CHAPTER 13

TECHNOLOGY MAPPING

Jason Cong
Department of Computer Science
California NanoSystems Institute
University of California–Los Angeles

Peichen Pan
Magma Design Automation, Inc.

Technology mapping is an essential step in an field-programmable gate array (FPGA) design flow. It is the process of converting a network of technology-independent logic gates into a network comprising logic cells on the target FPGA device. Technology mapping has a significant impact on the quality of the final FPGA implementation.

Technology-mapping algorithms have been proposed for optimizing area [29, 36, 58, 65], timing [9, 12, 13, 19, 21, 37, 58], power [2, 8, 34, 45, 52, 71], and routability [3, 67]. Mapping algorithms can be classified into those for general networks [13, 16] and those for special ones such as treelike networks [35, 36]. Algorithms for special networks may be applied to general ones through partitioning, with a possible reduction in solution quality.

Technology-mapping algorithms can be *structural* or *functional*. A structural mapping algorithm does not modify the input network other than to duplicate logic [12, 13]. It reduces technology mapping to a covering problem in which the technology-independent logic gates in the input network are covered with logic cones such that each cone can be implemented using one logic cell—for example, a K-input lookup table (K-LUT)—for LUT-based FPGAs. Figure 13.1 is an example of structural mapping. The logic gates in the original network (a) are covered with three logic cones, each with at most three inputs, as indicated (b). Note that node i is included in two cones and will be duplicated. The corresponding mapping solution (c) comprises three 3-LUTs.

A functional mapping algorithm, on the other hand, treats technology mapping in its general form as a problem of Boolean transformation/decomposition of the input network into a set of interconnected logic cells [15, 48, 58, 60]. It mixes Boolean optimization with covering. Functional mapping algorithms tend to be time consuming, which limits their use to small designs or to small portions of a design.

Note: This work is partially supported by the National Science Foundation under grant number CCF 0530261.

Recent advances in technology mapping try to combine mapping with other steps in the design flow. Such integrated mapping algorithms have the potential to explore a larger solution space than is possible with just technology mapping and thus have the potential to arrive at mapping solutions with better quality. For example, algorithms have been proposed to combine logic synthesis with covering to overcome the limitations of pure structural mapping [11, 22, 57].

13.1 STRUCTURAL MAPPING ALGORITHMS

Technology mapping is part of a logic synthesis flow, which typically consists of three steps. First, the initial network is optimized using technology-independent optimization techniques such as node extraction/substitution and don't-care optimization [33]. Second, the optimized network is decomposed into one consisting of 2-input gates plus inverters (that is, the network is *2-bounded*) to increase flexibility in mapping [12, 36]. Third, the actual mapping takes place, with the goal of covering the 2-bounded network with *K*-LUTs while optimizing one or more objectives. In the remaining discussion, we assume that the input network is 2-bounded.

A logic network can be represented as a graph where the nodes represent logic gates, primary inputs (PIs), and primary outputs (POs). The edges represent the interconnects or wires. A *cut* of a node v is a set of nodes in the input network such that every path from the primary inputs or sequential element outputs to v contains at least one node in the set. A *K-cut* is a cut with at most K nodes. For example, $\{a, b, z\}$ is a 3-cut for the node y in the network in Figure 13.1(a). Given a K-cut for v, we can obtain a K-LUT for v by collapsing the gates in the logic cone between the nodes in the cut, v, including v itself. For the 3-cut $\{a, b, z\}$ for y, the 3-LUT for y in Figure 13.1(c) is derived from the corresponding cone indicated for y in Figure 13.1(b).

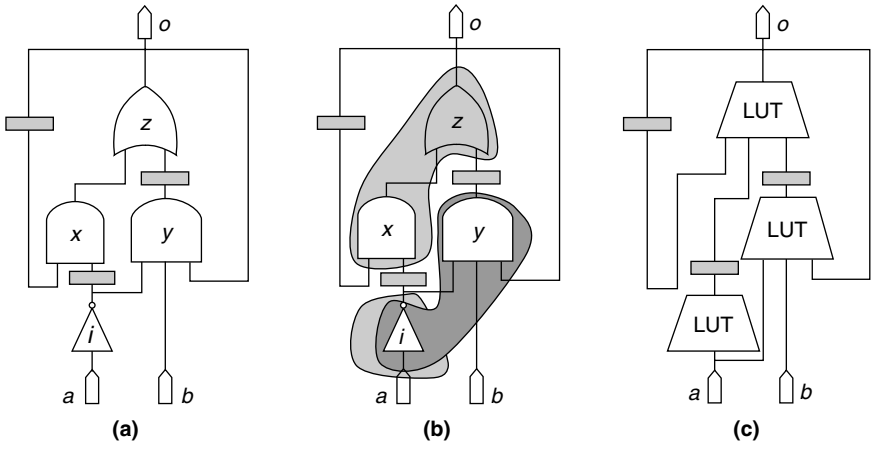

FIGURE 13.1 ■ Structural technology mapping: (a) original network, (b) covering, and (c) mapping solution.

Most structural mapping algorithms are based on the dynamic programming technique. They typically consist of the following steps:

1. Cut generation/enumeration
2. Cut ranking
3. Cut selection
4. Final mapping solution generation

Cut generation obtains one or more cuts that will be used to generate LUTs; it is discussed in the next section. Cut ranking evaluates the cuts obtained in cut generation to see how good they are based on the mapping objectives. It assigns a label or cost to each cut by visiting the nodes in a topological order from PIs to POs. Cut selection picks a cut with the best label for each node and is typically done in reverse topological order from POs to PIs. Cut ranking and selection may be carried out multiple times to refine solution quality.

After the final cut selection, a mapping solution is generated using the selected cuts. In this step, the nodes are visited in the reverse topological order, starting from POs and going back to PIs. At each node, a cut with the best label is selected and the corresponding LUT is added to the solution. Next, the nodes that drive the LUT are visited. This process is repeated until only PIs are left. At that point, a complete mapping solution is obtained.

13.1.1 Cut Generation

Early mapping algorithms combine cut generation and selection to determine one or a few "good" cuts for each node. The most successful example is the FlowMap algorithm, which finds a single cut with optimal mapping depth at each node via max-flow computation [16]. It computes the optimal mapping depth of each node in a topological order from PIs to POs, and at each node uses a max-flow formulation to test whether that node can have the same optimal mapping depth as the maximum depth of its input nodes. If not, the depth is set to one greater than the input nodes' maximum depth. It is shown that these are the only two possible mapping depths. The FlowMap algorithm was the first polynomial time algorithm to find a depth-optimal mapping solution for LUT-based FPGAs.

In practice, K, the number of inputs of the LUTs, is a small constant typically ranging between 3 and 6. It becomes practical to enumerate all K-cuts for each node. With all cuts available, we have additional flexibility in selecting cuts to optimize the mapping solution.

