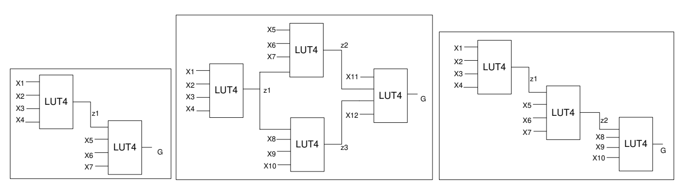


Figure 7: PLB templates used in resynthesis

Table 1 compares the MISO (column "MISO") and MIMO (column "MIMO") resynthesis by the 10 combinational applications. The MIMO resynthesis reduces area by 0.76% (up to 1.41% for application "spla") on average when compared to the ABC mapper, and it achieves 19% more area reduction with 15× runtime overhead compared to the MISO resynthesis on average. Note that the effectiveness of the MIMO resynthesis heavily depends on the structure of the combinational logics in an application and the PLB templates used for resynthesis. For example, MIMO resynthesis reduces 2× area compared to MISO based approach for applications "apex4" and "misex3", while MIMO and MISO resynthesis obtains the same results for applications "ex5p" and "spla".

Table 2 presents the results for the sequential circuits, where the MIMO resynthesis is used for both combinational (column "Comb") and sequential resynthesis (column "Seq"). Compared to the ABC mapper[2], sequential resynthesis reduces area by 5.07% (up to 15.91% for application "s298") on average. Compared with the combinational version, the sequential resynthesis achieves 35% more area reduction with 2× runtime overhead. Note that the runtime for sequential resynthesis depends on the circuit structure. Intuitively, since sequential resynthesis considers more logic blocks, and sequential computation for each logic block requires more time (see Section 4), we would expect sequential resynthesis runtime to be longer than that of combinational resynthesis. However, results counter to this expectation were obtained for certain circuits such as "bigkey" and "s298". The reason is that sequential resynthesis breaks the register boundaries to explore a larger solution space, and therefore has the potential to achieve faster convergence.

Benchmarks		Resynthesized LUT#		Runtime (min)	
	LUT#	MISO	MIMO	MISO	MIMO
alu4	720	716 (-0.56%)	714 (-0.83%)	29	870
apex2	965	959 (-0.62%)	958 (-0.73%)	14	429
apex4	791	790 (-0.13%)	789 (-0.25%)	1	247
des	1249	1243 (-0.48%)	1242 (-0.56%)	69	347
ex1010	1103	1090 (-1.18%)	1090 (-1.18%)	55	251
ex5p	541	538 (-0.55%)	538 (-0.55%)	1	58
misex3	736	733 (-0.41%)	730 (-0.82%)	8	406
pdc	2210	2194 (-0.72%)	2191 (-0.86%)	21	241
seq	998	995 (-0.30%)	994 (-0.40%)	2	434
spla	2126	2096 (-1.41%)	2096 (-1.41%)	38	198
ave	1144	1135 (-0.64%)	1134 (-0.76%)	24	348

Table 1: Resynthesis results for combinational circuits (MISO vs. MIMO)

Benchmarks			Resynthesized LUT#		Runtime(min)	
Circuit	LUT#	Latch#	Comb	Seq	Comb	Seq
bigkey	1261	224	1261 (0.00%)	1244 (-1.35%)	2709	1898
clma	4210	33	4167 (-1.02%)	4116 (-2.23%)	2697	3825
diffeq	674	305	674 (0.00%)	673 (-0.15%)	655	856
dsip	1554	224	1330 (-14.41%)	1338 (-13.90%)	705	1481
elliptic	441	194	419 (-4.99%)	419 (-4.99%)	32	370
frisc	2841	886	2660 (-6.37%)	2595 (-8.66%)	1364	1537
s298	44	14	41 (-6.82%)	37 (-15.91%)	186	125
s38417	3134	1462	3105 (-0.93%)	3117 (-0.54%)	3466	6092
s38584	3720	1260	3654 (-1.77%)	3655 (-1.75%)	2867	8363
tseng	946	385	935 (-1.16%)	934 (-1.27%)	1331	1492
ave	1883	499	1825 (-3.75%)	1813 (-5.07%)	1601	2604

Table 2: Resynthesis results for sequential circuits with MIMO blocks

6. CONCLUSIONS AND FUTURE WORK

We have proposed a new resynthesis algorithm that considers multi-output logic blocks and retiming. The new technique reduces up to 16% area (MCNC benchmark "s298") compared to the state-of-the-art technology mapper, and it obtains up to 1.3× more area reduction (MCNC benchmark "s298") compared to the exisiting resynthesis algorithm.

In our study, we find that the effectiveness of different resynthesis options heavily depends on the structures of logics in the applications and the PLB templates used in the resynthesis. In the future, we will study the essence of the correlation between the effectiveness of our proposed resynthesis and the logic and PLB strucutres for further improvement. In addition, due to the prevalence of existing FPGAs with heterogeneous PLBs, we hope to port the proposed techniques to technology mapping and resynthesis of new FPGA architectures such as those with mixed LUTs and macro-gates [20], where we again expect improvements over existing techniques.

