A Technology Mapping Algorithm for CPLD Architectures

Student: Shin-Liang Chen
Advisor: TingTing Hwang

Department of Computer Science
National Tsing Hua University
HsinChu, Taiwan 300

Contents

List of Figures

List of Tables

Abstract

In this thesis, we propose a technology mapping algorithm for CPLD architectures. Our algorithm proceeds in two phases: mapping for single-output PLAs and packing for multiple-output PLAs. In the mapping phase, based on the results in [4], we propose a Look-Up-Table (LUT) based mapping algorithm. We will take advantage of existing LUT mapping algorithms for area and depth minimization. We also study, for a given (i, p, o)-PLA block structure, the problem of selecting the values of input and product term constraints for mapping for single-output PLA. Benchmark results show that our algorithm produce better results in terms of area and depth as compared to those by TEMPLA.

Chapter 1

Introduction

Time to market pressure makes it imperative that design, development, production and testing time for VLSI chips be reduced as much as possible. Programmable Logic Devices (PLDs) offer an attractive solution to these problems. In particular, Field Programmable Gate Arrays (FPGA) and Complex Programmable Logic Devices (CPLD) have emerged as popular technologies due to their short design-production turn-around time and low manufacturing cost. In both FPGA and CPLD architectures, there are logic blocks for implementing the functions of a circuit. A block in FPGA is a k-input Look-Up Table (LUT) while CPLD is based on PLA-style blocks. A LUT block can implement any arbitrary Boolean function of up to k variables and a PLA block provides an and-plane and or-plane architecture for implementing two-level sum of products logic.

Architectures of FPGA and CPLD have been studied extensively [4, 5, 6]. FPGA offers high logic density and high capacity. However, when a design is implemented in FPGA, it might result in a slower circuit because critical paths go through multiple levels of blocks and blocks are connected by programmable interconnect blocks. Moreover, estimating the circuit delay is more difficult. On the other hand, CPLD offers lower logic density, but circuits implemented in CPLDs generally are faster than that in FPGAs because critical paths go through fewer levels of blocks and the delay within a block is much smaller than that between blocks. Moreover, LUT block size grows exponentially with input size while PLA-block size grows only quadratically [6]. Architecture study shows that in an FPGA, in terms of area and delay, the most efficient input size of a block is 4 or 5. On the other hand, in CPLD architecture, the most efficient choice is a PLA with 8-10 inputs, 3-4 outputs, and

12-13 product terms. Therefore, in terms of area and delay, CPLD is more efficient than FPGA as a coarse grain Programmable Logic Device.

Technology mapping is a process of transforming technology independent Boolean network into technology-based circuit. For LUT-based FPGAs, a technology mapper decomposes the Boolean network into a set of subnetworks such that the number of inputs of a subnetwork is no more than the LUT input constrain. For PLA-based technology mapping, a decomposed subnetwork is required to satisfy not only the input constrain but also the product term constrain.

Many LUT-based technology mapping algorithms have been proposed. Some of these algorithms focus on area minimization and others on delay/depth minimization [8, 10, 11, 9, 13]. On the other hand, only a limited number of algorithms have been published on synthesis for CPLDs [1, 2]. In [1], a tool TEMPLA was developed to minimize the number of PLAs. The mapping algorithm in TEMPLA proceeds in three steps: optimal tree mapping, partial collapsing, and output packing. In the steps of optimal tree mapping and partial collapsing, single output functions are mapped to single-output PLAs taking both input and product term constraints into consideration. In the step of output packing, single-output PLAs are packed to form multiple-output PLAs. In [2], delay/depth is the objective where the algorithm FlowMap [8] is modified so that for each PLA, both input and product term constraints are satisfied.

In this thesis, we will study technology mapping algorithms for CPLD. Our objectives are to minimize the number and the depth of PLAs. The rest of this thesis is organized as follows: In Chapter 2, existing technology mapping methods for LUT-based and PLA-based PLDs are reviewed. Some LUT-based FPGA and CPLD architectures are also presented.

Chapter 3 presents our target architecture and the motivation behind our mapping algorithm. The detailed description of the algorithm is given in Chapter 4. Chapter 5 gives experimental results. The quality of the technology mapping algorithm is compared with the results produced by tool TEMPLA. Conclusions are given in Chapter 6.

Chapter 2

Background and Previous Work

2.1 Introduction

This chapter will take a brief introduction to the logic block architecture of the LUT-based and PLA-based PLDs. Due to different logic block structures, different constraints in technology mapping are considered. We will first present the FPGA synthesis flow and compare the constrants of these two logic block structure. Finally, some LUT-based and PLA-based PLDs technology mapping methods are reviewed.

2.2 FPGA Synthesis Flow

The FPGA synthesis flow is shown in Figure 2.1. The input to the synthesis CAD flow may be a description in a hardware description language such as Verilog or VHDL. In the first step, logic optimization, some technology independent optimizations are accomplished. In technology mapping step, it decides which part of the network be implemented in an unit block. For area minimization, how to use minimum blocks to cover entire network is considered. For delay minimization, it minimizes the number of blocks in the critical paths. After technology mapping, placement decides the physical location of the logic blocks in the FPGA. The blocks location affects the interconnect delay and the routibility. Routing step is to connect all interconnects among blocks. Finally, the circuit downloads to the chip.