Cuts can be generated by a traversal of the nodes in a combinational network (or the combinational portion of a sequential network) from PIs to POs in a topological order [29, 67]. Let $\Phi(v)$ denote the set of all K-cuts for a node v. For a PI, $\Phi(v)$ contains only the trivial cut consisting of the node itself, that is, $\Phi(v) = \{\{v\}\}$. For a non-PI node v with two fanin nodes, u_1 and u_2, $\Phi(v)$ can be computed by merging the sets of cuts of u_1 and u_2 as follows:

$$\Phi(v) = \{\{v\} \cup \{c_1 \cup c_2 | c_1 \in \Phi(u_1),\ c_2 \in \Phi(u_2),\ |c_1 \cup c_2|\} <= K\} \tag{13.1}$$

In other words, the set of cuts of v is obtained by the pairwise union of the cuts of its fanin nodes and then the elimination of those cuts with more than K nodes. Note that the trivial cut is added to the set. This is necessary so the nodes driven by v can include v in their cuts.

13.1.2 Area-oriented Mapping

For LUT mapping, the area of a mapping solution can be measured by the total number of LUTs. It has been shown that finding an area-optimal mapping solution is NP-hard [35]. Therefore, it is unlikely that there is an accurate way to rank cuts for area. The difficulty of precise area estimation is mainly due to the existence of multiple fanout nodes. In fact, for treelike networks, area-optimal mapping solutions can be determined in polynomial time [35].

Cong et al. [29] proposed the concept of *effective area* as a way to rank and select cuts for area. A similar concept, *area flow*, was later proposed by Manohararajah et al. [55]. Intuition regarding effective area is to distribute the area for a multi-fanout node to its fanout nodes so that logic sharing and reconvergence can be considered during area cost propagation. Effective areas are computed in a topological order from PIs to POs. The effective area $a(v)$ of a PI node v is set to zero. Equation 13.2 is used to compute the effective area of a cut:

$$a(c) = \left(\Sigma_{u \in c} \left[a(u)/|output(u)|\right]\right) + A_c \tag{13.2}$$

where A_c is the area of the LUT corresponding to the cut c. The area cost of a non-PI node can then be set to the minimum effective area of its cuts: $a(v) = \min\{a(c)|\forall u \in \Phi(v)\}$.

It should be pointed out that effective area may not account for the situation where the node may be duplicated in a mapping solution. In the example shown in Figure 13.2, with $K = 3$, the LUT for w is introduced solely for the LUT for v. However, in effective area computation, only one-half is counted for v, and as a result the LUT for w is undercounted. In this example, the sum of effective area of the POs is 2.5 whereas the mapping solution has three LUTs. In general, effective area is a lower bound of the actual area.

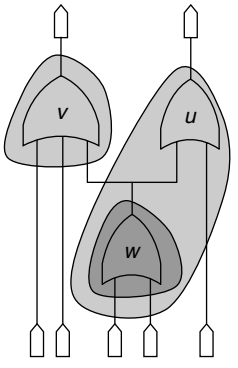

FIGURE 13.2 ■ Inaccuracy in effective area.

The PRAETOR algorithm [29] is an area-oriented mapping algorithm that ranks cuts using effective area. It further improves the basic mapping framework with a number of area reduction techniques. One such technique is to encourage the use of common subcuts. A cut for a fanout of a node v induces a cut for v (perhaps the trivial cut consisting of v itself). If two fanouts of v induce different cuts for v, the most likely result will be an area increase due to the need to duplicate v and possibly some of its predecessor nodes. To alleviate this problem, PRAETOR sorts and selects cuts with the same effective area in a predetermined order to avoid arbitrary selection. It assigns an integer ID to each node and then sorts all cuts with the same effective area according to the lexicographic order based on the IDs of the nodes in the cuts. The first cut with minimum effective area for each node is selected.

Another area reduction technique introduced in PRAETOR is to carry out cut selection twice. The nodes with LUTs selected in the first pass are declared nonduplicable and can only be covered by LUTs for themselves in the second pass. This encourages selection of cuts with less duplication. As an example, suppose that in the first pass of cut selection, the mapping solution shown in Figure 13.3(a), with four LUTs, is selected. In the second pass, the LUT containing v and u_1 is excluded from consideration for u_1. This exclusion will also encourage the selection of the cut that results in the LUT containing a for u_1. As a result, the mapping algorithm generates, in the second pass, the mapping solution in Figure 13.3(b), with only three LUTs. Experimental results show that PRAETOR can significantly improve area over previous algorithms.

The IMap algorithm proposed by Manohararajah et al. [55] is another mapping algorithm targeting area optimization. It introduced two enhancements: (1) iteration between cut ranking and cut selection multiple times, and (2) adjustment of the area costs between successive iterations using history information. In the effective area formula (equation 13.2), the fanout count of u in the initial network, $|output(u)|$, is used to estimate the fanout count of the LUT rooted at u in the mapping solution. In the IMap algorithm between iterations, the fanout

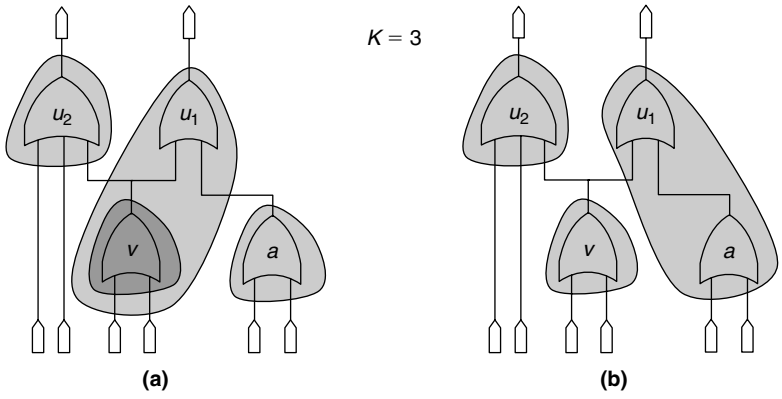

FIGURE 13.3 ■ Effect of excluding cuts across nonduplicable nodes: (a) initial mapping solution, and (b) improved solution with better area.

count estimation is updated by using a weighted combination of the estimated and the real fanout counts in previous iterations. As a result, equation 13.2 becomes $a(c) = \left(\Sigma_{u \in c} \left[a(u)/estimated\_fc(u)\right]\right) + A_c$, where $estimated\_fc(u)$ denotes the estimated fanout count for the current iteration.

Ling et al. [54] proposed a mixed structural and functional area-mapping algorithm that starts with a mapping solution (e.g., generated by a structural mapping algorithm). The key idea is a Boolean satisfiability (SAT) formulation for the problem of mapping a small circuit with up to ten inputs into the smallest possible number of LUTs. The algorithm iteratively selects a small logic cone to remap to fewer LUTs using an SAT solver. It is shown that for some highly structured (albeit small) designs, area can be improved significantly.