[2] Note that the "-s" option for the ABC's "if" command enables simultaneous retiming and technology mapping. However, it optimizes for delay minimization and produces results with larger area; therefore it was not used for comparison.

7. REFERENCES

[1] A. Farrahi and M. Sarrafzadeh, "Complexity of the lookup-table minimization problem for fpga technology mapping," vol. 13, no. 11, pp. 1319–1332, 1994.

[2] J. Cong and Y. Ding, "On area/depth trade-off in lut-based FPGA technology mapping," in *Proc. Design Automation Conf.*, pp. 213–218, 1993.

[3] J. Cong and Y.-Y. Hwang, "Simultaneous depth and area minimization in lut-based fgpa mapping," in *Proc. ACM Intl. Symp. Field-Programmable Gate Arrays*, 1995.

[4] V. Manohararajah, S. D. Brown, and Z. G. Vranesic, "Heuristics for area minimization in lut-based fpga technology mapping," 2004.

[5] D. Chen and J. Cong, "Daomap: A depth-optimal area optimization mapping algorithm for fpga designs," in *Proc. Intl. Conf. Computer-Aided Design*, 2004.

[6] "Abc: A system for sequential synthesis and verification," in *http://www.eecs.berkeley.edu/ alumni/abc/*.

[7] A. Mishchenko, S. Chatterjee, and R. Brayton, "Improvements to technology mapping for lut-based fpgas," in *Proc. ACM Intl. Symp. Field-Programmable Gate Arrays*, 2006.

[8] G. D. Micheli, "Synchronous logic synthesis: algorithms for cycle-time minimization," 1991.

[9] A. Mishchenko, S. Chatterjee, and R. Brayton, "Dag-aware aig rewriting," in *Proc. Design Automation Conf.*, 2005.

[10] S. Malik, E. Sentovich, R. Brayton, and A. Sangiovanni-Vincentelli, "Retiming and resynthesis: Optimizing sequential networks with combinational techniques," vol. 10, pp. 74–84, 1991.

[11] R. Brayton and A. Mishchenko, "Sequential rewriting," in *International Workshop on Logic Synthesis*, 2007.

[12] E. Lehman, Y. Watanabe, J. Grodstein, and H. Harkness, "Logic decomposition during technology mapping," vol. 16, no. 8, pp. 813–833, 1997.

[13] A. Mishchenko, X. Wang, and T. Kam, "A new enhanced constructive decomposition and mapping algorithm," in *Proc. Design Automation Conf.*, 2003.

[14] A. Ling, D. Singh, and S. Brown, "FPGA technology mapping: a study of optimality," in *Proc. Design Automation Conf.*, 2005.

[15] J. Cong and K. Minkovich, "Improved sat-based boolean matching using implicants for lut-based fpgas," in *Proc. ACM Intl. Symp. Field-Programmable Gate Arrays*, 2007.

[16] L. Benini and G. D. Micheli, "A survey of Boolean matching techniques for library binding," *ACM Transactions on Design Automation of Electronic Systems*, vol. 2, no. 3, pp. 193–226, 1997.

[17] Y. Hu, V. Shih, R. Majumdar, and L. He, "Exploiting symmetry in satbased boolean matching for heterogeneous fpga technology mapping," in *Proc. Intl. Conf. Computer-Aided Design*, 2007.

[18] K. Minkovich and J. Cong, "Optimality study of logic synthesis for lut-based fpgas," in *Proc. ACM Intl. Symp. Field-Programmable Gate Arrays*, 2006.

[19] S. Safarpour, A. Veneris, G. Baeckler, and R. Yuan, "Efficient sat based boolean matching for fpga technology mapping," in *Proc. Design Automation Conf.*

[20] Y. Hu, S. Das, S. Trimberger, and L. He, "Design, synthesis and evaluation of heterogeneous fpga with mixed luts and macro-gates," in *Proc. Intl. Conf. Computer-Aided Design*, 2007.

[21] A. Mishchenko and R. Brayton, "Sat-based complete dontcare computation for network optimization," in *Proc. Design Automation and Test Conf. in Europe*, 2005.

[22] 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*, 1999.

[23] C.E.Leiserson and J.B.Saxe, "Retiming synchronous circuitry," *Algorithmica*, pp. 5–35, 1991.

[24] Z. Xiu, D. A. Papa, P. Chong, A. Kuehlmann, R. A. Rutenbar, and I. L. Markov, "Early research experience with openaccess gear:," in *Proc. Intl. Symp. Physical Design*, 2005.

[25] N. Een and N. Sorensso, *http://www.cs.chalmers.se/Cs/Research/FormalMethods/MiniSat/MiniSat.html*.

[26] MCNC, *MCNC Designers' Manual*, 1993.