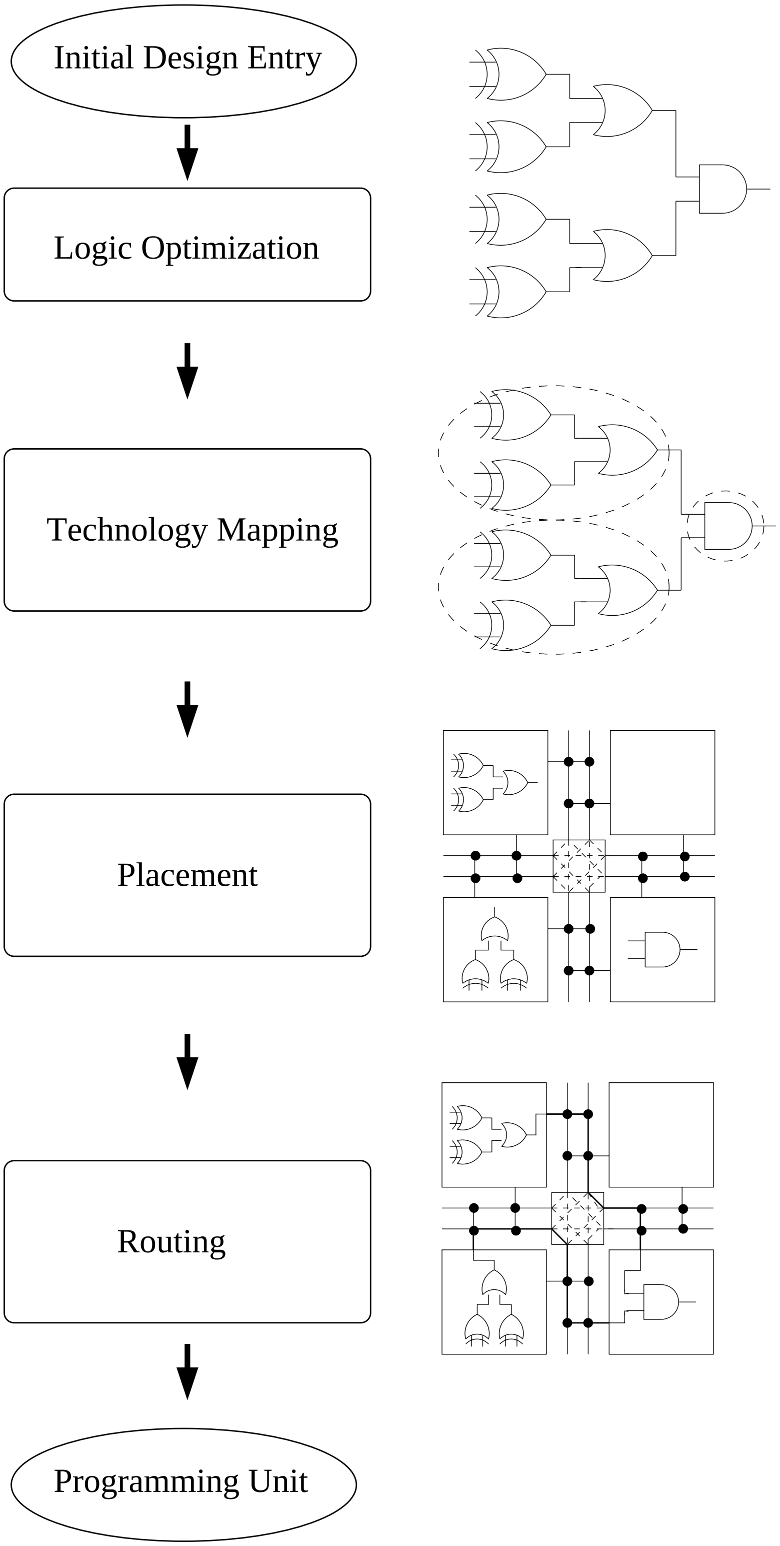


Figure 2.1: The FPGA synthesis flow.

2.3 Technology Mapping Methods

2.3.1 Look-Up-Table Logic Blocks

The look-up-table logic block is essentially an SRAM (e.g., Configurable Logic Block (CLB) of Xilinx.) The truth table for a k-input logic function is stored in a $2^k \times 1$ SRAM. The address lines of the SRAM are the input of the logic function. Therefore, a LUT can implement any logic function with input no more than k. For example, the truth table and LUT-block implementation of logic function $f = ab + c$ are shown in Figure 2.2.

a	b	c	f
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

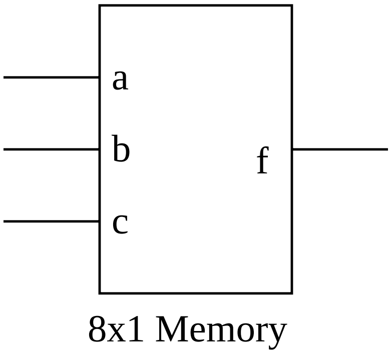


Figure 2.2: Look up table-based logic.

If we apply the traditional cell-based technology mapping method for K-LUT FPGA synthesis. We need a cell library for mapper to replace original circuit by library cells. In this case, the library size will be 2^{2^k} in order to represent all k-input functions. For large k, the run time cost of mapping will be large. So the cell based technology mapping method does not fit to LUT-based PLDs technology mapping. Instead of building a cell library, LUT-based technology mapper decomposes a network into subnetworks such that the number of inputs of subnetworks satisfies the k-input constraint.

In the following, we review some well-known technology mappers. In *Mis-pga* [9], the first step is to decompose the nodes with more than k inputs in the network. The decomposition is performed by Roth-Karp decomposition and Kernel extraction. After the decomposition step, all nodes has inputs less than k. The second step is to cover the network such that each covered subnetwork with input no more than k. The covering step is performed by max-flow min-cut algorithm.

Chortle-crf [10] is an graph-based LUT technology mapping method. It divides the input Boolean network into a forest of trees and then optimally maps each tree. Gate-level decompositions and bin-packing methods are chosen to perform the mapping. The method also exploits reconvergent paths and replication of logic at fanout nodes to reduce the number of logic blocks.

DAG-Map [13] is a delay optimization mapper. It has two phases. Phase one labels each node in the network in a topological order. Primary inputs has a level of 0. Internal nodes with a level of h means at least h level of LUTs are needed to cover the subnetwork rooted at this node. Phase two maps the network into LUTs. A top-down method to map nodes into LUT with minimum depths is used.

Flow-Map [8] similar to *DAG-Map* is a depth minimization algorithm for LUT-based technology mapper. Given an input constraint k, *Flow-Map* computes a minimum height k-feasible cut in a network by network flow based computation. *Flow-Map* is performed in two phases: labeling and mapping. Labeling phase gives all nodes in network a label by min-cut-max-flow method. The label is the depth of the optimal mapping solution when the subnetwork is rooted at this node. In mapping phase, according to the labeled value of nodes, a top-down method is used to decide which k-feasible cut will be mapped into a LUT-block.

2.3.2 Programmable-Logic-Array Logic Block

The programmable-logic-array block is divided into two parts: one is **AND**-plane and the other is **OR**-plane in Figure 2.3. The **AND**-plane provides the product terms and the **OR**-plane collects these product terms into correct multi-output functions.

The PLA-based technology mapping has more constrains than LUT-based mapping. The subnetwork can be mapped to a logic block satisfies not only the number of inputs but also the number of product terms. Therefore, it is more diffcult to develop a mapping algorithm to satisify these constrains at the same time. In the recent years, many studies on the LUT-

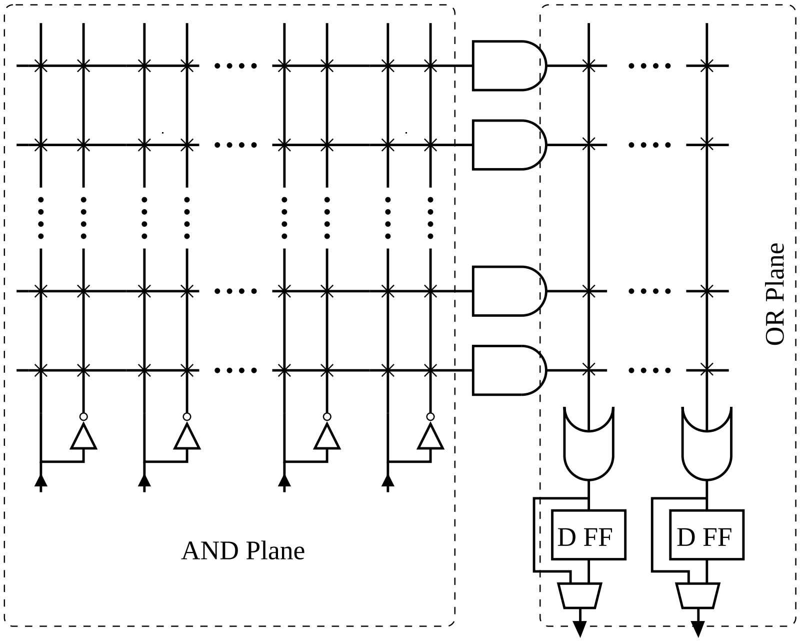


Figure 2.3: Programmable-Logic-Array logic.

based technology mapping have been published. But few of PLA-based technology mapping method can be found. *TEMPLA* [1] is a PLA-based mapping tool. It divides into three phases. Phase one begins by partitioning a circuit's DAG into a forest of fanout-free trees. Then, dynamic programming is used to perform an optimal tree mapping. In the phase two, partial collapse is performed. Nodes that can be collapsed into their fan-out are eliminated. Phase three, first-fit-decreasing bin-packing algorithm is performed to pack single output PLAs into multi-output PLA blocks.