Most area optimization techniques are heuristic. A natural question is how close the mapping solutions obtained using existing mapping algorithms are from optimal. Cong and Minkovich [24] constructed a set of designs with known optimal area-mapping solutions, called LEKO (logic synthesis examples with known optimal bounds) examples, and tried existing academic algorithms and commercial tools on them. The average gap from optimal varied from 5 to 23 percent. From LEKO examples, they further derived LEKU (logic synthesis examples with known upper bounds) examples that require both logic optimization and mapping. Existing algorithms perform poorly on LEKU examples, with an average optimality gap of more than 70 times. This indicates that more research is needed in area-oriented mapping and optimization.

13.1.3 Performance-driven Mapping

The FlowMap algorithm and its derivatives can find a mapping solution with optimal depth. Recent advances in delay mapping focus on achieving the best performance with minimal area.

Exact layout information is not available during technology mapping in a typical FPGA design flow. Mapping algorithms usually ignore routing delays and try to optimize the total cell delays on the longest combinational paths in the mapping solution.

Most delay optimal mapping algorithms use the labeling scheme introduced in the FlowMap algorithm to rank and select cuts. The label of a PI is set to zero, assuming that the signal arrives at the beginning of the clock edge. After the labels for all the nodes in the fanin cone of a node v are found, the label of a cut c of v is determined using the formula in equation 13.3:

$$l(c) = \max\{l(u) + D_c | \forall u \in c\} \tag{13.3}$$

where D_c is the delay of the LUT corresponding to c. Intuitively, $l(c)$ is the best arrival time at v if it is covered using the LUT generated from c. The label of v is then the smallest label among all of its cuts: $l(v) = \min\{l(c) | \forall c \in \Phi(v)\}$.

DAOmap [9] is a mapping algorithm that guarantees optimal delay while at the same time minimizing the area. It introduces three key techniques to optimize area without degrading timing. First, it enhances effective area computation to make it better avoid node duplication. Second, it applies area optimization

techniques on noncritical paths. Last, it uses an iterative cut selection procedure to explore and perturb the solution space to improve solution quality.

DAOmap first picks cuts with the minimum label for each node. From those, it then picks one with minimum effective area. Furthermore, when there is positive slack, which is the difference between required time and arrival time at a node, it picks a cut with as small an area cost as possible under the condition that the timing increase does not exceed the slack.

Recognizing the heuristic nature of effective area computation, DAOmap also employs the technique of multiple passes of cut selection. Moreover, it adjusts area costs based on input sharing to encourage using nodes that have already been contained in selected cuts. This reduces the chance that a newly picked cut cuts into the interior of existing LUTs. Between successive iterations of cut selection, DAOmap also adjusts area cost to encourage selecting cuts containing nodes with a large number of fanouts in previous iterations. There are a few other secondary techniques used in DAOmap. The interested reader is referred to Chen and Cong [9] for details.

Based on the results reported, DAOmap can improve the area by about 13 percent on a large set of academic and industrial designs while maintaining optimal depths. It is also many times faster than previous mapping algorithms based on max-flow computation, mainly because of efficient implementation of cut enumeration.

A recent delay optimal mapping algorithm introduced several techniques to improve area while preserving performance [57]. Like DAOmap, this algorithm goes through several passes of cut selection, with each pass selecting cuts with better areas among the cuts that do not degrade timing. It is also based on the concept of effective area (or area flow). However, it does cut selection from PIs to POs instead of from POs to PIs, as in most other algorithms. With this processing order, the algorithm tries to use timing slacks on nodes close to PIs to reduce area cost. This is based on the observation that logic is typically denser when close to PIs, so slack relaxation is more effective for nodes closer to PIs. Experimental data shows 7 percent better area over DAOmap for the same optimal depths.

13.1.4 Power-aware Mapping

Power has become a major concern for FPGAs [51, 68]. Dynamic power dissipation in FPGAs results from charging and discharging capacitances. It is determined by the switching activities and the load capacitance of the LUT outputs and can be captured by equation 13.4:

$$P = \frac{1}{2}\Sigma_v C_v \cdot f_v \cdot V^2 \tag{13.4}$$

where C_v is the output load capacitance of node v, f_v is the switching activity of node v, and V is the supply voltage. Given a fixed supply voltage, power consumption in a mapped netlist is determined by switching activities and load capacitance of the LUT outputs.

Because technology mapping for power is NP-hard [34], a number of heuristic algorithms have been proposed. Most power-aware mapping algorithms try to reduce switching activities by hiding nodes with high switching activities inside LUTs, hence leaving LUTs with small output-switching activities in the mapped netlist.

Anderson and Najm [2] proposed a mapping algorithm to reduce switching activities and minimize logic duplication. Logic duplication is necessary to optimize timing and area, but can potentially increase power consumption. The algorithm uses the following power-aware cost function to rank cuts: $Cost(c) = l(c) + \beta \cdot P(c) + \gamma \cdot R(c)$, where $l(c)$ is the depth label of the cut c as given in equation 13.3 and $P(c)$ and $R(c)$ are the power and replication costs of the cut, respectively. The weighting factors β and γ can be used to bias the three cost terms. Anderson and Najm suggest a very small β to get a depth-optimal mapping solution with minimal power.

Power cost $P(c)$ is defined in such a way that it encourages absorbing high-activity connections inside LUTs. The replication cost tries to discourage logic duplication on timing noncritical paths. Power savings of over 14 percent were reported over timing-oriented mapping algorithms when both targeted optimal depths. When the mapping depth was relaxed by one level over optimal, additional power reduction of about 8 percent for 4-LUTs and 10 percent for 5-LUTs was reported.

One serious limitation of the power-based ranking in Anderson and Najm [2] is that it cannot account for multiple fanouts and reconvergence, which are common in most practical designs. Chen et al. [8] proposed a low-power technology-mapping algorithm based on an improved power-aware ranking in equation 13.5:

$$P(c) = (\Sigma_{u \in c}[P(u)/|output(u)|]) + U_c \tag{13.5}$$

where U_c is a cost function that tries to capture power contributed by the cut c itself. Experimental results show that this algorithm outperforms previous power-aware mapping algorithms. It has also been extended to handle dual supply voltage FPGA architectures.

13.2 INTEGRATED MAPPING ALGORITHMS

Technology mapping is a step in the middle of an FPGA design flow. Technology-independent optimization is carried out before mapping; placement is carried out after. Sequential optimization such as retiming can be carried out before or after mapping. A separate approach can miss the best overall solutions even if we can solve each individual step optimally. In the section that follows we discuss mapping algorithms that combine mapping with other steps in the design flow.

13.2.1 Simultaneous Logic Synthesis, Mapping

Technology-independent Boolean optimizations carried out prior to technology mapping can significantly impact the mapping solution. During technology-independent optimization, we have the freedom to change the network structures,

but accurate estimation of their impact on mapping is not available. During technology mapping, we can achieve optimal or close to optimal solutions using the algorithms discussed in Section 13.1. However, we are stuck with a fixed network. It is desirable to capture the interactions between logic optimization and mapping to arrive at a solution with better quality.

Lossless synthesis has been proposed by Mishchenko et al. [57] as a way to consider technology-independent optimization during mapping. It is based on the concept of *choice networks*, which is similar to the concept of mapping graphs [11, 49]. A choice network contains choice nodes that encode functionally equivalent but structurally different alternatives. The algorithm operates on a simple yet powerful data structure called *AIG*, which is a network of AND2 and INV gates. A combination of SAT and simulation techniques is used to detect functionally equivalent points in different networks and compress them to form one choice network.

Figure 13.4 illustrates the construction of a network with choices from two equivalent networks with different structures. The nodes x_1 and x_2 in the two networks are functionally equivalent. They are combined in an equivalence class in the choice network, and an arbitrary member (x_1 in this case) is set as the class representative. Note that p does not lead to a choice because its implementation is structurally the same in both networks. Similarly, o does not lead to a choice node.

Rather than try to come up with one "good" optimized network before mapping, the algorithm proposed by Mishchenko et al. [57] accumulates choices by combining intermediate networks seen during logic synthesis to generate a network with many choices. In a sense, it does not make judgments on the goodness of the intermediate networks but defers that decision to the mapping phase, when the best combination of these choices is selected. In the final mapping solution, different sections may come from different intermediate networks. For example, the timing-critical sections of the final mapping solution may come

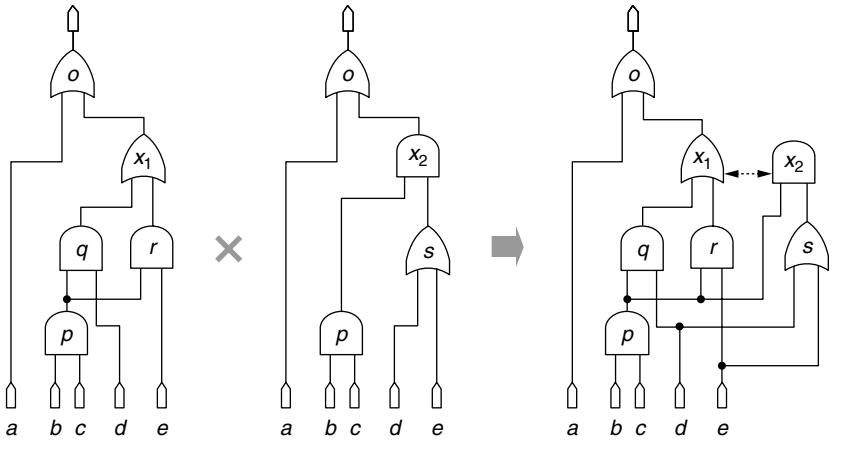

FIGURE 13.4 ■ Combining networks to create a choice network.

from networks optimized for timing, while the timing noncritical sections of the final mapping solution may come from networks optimized for area.

For mapping on choice networks, cut generation and cut ranking are extended to choice nodes. For example, the set of cuts of a choice node is simply the union of the sets of cuts of all of that node's fanin nodes. Similarly, the label of a choice node is the smallest one among the labels of its fanin nodes. The rest of the approach is similar to a conventional mapping algorithm. Results reported by Mishchenko et al. [57] show that both timing and area can be improved by over 7 percent on a set of benchmark designs compared to applying mapping to just one "optimized" network.

13.2.2 Integrated Retiming, Mapping

Retiming (discussed in Chapter 18) is an optimization technique that relocates flip-flops (FFs) in a network while preserving functionality of the network [50]. Retiming can shift FF boundaries and change the timing. If retiming is applied after mapping, mapping may optimize the wrong paths because the critical paths seen during mapping may not be critical after the FFs are repositioned. On the other hand, if retiming is applied before mapping, it will be carried out using less accurate timing information because it is applied to an unmapped network. In either approach, the impact of retiming on cut generation cannot be accounted for.

The network in Figure 13.5(a) is derived from the design in Figure 13.1(a) by retiming the FFs at the outputs of y and i to their inputs. After the retiming, all gates can be covered with one 3-LUT, as indicated in (a). The corresponding mapping solution is shown in (b). This mapping solution is obviously better than the one in Figure 13.1(c) in both area and timing.

Pan and Liu [63] proposed a polynomial time-mapping algorithm that can find a solution with the best cycle time in the combined solution space of

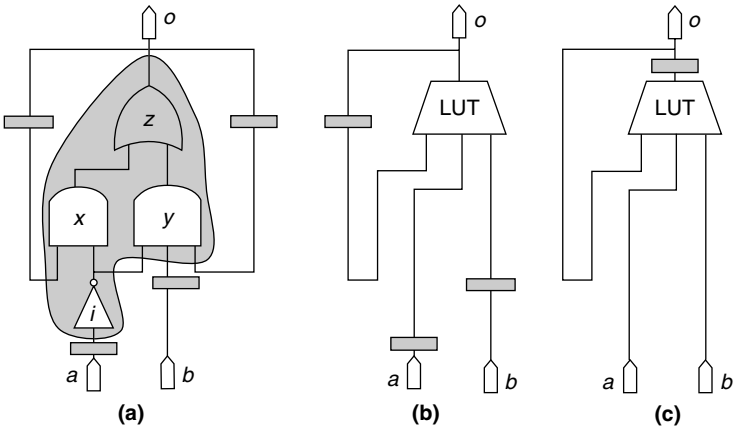

FIGURE 13.5 ■ Retiming, mapping: (a) retiming and covering, (b) mapping solution, and (c) retimed solution.

retiming and mapping. In other words, the solution obtained is the best among all possible ways of retiming and mapping a network. Improved algorithms were later proposed that significantly reduce runtime while preserving the optimality of the final mapping solution [25, 27]. These algorithms, like the FlowMap algorithm, are all based on max-flow computation.

A cut enumeration-based algorithm for integrated retiming and mapping was proposed by Pan and Lin [61]. In it, cut generation is extended to go across FF boundaries to generate sequential cuts. In a network with FFs, a gate may go through zero or more FFs in addition to logic gates before reaching gate v. To capture this information, an element in a cut for a node v is represented as a pair consisting of the driving node u and the number of FFs d on the paths from u to v, denoted by u^d. Note that one node may reach another node through paths with different FF counts. In that case, the node will appear in the cut multiple times with different values of d. For example, for the cone in Figure 13.5(a), the corresponding cut is $\{z^1, a^1, b^1\}$. Pan and Lin [61] suggested an iterative procedure to determine the sequential cuts for all nodes.