Chapter 3

Target Architecture and Motivation

The target architecture of our study is a number of PLA-style blocks connected through a programmable interconnection network is shown in Figure 3.1. The structure of a block is shown in Figure 3.2 [1]. A block is characterized by the number of inputs to the block, i, the number of product terms in the block, p, and the maximum number of outputs, o. We use the notation an (i, p, o)-PLA to denote one such block. The goal of our technology mapping algorithm is to minimize the number of PLA blocks or the delay/depth of a combinational circuit.

Previous technology mapping algorithms [1, 2] take a two-stage approach. In the first stage, mapping for single-output PLA is performed. In the second stage, the single-output PLAs are packed to form multiple-output PLAs. Since the packing algorithm is quite straight forward and well understood, both of these algorithms place emphasis on the first stage, the mapping stage. Unlike FPGA LUT block mapping which only need to meet the input constraint, single-output PLA block mapping must meet two constraints: the input constraint and the product term constraint. The key idea of both algorithms is to modify technology mapping algorithms for standard cell and LUT FPGA [1, 2] to satisfy these two constraints. In [1], a tree covering algorithm for standard cell is modified so that only candidates satisfying the input and product term constraints are considered. In [2], a technology mapping algorithm for LUT FPGA is modified so that only cuts satisfying both constraints are considered. Since it is more difficult to meet the k-inputs, p-product terms constraints at the same time, computation is more complicated.

There are two distinct features in our approach: (1) In our algorithm, we consider the

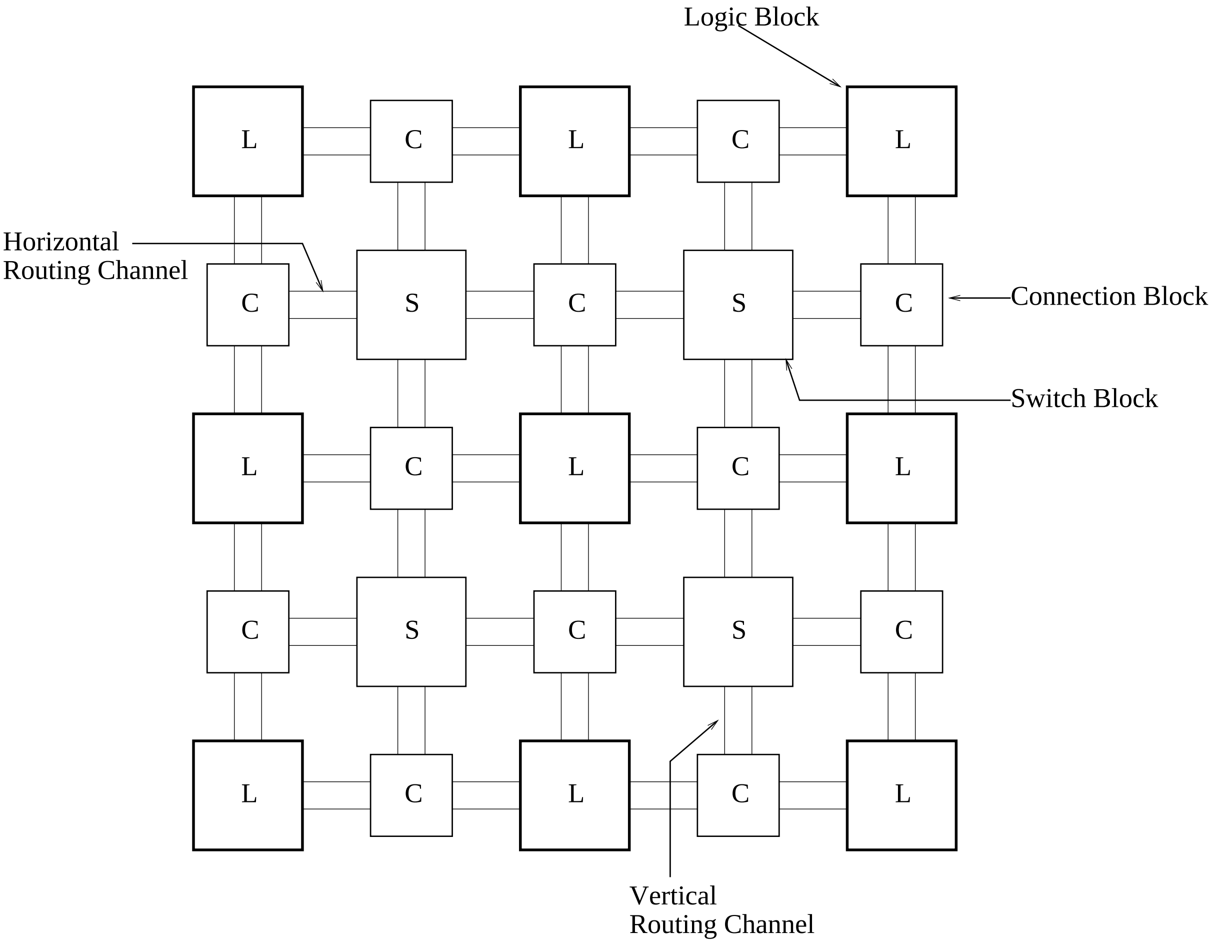


Figure 3.1: The architecture of a symmetric CPLD.