To consider retiming effect, the concept of labels is extended using sequential arrival times [62, 63]. The label of a cut c is now defined as follows:

$$l(c) = \max\{l(u) - d \cdot \phi + D_c | \forall u^d \in c\} \tag{13.6}$$

where ϕ is the target cycle time and D_c is the delay of the LUT corresponding to c. The combinationl cut formula (equation 13.3) can be viewed as a special case of equation 13.6 when $d = 0$. As in combinational mapping algorithms, the label of a gate v is the minimum of the labels of its cuts: $l(v) = \min\{l(c) | \forall c \in \Phi(v)\}$. The label of each PI is zero, and the label for each PO is that of its driver.

Pan and Lin's algorithm finds the labels for cuts and nodes through successive approximation by going through the nodes in the initial network in passes. After the labels for all nodes are computed and the target cycle time is determined to be achievable, the next step is to generate a mapping solution. As in the combinational case, a mapped network is constructed starting from POs and going backward. At each node v, the algorithm selects one of the cuts that realize the node's label and then moves on to select a cut for u if u^d is in the cut selected for v. On the interconnection from u to v, d FFs are inserted. To obtain the final mapping solution with a cycle time of ϕ, the algorithm retimes the LUT for each non-PI/PO node v by $\lceil l(v)/\phi \rceil - 1$. For the initial network in Figure 13.1(a), the final mapping solution with optimal cycle time generated by the algorithm is shown in Figure 13.5(c). Experimental results show that the algorithm is very efficient and consistently produces mapping solutions with better performance than combinational depth optimal mapping followed by optimal retiming.

13.2.3 Placement-driven Mapping

One drawback of the conventional mapping flow is the lack of accurate timing information on interconnects. Most algorithms use logic depth to measure timing. However, optimal-depth mapping solutions may not always be good

after placement. To overcome this problem, we need to combine mapping with placement so that mapping can see more accurate interconnect information.

A number of algorithms try to carry out placement and mapping simultaneously [3, 6, 53, 59, 69]. For example, the MIS-pga algorithm of Murgai et al. [59] performs iterative logic optimization and placement. Chen et al. [6] proposed an algorithm that tightly couples technology mapping and placement by mapping each cell and placing it at the same time. In practice, such integrated approaches suffer a serious limitation: Because of the complexity of the combined problem, simple mapping, placement techniques are employed. As a result, the benefit of the combined approach is diminished.

Another approach is to iterate between mapping and placement (or placement refinement). Here, the design is first mapped and placed. Then the netlist is back-annotated and remapped under the given placement. This process can be repeated until a satisfactory solution is found. Figure 13.6 outlines the major steps in the iterative mapping and placement algorithm proposed by Lin et al. [53]. The key step is placement-driven remapping. The remapping step may make the placement illegal—for example, it may place more than one cell at the same location. If this happens, the placement needs to be legalized and refined.

Lin et al.'s algorithm [53] uses table lookup to estimate interconnect delays based on placement locations. Given two locations, it looks up the estimated delay in a prestored table for the wiring between the two locations. This is more accurate and realistic than the "fixed" interconnect delays used in earlier layout-based mapping algorithms [56, 72].

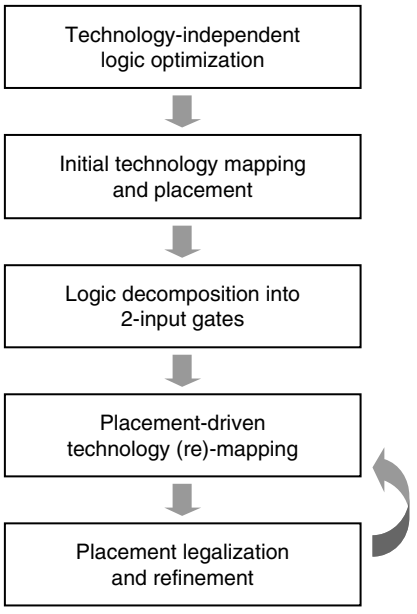

FIGURE 13.6 ■ Iterative mapping, placement.

One difficulty in placement-driven mapping is that the placement may not become legal because of cell overlaps. Another is that timing predicted in the labeling phase may be unrealizable because of congestion in the new mapping solution. Congestion means that many LUTs are assigned to a small region, which requires many cell relocations to legalize the placement, which in turn, perturbs the placement and eventually the timing. To overcome this problem, the algorithm employs an iterative process with multiple passes of cut selection. Each pass uses the cell congestion information gathered during previous iterations to guide the mapping decisions. Several techniques have been proposed to relieve congestion. One is a hierarchical area control scheme to evaluate the local congestion cost, in which the chip is divided into bins with different granularities. Area increase is tallied in bins, and penalty costs are given to bins with area overflows.

Once a mapping solution is generated, the algorithm invokes timing-driven legalization that moves overlapping cells to empty locations in their neighborhood based on the timing slack available to the cells. Finally, a simulated annealing-based placement refinement phase is carried out to improve performance. Experimental results show that the algorithm can improve timing by more than 12 percent, with minimal area penalty due to remapping.

13.3 MAPPING ALGORITHMS FOR HETEROGENEOUS RESOURCES

Up to this point, we have assumed that all logic cells are LUTs with a uniform input size K. In reality, commercial FPGA architectures contain heterogeneous resources (e.g., LUTs of different input sizes, embedded memory, and PLA-like logic cells). We briefly summarize mapping algorithms that target or take advantage of such architectural features.

13.3.1 Mapping to LUTs of Different Input Sizes

There are a number of commercial FPGA architectures that support LUTs with multiple input sizes on the same device. Mapping algorithms have been proposed to optimize area [29, 39, 40, 43] and timing [30, 32].

In the special case of tree networks, Korupolu et al. [43] presented a polynomial area optimal algorithm. For general networks, the PRAETOR algorithm discussed in Section 13.1.2 can be applied to these architectures by assigning different area costs for LUTs with different input sizes.

For timing optimization, the algorithm proposed by Cong and Xu [30] is an extension of FlowMap. Like FlowMap, it is also based on flow computation and can be cast in the cut enumeration framework. Assume that there are two types of LUTs with input sizes K_1 and K_2, and delays d_1 and d_2, where $K_1 < K_2$, $d_1 < d_2$. We can enumerate all K_2-cuts. When labeling a cut, we can set its delay to d_1 or d_2 depending on its size. With this simple modification, an algorithm for homogeneous LUT architectures can be used for architectures with different LUT sizes.

When there are resource bounds on available LUTs of different sizes, the mapping problem becomes NP-hard. Assuming that there can be at most r K_2-LUTs, a heuristic algorithm was proposed that starts out by finding a mapping solution without considering resource bounds [31]. If the current mapping solution meets the resource bound, it stops. If not, it increases d_2, the delay of K_2-LUTs, and solves the unconstrained version again, which should lead to another mapping solution with a decreased number of K_2-LUTs. This process is repeated until the resource bound is met.

13.3.2 Mapping to Complex Logic Blocks

FPGA devices typically contain additional logic that, together with LUTs, can form complex *programmable logic blocks* (PLBs). PLBs can implement complex logic functions. Figure 13.7 shows two PLBs that consist of LUTs and logic gates and can implement functions of up to nine inputs.

A simple approach to PLB mapping is to map the initial network to the constituent cells inside the PLBs. For example, for a device with the PLB in Figure 13.7(a), we can first map the initial network to 3-LUTs and 4-LUTs. Afterwards, the LUTs are clustered to obtain a network of PLBs. Such a two-step approach is obviously suboptimal.

Recent approaches try to map directly to PLBs [13, 23, 47, 65]. The cut enumeration framework can still be used after enhancements. Because a PLB can have more inputs than a typical LUT, a node may have too many cuts. Intelligent cut pruning, using techniques such as those proposed by Chatterjee et al. [5] and Ling et al. [54], is necessary to avoid long runtime and memory explosion. Unlike in the case of LUTs, a PLB has limited functional capability in that it cannot implement all of the functions of its inputs. For example, the PLB in Figure 13.7(b) can implement all functions of up to five inputs, but it can only implement some of the functions with six inputs. An essential step in PLB mapping is Boolean matching, which, given a cut, decides if the corresponding logic cone can be implemented by a PLB.

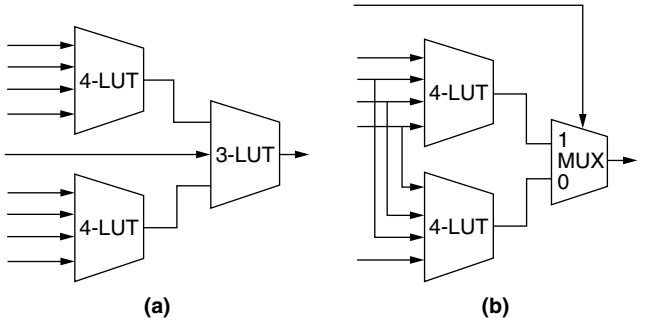

FIGURE 13.7 ■ Two PLB examples.

Algorithms for Boolean matching for PLBs can be classified into two categories: decomposition based [13, 23] and satisfiability (SAT) based [25, 54, 65]. Decomposition-based Boolean matching tries to decompose the input function according to the structure of the target PLB using functional decomposition. Cong and Hwang [23] proposed matching procedures for a wide variety of common PLBs.

A drawback of decomposition-based Boolean matching is that each PLB needs a specialized matching procedure. Decomposition-based Boolean matching can also be slow and memory intensive because of extensive use of BDD operations. On the other hand, SAT-based Boolean matching encodes the function, the target PLB, and their matching in a Boolean expression in conjunctive normal form (CNF). Then it leverages an efficient SAT solver (e.g., the one proposed by Moskewicz et al. [58]) to check whether the PLB can be configured to implement the function. The size of the CNF expression can have significant impact on the runtime of an SAT-based matching algorithm. An improved SAT formulation with smaller expressions was proposed recently by Cong and Minkovich [25].

13.3.3 Mapping Logic to Embedded Memory Blocks

On-chip memory has become a common feature of high-performance FPGAs. Dedicated embedded memory blocks (EMBs) can be used to improve clock frequencies and lower costs for large designs that require memory. If a design does not need all the available EMBs, unused ones can be employed to implement logic, which essentially turns them into large multi-input multi-output LUTs.

EMBs usually have configurable widths and depths, so they can be used to implement functions with different numbers of inputs/outputs. For example, a 2K-bit memory with configurations 2048×1, 1024×2, and 512×4 can be used to implement an 11-input/1-output, 10-input/2-output, or 9-input/4-output logic function, respectively.

Mapping logic to EMBs is typically done as a postprocessing step after LUT mapping. These algorithms start with an optimized LUT-mapping solution and then pack groups of LUTs into unused EMBs [26, 70]. The SMAP algorithm [70] maps one EMB at a time. It begins by selecting a seed node. A fanin cone of the seed node is generated by finding a d-feasible cut that covers as many nodes as possible, where d is the bit width of the address line of the target EMB. Because d is considerably large, flow-based cut generation is used. After the cone is generated, the output selection process selects signals to be the EMB outputs. Output selection tries to select a set of signals so that the resulting EMB can eliminate as many LUTs as possible. This is done by assigning each node a score that reflects the number of eliminated nodes if the node is selected. The w highest-scoring nodes are selected as the EMB outputs, where w is the number of outputs of the target EMB.

The selection of the seed node is critical for this method. The algorithm tests each candidate node and selects the one that leads to the maximum number of

eliminated LUTs. Heuristics were introduced to consider EMBs with different configurations and to preserve timing.

Another algorithm, EMB_Pack, proposed by Cong and Xu [26], takes a slightly different approach. It finds the logic to map to EMBs altogether instead of one at a time, as in SMAP, which can potentially find better mapping.

13.3.4 Mapping to Macrocells

Complex programmable logic devices (CPLDs) are a class of programmable logic devices that are more coarse grained than typical FPGAs. Each CPLD logic cell (called Pterm block) is essentially a programmable logic array (PLA) that consists of a set of product terms (Pterms) with multiple outputs. A Pterm block can be characterized by a 3-tuple (k,m,p) where k is the number of inputs, p is the number of outputs, and m is the number of Pterms for the block. The input size k is typically much larger than that of FPGA logic cells.

Relatively speaking, there is much less mapping work reported for CPLDs. A fast heuristic partition method for PLA-based structures was presented by Hasan et al. [38]. The DDMap algorithm [42] adapts a LUT mapper for CPLD mapping. It uses wide cuts to form big LUTs and decomposes the big LUTs into Pterms allowed in the target CPLD. Packing is used to form multi-output Pterm cells. An area-oriented mapping algorithm was proposed for CPLDs by Anderson and Brown [1]. Cong et al. [20] investigated an FPGA architecture consisting of single-output Pterm blocks, and proposed a timing-oriented mapping algorithm.

PLAmap is a timing-oriented mapping algorithm for CPLDs [7]. Like the LUT mapping algorithms discussed earlier, it has a labeling phase and a mapping phase. In the labeling phase, it tries to find the minimal mapping depth for each node using a logic cell $(k,m,1)$—that is, a single-output Pterm block, assuming that each logic cell has one unit delay. The labeling procedure is based on Lawler et al.'s clustering algorithm [46]. Let l be the largest label of the nodes in the fanin cone of a node. The algorithm forms a cluster for the node by grouping it with all nodes in its fanin cone with the label l. If the cluster can be implemented by a $(k,m,1)$ cell, the node is assigned the label l; otherwise, the node gets the label $l+1$ with a cluster consisting of the node itself. Note that this is a heuristic in that the label may not be the best because of the so-called *nonmonotone property* [7]. The mapping phase is done in reverse topological order from the POs. The algorithm tries to merge the clusters generated in the labeling phase to form (k, m, p) cells whenever possible. Cluster merging is done in such a way that duplication is minimized and the labels of the POs do not exceed the performance target. Experimental results show that PLAmap outperforms commercial tools and other algorithms with no (or a very small) area penalty.