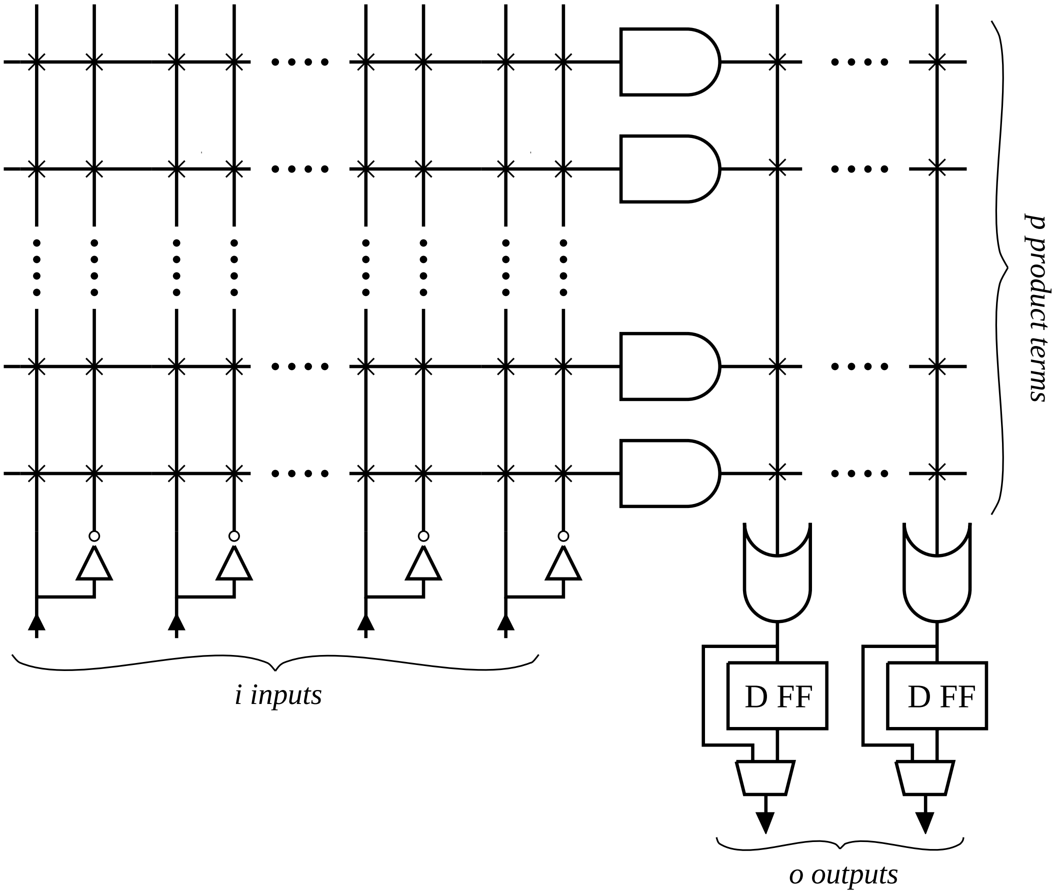


Figure 3.2: The structure of a (i, p, o) PLA block.

constraints one at a time. First, we will apply an LUT mapper to generate logic blocks that satisfy the input constraint. Then, we check if each logic block generated this way satisfies the product term constraint. If not, we perturb a partial network which is constructed by collecting all logic blocks that do not satisfy the product term constraint and perform LUT mapping again. The motivation for this such an approach is twofold. Research in [6] shows that for most benchmark circuits, the average number of product terms used in a LUT block is linear in the number of inputs. For example, for LUT blocks with 5, 6, 7, 8 inputs, the average number of product terms used in a logic block is 2.88, 3.30, 3.84, 5.12, respectively. Therefore, if we carefully select the input size for LUT mapping, it is likely that most of LUT blocks generated will also meet the product term constraint. Second, there exist many good LUT-based FPGA mapping algorithms for either small area or small depth/delay. We can take advantage of these algorithms and utilize these tools to minimize either area or depth/delay.

(2) We also note that previous research [1, 2] did not take the step of packing single-output PLAs to form multiple-output PLAs into consideration in the mapping phase. They set the input and product term constraint the same as those of multiple-output PLA block. However, in order to pack more single PLA blocks to form multiple-output PLAs, the number of inputs and the number of product terms in mapping phase should be chosen to be less than that of the multiple-output PLA blocks. We will have a detailed discussion on how to select the input and product term constraints in the mapping phase.

Chapter 4

Algorithm Descriptions

Before the presentation of our algorithm description, we first define some terms. Let the given circuit be represented as a directed acyclic graph $G = (N, E)$. Then, A **logic block** is a block of nodes in G which implements a single logic function. A **PLA logic block** is a logic block that can be implemented as a PLA block. A **LUT logic block** a logic block that can be implemented as a LUT block.

Our algorithm proceeds in two stages: mapping for single-output PLAs and packing them to form multiple-output PLAs. In the mapping stage, the first step is to select the input-size k ($k \leq i$, where i is the input constraint of the (i, p, o)-PLA blocks) for LUT mapping. The second step is to perform k-input LUT mapping. The third step is to call a two-level logic minimization tool, ESPRESSO, to check if every single-output PLA satisfies the product term constraint. If there are PLA blocks that do not satisfy the product term constraint, in the fourth step, re-map for these nodes is performed.

For area minimization, we collect all these blocks to form a partial network, perturb the partial network, and then iterate the selection of input-size k and the LUT mapping steps. The loop continues till all logic blocks satisfy the input and product term constraints. For delay minimization, we decompose these blocks by a weighted functional decomposition and iterate the decomposition until all blocks satisfy product term constraints. The last step is to perform output packing.

4.1 Area Minimization

The flow of the area minimization algorithm is depicted in Figure 4.1. In the following, we will explain each step in more detail.

Step 1: Selection of Input-size k for LUT Mapping. First, we select the input-size k for the LUT mapping. The selection of k is determined by the parameters i, p, o of a PLA block. Its selection will affect the final results significantly. If a large k is selected, a logic block may cover many nodes of the circuit. Hence, the depth of the mapped circuits is smaller and the number of logic blocks is also smaller. However, the number of product terms in a logic block will be large [6] and is likely to exceed the product term constraint. Moreover, when the value of k is close to that of i, it will become difficult to pack single-output PLAs to multiple-output PLAs.

Our selection rule is based on the results in [6]. The research shows that for most benchmark circuits, the average number of product terms used in a LUT block is linear in the number of inputs. For example, for LUT blocks with 5, 6, 7, 8 inputs, the average number of product terms used in a LUT block is 2.88, 3.30, 3.84, 5.12, respectively. Suppose we are given an (i, p, o)-PLA block, we are to select the input size, k, for LUT mapping. Let $npterm_k$ denote the average number of product terms used in a k-input LUT block. Then k must be selected such that the following equation is true.

$$(npterm_k \times o)/1.1 \leq p$$

The term, $npterm_k \times o$, estimates the total number of product terms for o LUT blocks. Since the product terms can be shared among different outputs if these blocks are implemented in an (i, p, o)-PLA, the actual number of product terms can be reduced. Based on the research in [6], about 10% of the product terms can be shared. Hence, the quantity $npterm_k \times o$ is divided by 1.1. The $\leq$ means that the computed quantity on the left hand side is smaller than but as close to p as possible. For example, for $(10, 12, 4)$-PLAs, k must be chosen as 6 because $npterm_6 = 3.30$ and $(3.30 \times 4)/1.1 = 12$. Experimental results on the effect of varying k is presented in Chapter 5.

Step 2: LUT Mapping. The second step is to perform k input LUT mapping. In this step, among the three constraints, we consider only the input constraint. The reason

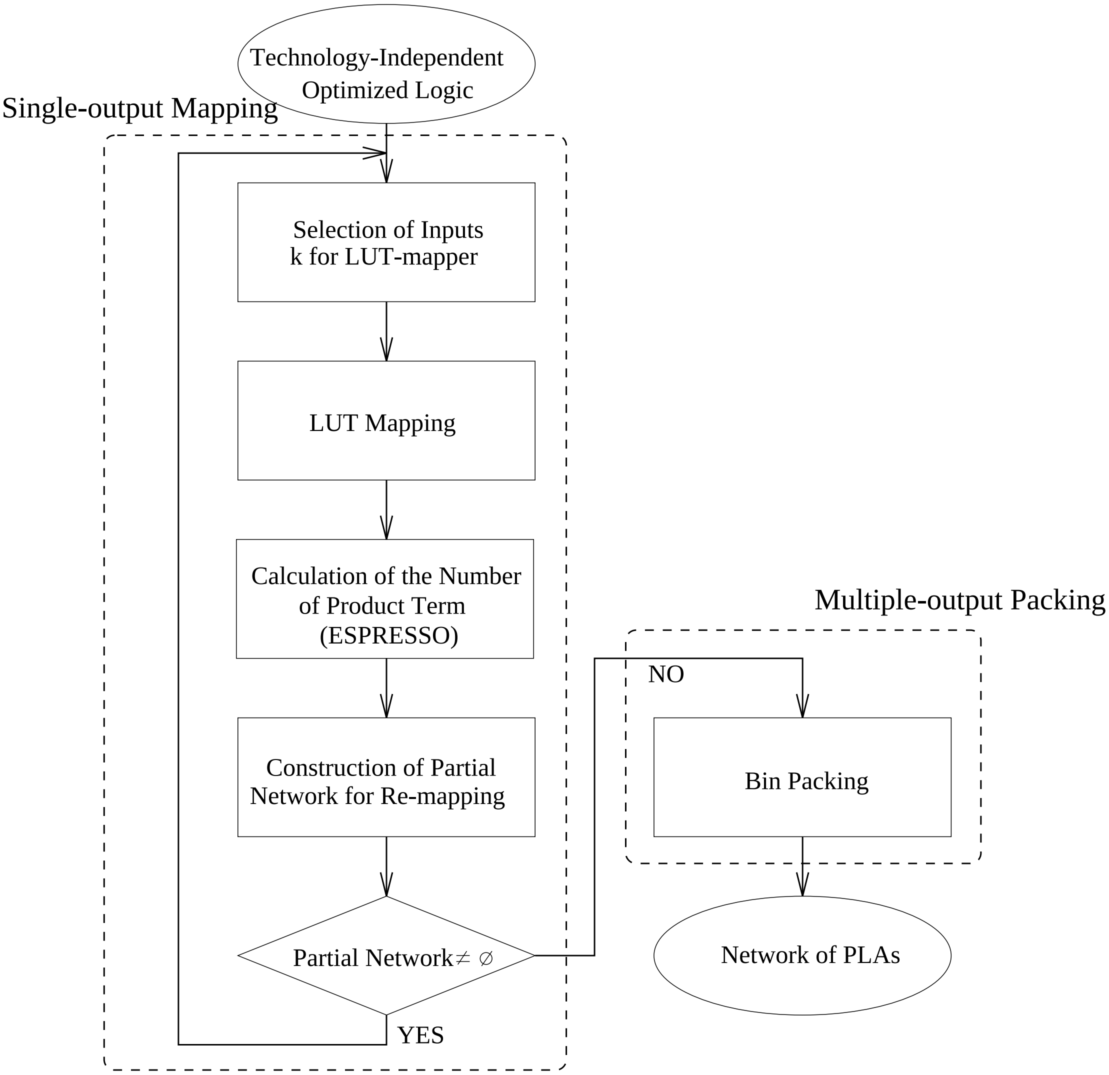

Technology-Independent Optimized Logic
Single-output Mapping
Selection of Inputs k for LUT-mapper
LUT Mapping
Calculation of the Number of Product Term (ESPRESSO)
Construction of Partial Network for Re-mapping
Partial Network≠ ∅
YES
NO
Multiple-output Packing
Bin Packing
Network of PLAs

Figure 4.1: The flow of area minimization algorithm.

is based on the research [6] where it was shown that for most benchmark circuits, the average number of product terms used in a LUT block is very small. Therefore, if k is carefully selected, most of LUT block generated will also satisfy the product term constraint. Moreover, many efficient LUT-based FPGA mapping tools have been developed for area or depth/delay minimization, we can test many alternatives and select the best one. In this thesis, for area minimization, the SIS standard FPGA mapping script [12] will be used. The objective of the script is to minimize LUT block counts.

Step 3: Calculation of the Number of Product Terms for a Logic Block. After the LUT mapping is performed, we have a network of logic blocks satisfying the input constraint. The next step is to check if each logic block satisfies the product term constraint. We will call a two-level logic minimization tool to minimize the logic and compute the number of product terms in each logic block. ESPRESSO [12] is used.

Step 4: Construction of Partial Network for Re-mapping. After the initial mapping and calculation of the number of product terms for each logic block, the next step is to pick out the logic blocks that satisfy both input and product term constraints which will be retained in the final mapping solution.

It should be noted that we are free to set the product term constraint for each logic block. Let the product term constraint for each logic block be denoted *lbp*. Then, the first condition for *lbp* is

$$lbp \leq p$$

where p is the product term constraint for an (i, p, o)-PLA block. However, in order to pack more single-output logic blocks to form a multiple-output PLA block, intuitively, we should set *lbp* to a value smaller than p. To verify this intuition, we have performed a set of experiments on varying the value of *lbp*. The experimental results are shown in Figure 4.2, where four circuits were tested for $(10, 12, 4)$-PLA blocks. For different product term constraints for a logic block, the top figure shows the number logic blocks that do not satisfy the product term constraint, *lbp*, (called *lbp*-infeasible logic blocks) after the first iteration of LUT mapping. The middle figure shows the number of logic blocks which satisfy the *lbp* product term constraint after all the iterations of the mapping phase, and the bottom figure shows the number of (10, 12, 4)-PLA blocks after packing.

Somewhat unexpectedly, our experimental results show that when *lbp* is set to p, $p-1$,..., $p-4$, the number of *lbp*-infeasible logic blocks, the number of logic blocks before packing, and the final (10, 12, 4)-PLA blocks are the same. However, when *lbp* is set to be a relatively small number, 4, the final results become much worse than those when *lbp* is set to 12.

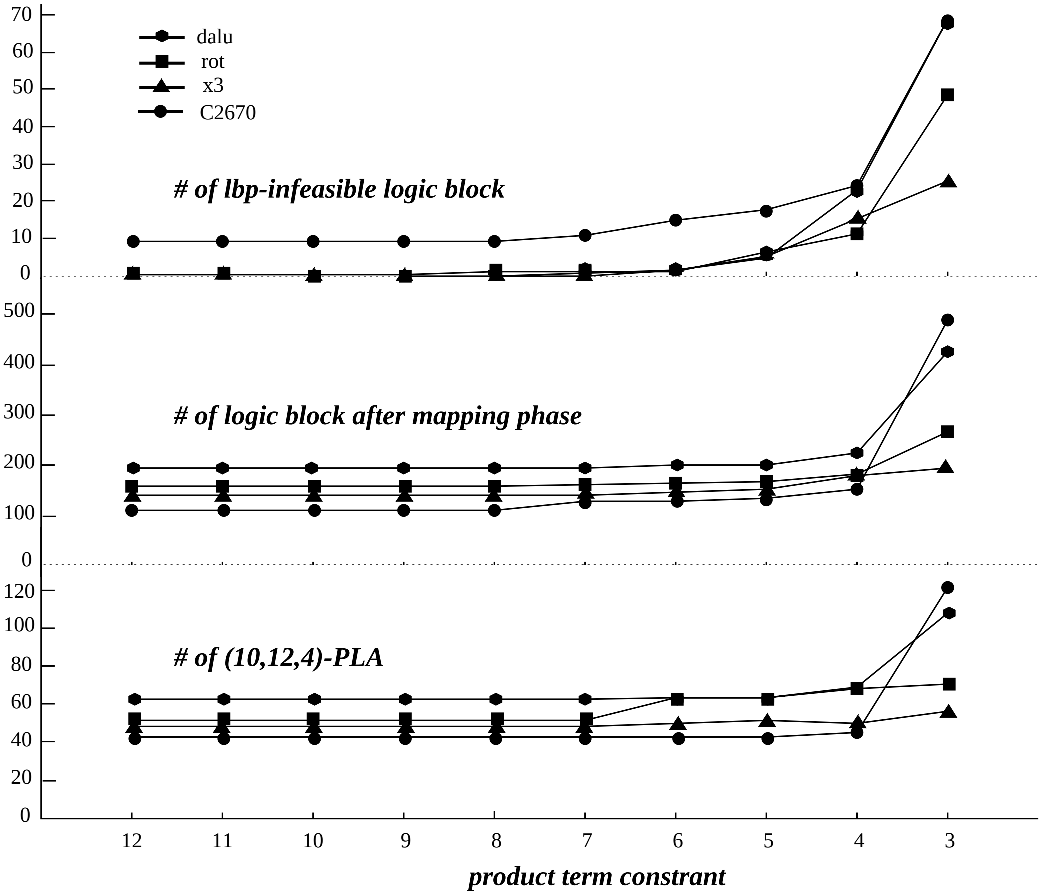


Figure 4.2: Varying the product term constraints, *lbp*.