Pterm blocks or macrocells are suitable for implementing wide-fanin, low-density logic, such as finite-state machines. They can potentially complement fine-grained LUTs to improve both performance and utilization. Device architectures with a mixture of LUTs and Pterm blocks or macrocells have been suggested to take advantage of different types of logic cells. Technology mapping algorithms have been proposed for such hybrid architectures [41, 42, 44].

13.4 SUMMARY

This chapter discussed technology mapping algorithms for FPGAs. Emphasis was placed on state-of-the-art algorithms that have been, or most likely will be, reduced to practice. We discussed mapping algorithms for different objectives, such as area, timing, and power, as well as mapping algorithms that take advantage of heterogeneous resources in modern FPGA devices.

FPGA technology mapping has been and continues to be a subject of active research. A general trend is to integrate technology mapping with other steps in the FPGA design flow to improve the quality of final implementations (e.g., combining mapping and clustering [10]).

As semiconductor technologies advance, new FPGA architecture features are being introduced to improve area utilization, performance, and power consumption. For example, architectures have been introduced or proposed that use large LUTs (much larger than traditional 4-/5-LUTs) or multiple supply voltages. New mapping techniques are being developed to take advantage of these architecture features.

References

[1] J. H. Anderson, S. D. Brown. Technology mapping for large complex PLDs. *ACM/IEEE Design Automation Conference*, 1998.

[2] J. H. Anderson, F. N. Najm. Power-aware technology mapping for LUT-based FPGAs. *IEEE International Conference on Field-Programmable Technology*, 2002.

[3] N. Bhat, D. D. Hill. Routable technology mapping for LUT FPGAs. *IEEE International Conference on Computer Design*, 1992.

[4] S. C. Chang, M. Marek-Sodowska, T. Hwang. Technology mapping for LUT FPGA based on decomposition of binary decision diagrams. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 15(10), October 1996.

[5] S. Chatterjee, A. Mishchenko, R. Brayton. Factor cuts. *International Conference on Computer-Aided Design,* 2006

[6] C. Chen, Y. Tsay, Y. Hwang, T. Wu, Y. Lin. Combining technology mapping, placement for delay-optimization in FPGA designs. *International Conference on Computer-Aided Design*, 1993.

[7] D. Chen, J. Cong, M. Ercegovac, Z. Huang. Performance-driven mapping for CPLD architectures. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 22(10), October 2003.

[8] D. Chen, J. Cong, F. Li, L. He. Low-power technology mapping for FPGA architectures with dual supply voltages. *International Symposium on Field-Programmable Gate Arrays*, February 2004.

[9] D. Chen., J. Cong. DAOmap: A depth-optimal area optimization mapping algorithm for FPGA designs. *International Conference on Computer-Aided Design*, 2004.

[10] D. Chen, J. Cong, J. Lin. Optimal simultaneous mapping, clustering for FPGA delay optimization. *ACM/IEEE Design Automation Conference*, 2006.

[11] G. Chen, J. Cong. Simultaneous logic decomposition with technology mapping in FPGA designs. *International Symposium on Field-Programmable Gate Arrays,* 2001.

[12] K. C. Chen, J. Cong, Y. Ding, A. B. Kahng, P. Trajmar. DAGmap: Graph-based FPGA technology mapping for delay optimization. *IEEE Design and Test of Computers* 9(3), September 1992.

[13] M. Chikodikar, S. Laddha, A. Sirasao. A technology mapper for Xilinx FPGAs. *Tenth International Conference on VLSI Design*, January 1997.

[14] J. Cong, Y. Ding. An optimal technology-mapping algorithm for delay optimization in lookup table–based FPGA designs. *International Conference on Computer-Aided Design*, November 1992.

[15] J. Cong, Y. Ding. Beyond the combinatorial limit in depth minimization for LUT-based FPGA designs. *International Conference on Computer-Aided Design*, 1993.

[16] J. Cong, Y. Ding. FlowMap: An Optimal technology-mapping algorithm for delay optimization in lookup table–based FPGA designs. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 13(1), January 1994.

[17] J. Cong, Y. Ding. On area/depth trade-off in LUT-based FPGA technology mapping. *IEEE Transactions on VLSI Systems* 2(2), 1994.

[18] J. Cong, Y. Ding. Combinational logic synthesis for LUT-based field-programmable gate arrays. *ACM Transactions on Design Automation of Electronic Systems* 1(2), April 1996.

[19] J. Cong, Y. Ding. T. Gao, K. C. Chen. LUT-base, FPGA technology mapping under arbitrary net-delay model. *Computers and Graphics* 18(4), 1994.

[20] J. Cong, H. Huang, X. Yuan. Technology mapping and architecture evaluation for k/m-macrocell-based FPGAs. *ACM Transactions on Design Automation of Electronic Systems*, January 2005.

[21] J. Cong, Y. Hwang. Simultaneous depth and area minimization in LUT-based FPGA mapping. *International Symposium on Field-Programmable Gate Arrays*, February 1995.

[22] J. Cong, Y. Hwang. Structural gate decomposition for depth-optimal technology mapping in LUT-based FPGA design. *Design Automation Conference*, 1996.

[23] J. Cong, Y. Hwang. Boolean matching for LUT-based logic blocks with applications to architecture evaluation and technology mapping. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 20(9), 2001.

[24] J. Cong, K. Minkovich. Optimality study of logic synthesis for LUT-based FPGAs. *International Symposium on Field-Programmable Gate Arrays*, February 2006.

[25] J. Cong, K. Minkovich. Improved SAT-based Boolean matching using implicants for LUT-based FPGAs. *International Symposium on Field-Programmable Gate Arrays*, February 2007.

[26] J. Cong, S. Xu. Technology mapping for FPGAs with embedded memory blocks. *International Symposium on Field-Programmable Gate Arrays*, 1998.

[27] J. Cong, C Wu. FPGA Synthesis with retiming and pipelining for clock period minimization of sequential circuits, *Design Automation Conference,* 1997.

[28] J. Cong, C Wu. Optimal FPGA mapping, retiming with efficient initial state computation. *Design Automation Conference,* 1998.

[29] J. Cong, C. Wu, Y. Ding. Cut ranking and pruning: Enabling a general, efficient FPGA mapping solution. *International Symposium on Field-Programmable Gate Arrays*, February 1999.

[30] J. Cong, S. Xu. Delay-optimal technology mapping for FPGAs with heterogeneous LUTs. *Design Automation Conference*, 1998.

[31] J. Cong, S. Xu. Delay-oriented technology mapping for heterogeneous FPGAs with bounded resources. *International Conference on Computer-Aided Design*, 1998.