The explanation is as follows. After the initial mapping, most of logic blocks already satisfy the product term constraint, *lbp*. Only very few logic blocks have a large number of product terms. The functionalities (e.g., xor-type) of these logic blocks make it impossible for them to have small number of product terms. Therefore, if *lbp* is set too small, these *lbp*-infeasible logic blocks will be decomposed into many logic blocks. The decrease in the number of PLA block after packing do not offset the increase in the number of logic blocks due to setting *lbp* to a small number. Therefore, we will set *lbp* to be p,

$$lbp = p$$

After LUT mapping, we mark all *lbp*-infeasible logic blocks. For these *lbp*-infeasible logic blocks, instead of decomposing them one by one, we collect them to form a partial network. Then, we re-synthesize the partial network and iterate the mapping phase. The purpose of re-synthesis is to perturb and optimize the partial network so that in the iterated LUT mapping step, it will not result in the same mapping solution.

In the next iteration of LUT mapping, the value of the input constraint k will be selected again. The heuristic is as follows. First, we will try the same value for k in the first iteration of LUT mapping. However, if no new feasible logic block is produced, the value of k is decreased by 1 at a time until new feasible logic blocks are produced.

The LUT mapping step is iterated until the partial network is empty. Figure 4.3 show an example of our algorithm, where k is selected to be 4 for LUT mapping and the product term constraint for each logic block, *lbp*, is set to 3. Figure 4.3(a) shows the result of LUT mapping, where 4 logic blocks are produced. Figure 4.3(b) shows the number of product terms computed for each logic block and two logic blocks have more than *lbp* product terms, 3, which are collected to form the partial network shown in Figure 4.3(c).

Step 5: Bin-packing for Multiple-output PLAs. Bin-packing step will merge logic blocks that satisfy both input and product term constraints into multi-output PLA blocks. Since the bin packing is well understood, this step is accomplished using packing algorithm proposed in TEMPLA. TEMPLA uses a first-fit-decreasing bin packing algorithm that attempts to maximize the number of shared inputs between logic blocks that are packed into the same PLA.

4.2 Delay Minimization

LUT-based FPGA have been widely studied. Many LUT-based mapping algorithms have been designed to minimize the delay. Among them, *Flow-Map* can produce network with minimal level. In the first step, our algorithm will use a LUT-based FPGAS algorithm for delay minimization. Then, for those nodes that do not satisfy the product term constraint, a

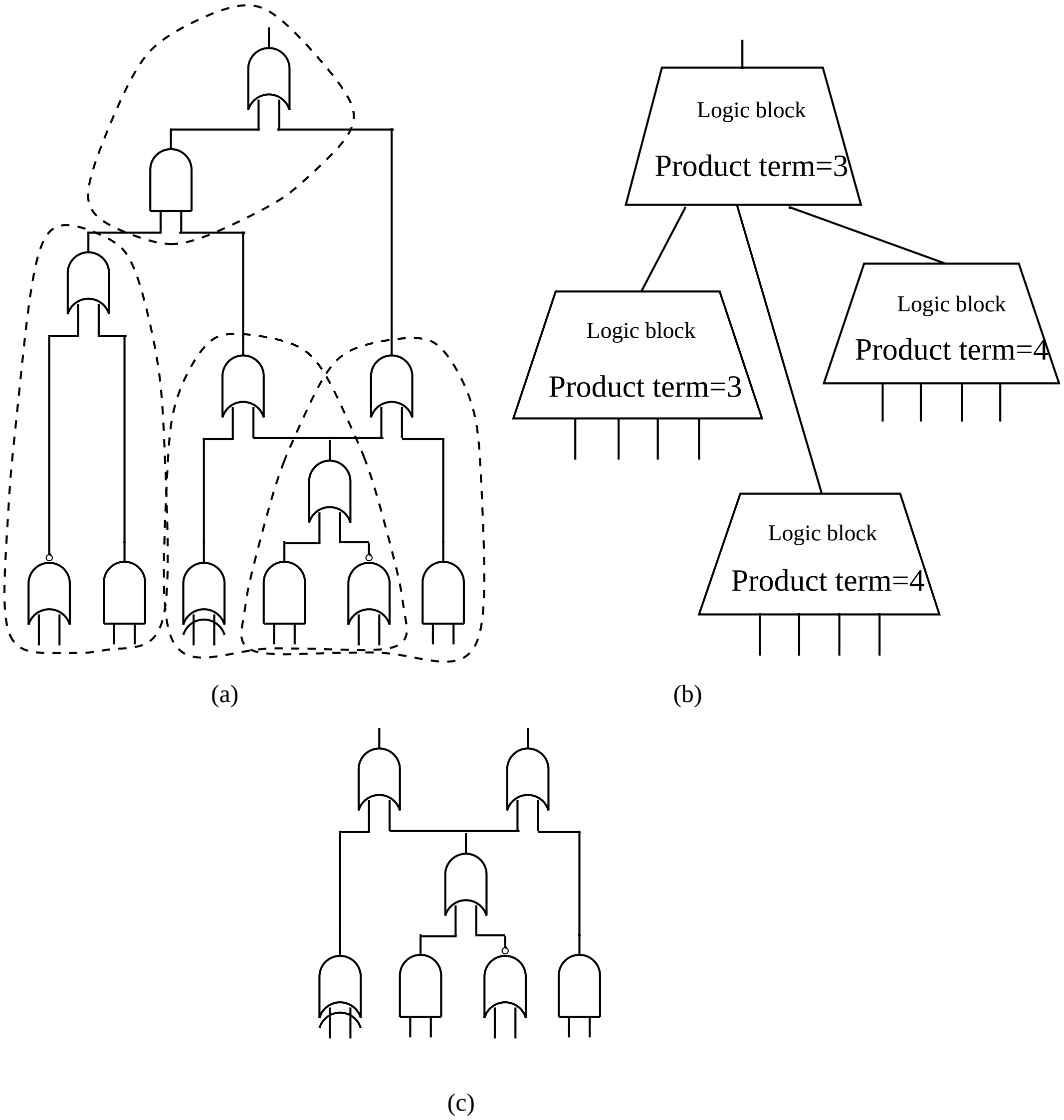


Figure 4.3: An example of single-output mapping:(a) LUT mapping (b) product terms of each logic block (c) construction of partial network for re-mapping.

level driven decomposition for those nodes will be performed. The flow of delay minimization algorithm is depicted in Figure 4.4. In the following, we will explain each step in more detail.

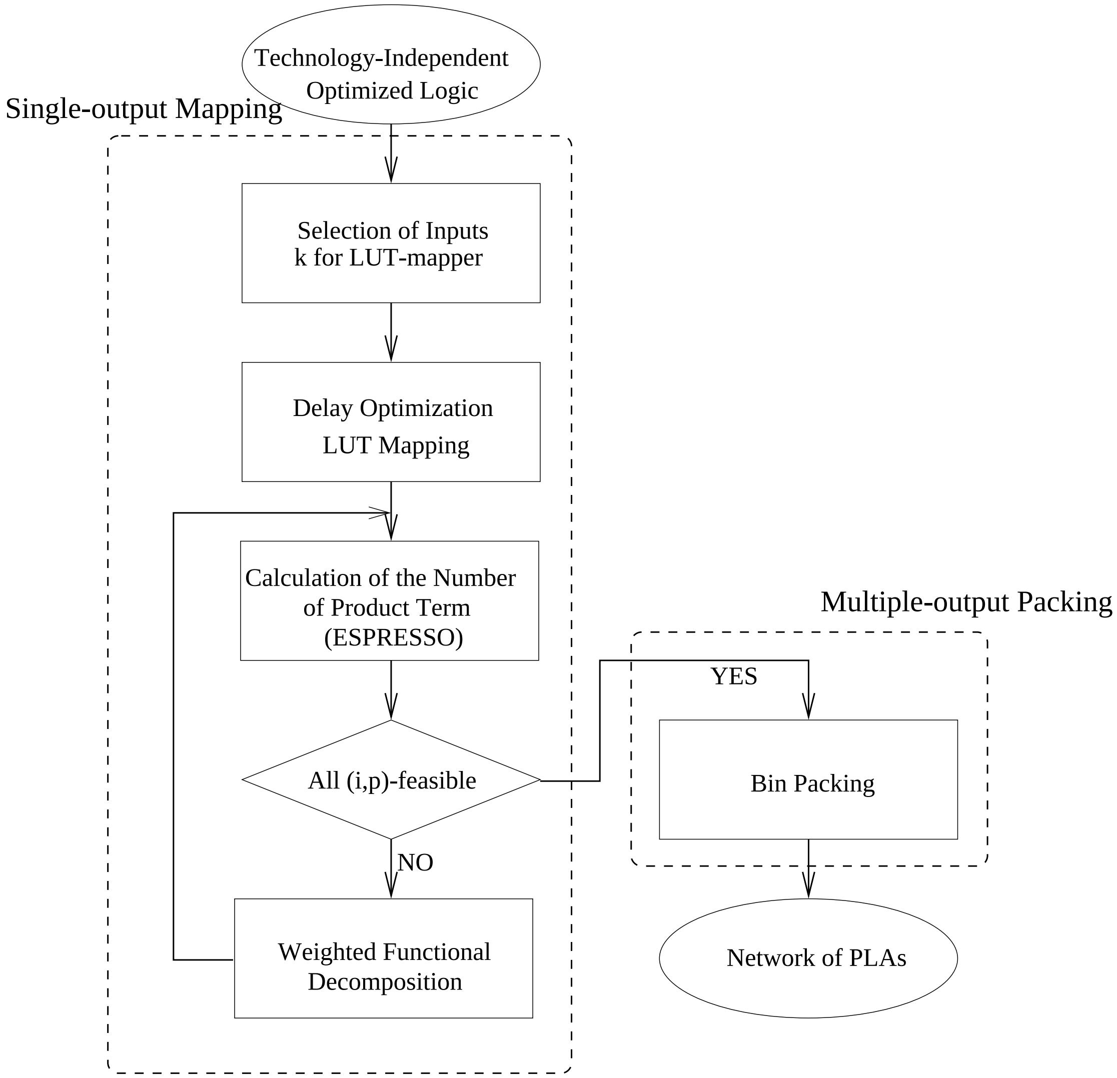

Technology-Independent Optimized Logic
Single-output Mapping
Selection of Inputs k for LUT-mapper
Delay Optimization LUT Mapping
Calculation of the Number of Product Term (ESPRESSO)
All (i,p)-feasible
NO
YES
Weighted Functional Decomposition
Multiple-output Packing
Bin Packing
Network of PLAs

Figure 4.4: The flow of delay minimization algorithm.

Step 1: Selection of Input-size k for Delay Minimization of LUT Mapping. Similar to area minimization method, we have to first select the input-size k for delay minimization of LUT mapping. We use the same method as that for area minimization to decide the input-size. Experimental results on the effect of varying k is presented in Chapter 5.

Step 2: LUT Mapping. LUT-based technology mapping algorithm for delay optimization is applied. For delay/depth optimization, *Flow-Map* can be used. It produces mapping results of optimal depth. Similar to area minimization, the selected input-size allows most of generated blocks satisfying the product term constraint.

Step 3: Calculation of the Number of Product Terms for a Logic Block. This step will be the same as step 3 in algorithm for area minimization.

Step 4: Weighted Functional Decomposition. In this step, all blocks of the network satisfy the input-size constraint. But some of them have product term more than product term constraint. These nodes need to be further decomposed so that each that all blocks satisfy the product term constraint.

In step 2, the LUT mapping algorithm produces results of optimal level. When the p-infeasible blocks are further decomposed, we should try not to increase the level of the network. Functional decomposition will be used to decompose the p-infeasible blocks.

Given a logic function f and a partition of its n input variables into the bound set $\{x_1, \ldots, x_i\}$ and the free set $\{x_{i+1}, \ldots, x_n\}$, functional decomposition determines *encoding functions* $d_1, \ldots, d_j$ and the *base function* g, such that

$$f(x_1, \ldots, x_n) = g(d_1(x_1, \ldots, x_i), \ldots, d_j(x_1, \ldots, x_i), x_{i+1}, \ldots, x_n).$$

From the above functional decomposition, we know that the number of inputs to base function g is equal to $j + n - i$ where j is the number of encoding functions, n is the number of inputs to f and i the number of input in the bound set. It allows the number of inputs to g is no more than that to the original function f. This is an important property because after performing the functional decomposition, all nodes will still satisfy the input-size constraint.

The next question is how to select bound set. In our decomposition, both area (the number of blocks) and delay (the level of the network) are considered.

Level of a node is usually used to estimate the circuit delay. We will first define the level of a node as follows. The level of primary input is 0 and the level of node, v, is defined as

$$level(v) = \max_{v_j \in fanin(v)} level(v_j) + 1.$$

The selection of bound set proceeds as follows. First, we sort all inputs in increasing order of their level. Then, we form candidate bound sets as follows. The first bound set contains the first inputs in the sorted list. Then, the next bound set is formed by adding the next input in the sorted list. The same adding procedure continues to form more bound sets until only two inputs are left in the free set. For each bound set generated, we test if it is decomposable i.e., the number of encoding functions is less than the size of bound set. If it is not decomposable, we delete it from the generated bound sets.