[32] J. Cong, S. Xu. Performance-driven technology mapping for heterogeneous FPGAs. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 19(11), November 2000.
[33] G. De Micheli. *Synthesis: Optimization of Digital Circuits*, McGraw-Hill, 1994.
[34] A. H. Farrahi, M. Sarrafzadeh. FPGA technology mapping for power minimization. *International Workshop on Field-Programmable Logic and Applications*, 1994.
[35] A. Farrahi, M. Sarrafzadeh. Complexity of the lookup-table minimization problem for FPGA technology mapping. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 13(11), November 1994.
[36] R. J. Francis et al. Chortle-CRF: Fast technology mapping for lookup table–based FPGAs. *Design Automation Conference*, 1991.
[37] R. J. Francis, J. Rose, Z. Vranesic. Technology mapping for lookup table–based FPGAs for performance. *International Conference on Computer-Aided Design*, November 1991.
[38] Z. Hasan, D. Harrison, M. Ciesielski. A fast partition method for PLA-based FPGAs. *IEEE Design and Test of Computers*, December 1992.
[39] J. He, J. Rose. Technology mapping for heterogeneous FPGAs. *International Symposium on Field-Programmable Gate Arrays,* 1994.
[40] M. Inuani, J. Saul. Resynthesis in technology mapping for heterogeneous FPGAs. *International Conference on Computer-Aided Design*, 1998.
[41] A. Kaviani, S. Brown. Technology-mapping issues for an FPGA with lookup tables, PLA-like blocks. *International Symposium on Field-Programmable Gate Arrays,* 2000.
[42] J. L. Kouloheris. *Empirical Study of the Effect of Cell Granularity on FPGA Density, Performance*, Ph.D. thesis, Stanford University, 1993.
[43] M. R. Korupolu, K. K. Lee, D. F. Wong. Exact tree-based FPGA technology mapping for logic blocks with independent LUTs. *Design Automation Conference,* 1998.
[44] S. Krishnamoorthy, R. Tessier. Technology-mapping algorithms for Hybrid FPGAs containing lookup tables, PLAs. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 22(5), May 2003.
[45] J. Lamoureux, S. J. E. Wilton. On the interaction between power-aware FPGA CAD algorithms. *IEEE International Conference on Computer-Aided Design*, November 2003.
[46] E. L. Lawler, K. N. Levitt, J. Turner. Module clustering to minimize delay in digital networks. *Transactions on Computers* 18(1), 1969.
[47] K. Lee, D. Wong. An exact tree-based, structural technology-mapping algorithm for configurable logic blocks in FPGAs. *International Conference on Computer-Aided Design*, 1999.
[48] C. Legl, B. Wurth, K. Eckl. A Boolean approach to performance-directed technology mapping for LUT-based FPGA designs. *Design Automation Conference*, June 1996.
[49] E. Lehman, Y. Watanabe, J. Grodstein, H. Harkness. Logic decomposition during technology mapping. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 16(8), 1997.
[50] C. E. Leiserson, J. B. Saxe. Retiming synchronous circuitry. *Algorithmica* 6, 1991.
[51] F. Li, D. Chen, L. He, J. Cong. Architecture evaluation for power-efficient FPGAs. *International Symposium on Field-Programmable Gate Arrays*, February 2003.
[52] H. Li, S. Katkoori, W. K. Mak. Power minimization algorithms for LUT-based FPGA technology mapping. *ACM Transactions on Design Automation of Electronic Systems* 9(1), January 2004.

[53] J. Lin, A. Jagannathan, J. Cong. Placement-driven technology mapping for LUT-based FPGAs. *International Symposium on Field-Programmable Gate Arrays*, February 2003.
[54] A. Ling, D. Singh, S. Brown. FPGA technology mapping: A study of optimality. *Design Automation Conference*, 2005.
[55] V. Manohararajah, S. D. Brown, Z. G. Vranesic. Heuristics for area minimization in LUT-based FPGA technology mapping. *International Workshop on Logic Synthesis*, 2004.
[56] A. Mathur, C. L. Liu. Performance-driven technology mapping for lookup table–based FPGAs using the general delay model. *International Workshop on Field-Programmable Gate Arrays*, February 1994.
[57] A. Mishchenko, S. Chatterjee, R. Brayton. Improvements to technology mapping for LUT-based FPGAs. *International Symposium on Field-Programmable Gate Arrays,* 2006.
[58] M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, S. Malik. Chaff: Engineering an efficient SAT solver. *Design Automation Conference*, 2001.
[59] R. Murgai et al. Improved logic synthesis algorithms for table lookup architectures. *International Conference on Computer-Aided Design*, November 1991.
[60] R. Murgai et al. Performance directed synthesis for table lookup programmable gate arrays. *International Conference on Computer-Aided Design*, November 1991.
[61] P. Pan, C. C. Lin. A new retiming-based technology-mapping algorithm for LUT-based FPGAs. *International Symposium on Field-Programmable Gate Arrays,* 1998.
[62] P. Pan, C. L. Liu. Technology mapping of sequential circuits for LUT-based FPGAs for performance. *International Symposium on Field-Programmable Gate Arrays,* 1996.
[63] P. Pan, C. L. Liu. Optimal clock period FPGA technology mapping for sequential circuits. *Design Automation Conference*, June 1996.
[64] P. Pan, C. L. Liu. Optimal clock period FPGA technology mapping for sequential circuits. *ACM Transactions on Design Automation of Electronic Systems* 3(3), 1998.
[65] S. Safarpour, A. Veneris, G. Baeckler, R. Yuan. Efficient SAT-based Boolean matching for FPGA technology mapping. *Design Automation Conference*, July 2006.
[66] P. Sawkar, D. Thomas. Technology mapping for table lookup–based field-programmable gate arrays. *ACM/SIGDA Workshop on Field-Programmable Gate Arrays*, February 1992.
[67] M. Schlag, J. Kong, P. K. Chan. Routability-driven technology mapping for lookup table–based FPGAs. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 13(1), 1994.
[68] L. Shang, A. Kaviani, K. Bathala. Dynamic power consumption in Virtex-II FPGA family. *International Symposium on Field-Programmable Gate Arrays*, February 2002.
[69] N. Togawa, M. Sato, T. Ohtsuki. Maple: A simultaneous technology mapping, placement, and global routing algorithm for field-programmable gate arrays. *International Conference on Computer-Aided Design*, 1994.
[70] S. Wilton. SMAP: Heterogeneous technology mapping for area reduction in FPGAs with embedded memory arrays. *International Symposium on Field-Programmable Gate Arrays*, 1998.
[71] Z. H. Wang, E. C. Liu, J. Lai, T. C. Wang. Power minimization in LUT-based FPGA technology mapping. *Asia South Pacific Design Automation Conference,* 2001.
[72] H. Yang, D. F. Wong. Edge-map: Optimal performance-driven technology mapping for iterative LUT-based FPGA designs. *International Conference on Computer-Aided Design,* November 1994.