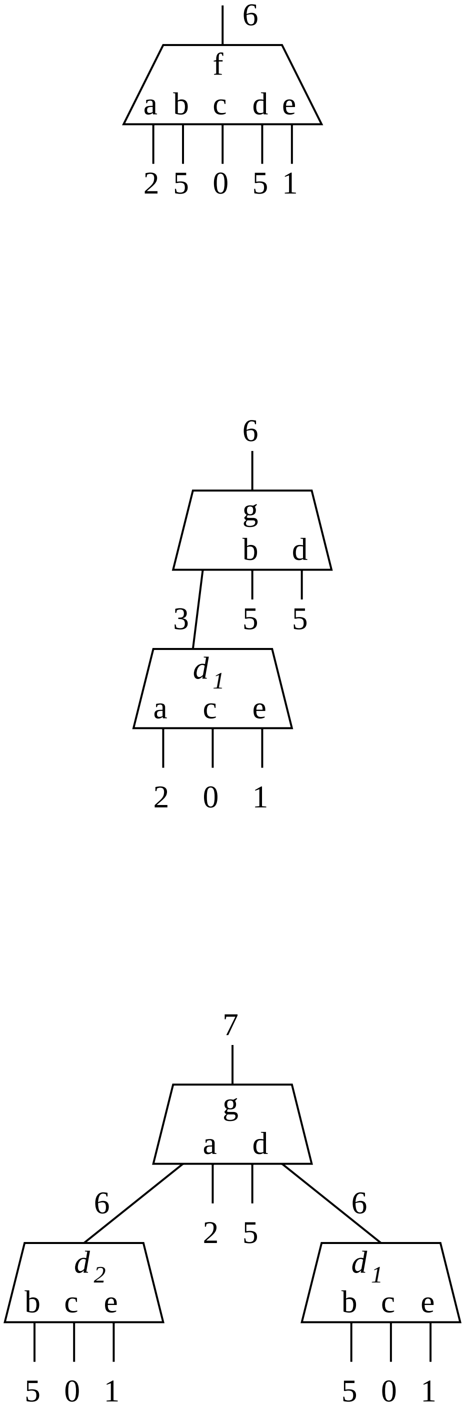
The next step is then to compute the cost of each bound set. The cost of bound set takes level and the number of product-term into consideration. Let $P(g)$ denote the number of product term of node (or function) g, $S = \{f \mid P(f) >$ product term constraint and f is an encoding function or base function $\}$, $C = \{q \mid q$ is an encoding function with critical inputs $\}$. Then, the cost function of a bound set is defined as follows:

$$Cost = \alpha \times (\sum_{k=1}^{j} P(d_k) + P(g) + \sum_{f \in S} P(f)) + (1 - \alpha) \sum_{q \in C} P(q).$$

where d_k are encoding functions and g base function for the bound set. The first three terms give the area penalty and the last term the level increase penalty. α is a ratio to control the effects of area and level. Figure 4.5 shows an example to compute the costs of two bound sets.

After we compute the cost for all candidate bound sets. We select the one with minimal cost. Finally, if a node is found to be not decomposable, the and-or decomposition will be used. The decomposition repeats until all nodes satisfy the product term constraint.

Step 5: Bin-packing for Multiple-output PLAs. The final step of delay minimization is also to merge logic blocks that satisfy both input and product term constraints into multi-output PLA blocks.



level(f)=max(level of fanins)+1
=5+1
=6

$P(f) = 10$

$level(a)=2$
$level(b)=5$
$level(c)=0$
$level(d)=5$
$level(e)=1$

bound set = {a,c,e}
$P(d_1)=6$
$P(g)=7$
$level(g)=5+1=6$
$Cost=0.5(6+7+0)+0.5(0)=7.5$

bound set = {b,c,e}
$P(d_1)=5$
$P(d_2)=6$
$P(g)=11$
$level(g)=6+1=7$
$Cost=0.5(5+6+11+11)+0.5(6+5)$
$=22$

Figure 4.5: The computation of cost for two bound sets.

Chapter 5

Experimental Results

Our algorithms were implemented in C language and executed on a SUN Ultra 80 platform. Several experiments were conducted to demonstrate their effectiveness using a randomly selected subsets of the MCNC benchmark suite. The experiment was executed in SIS framework [12]. The initial circuits were optimized by SIS *rugged_script*. In the LUT-mapping step, LUT mapping script in SIS is used to minimize the number of LUT blocks. In the two-level logic minimization step, ESPRESSO is called to minimize the number of product terms of each LUT block.

5.1 Area Minimization

5.1.1 Different Input-size k for LUT Mapping in Area Minimization

The first experiment is to understand for a given (i, p, o) architecture of PLA blocks, how the value of input-size k for LUT mapping affects the final results. In this experiment, a (10,12,4)-PLA was selected as our target PLA block architecture because it was also used in [1].

Table 5.1 shows the area utilization results. The columns labeled LUT are the numbers of single-output LUT blocks produced after the mapping phase. The columns labeled PLA are the number of multiple-output PLA blocks after bin-packing. The columns labeled UT is the utilization of multiple output feature of the PLA block which is defined as $UT = LUT/PLA$. We can see from the table that when k is decreased from 7 to 5, the number of LUT blocks

Table 5.1: The effect of varying value k for LUT mapping

Benchmarks	$k = 7$			$k = 6$			$k = 5$		
	LUT	UT	PLA	LUT	UT	PLA	LUT	UT	PLA
apex6	136	2.83	48	144	3.06	47	166	3.25	51
dalu	164	2.10	78	197	3.08	64	229	3.31	69
rot	150	2.88	52	162	3.38	48	179	3.72	48
x3	142	2.96	48	148	3.22	46	189	3.57	53
frg2	171	2.80	61	180	3.33	54	231	3.79	61
i8	188	1.98	95	215	2.53	85	252	2.93	86
pair	253	2.01	126	295	3.01	98	346	3.36	103
apex5	185	2.53	73	203	2.86	71	247	3.25	76
C499	88	2.20	40	78	2.11	37	82	1.86	44
duke2	124	2.38	52	144	3.06	47	153	3.40	45
C2670	133	2.89	46	115	2.74	42	142	3.23	44
Total	1734	2.41	719	1889	2.96	639	2216	3.26	680

LUT: number of single-output LUT nodes.
PLA: number of multiple-output (10,12,4)-PLA.

is increased. However, when k= 7, the smallest number of LUTs count did not result in the smallest number of PLAs. This is because a large LUT block uses many inputs and product terms and hence it is difficult to pack them. Such an observation is justified by the low output utilization. On the other hand, when k is 5, although it has the highest output utilization, too many LUT blocks were produced. Hence, it results in a large number of PLAs.

Based on our algorithm, for the $(10, 12, 4)$-PLA block architecture, we compute the expected number of product terms in a PLA block for $k = 5, 6, 7$ as $(2.88 \times 4)/1.1 = 10.47$, $(3.30 \times 4)/1.1 = 12.00$, $(3.84 \times 4)/1.1 = 13.96$, where 2.88, 3.3, 3.84 are the average number of product terms used in a LUT block [4, 5], 1.1 is the utilization (sharing) of one product term among multiple outputs [4, 5], and 4 the number of outputs per PLA block. It can be seen that for $k = 6$ the computed value, 12, is closest to the number of product terms allowed in our PLA block. Hence the result for $k = 6$ is the best among the three cases.

5.1.2 Comparison with TEMPLA in Area

The second experiment is to compare our algorithm with TEMPLA. The (10,12,4)-PLA was selected as our target PLA logic block architecture. In this experiment, k is set to 6 in LUT Mapping step. The mapping phase of TEMPLA is based on the optimal tree covering algorithm. It is also implemented in the SIS environment and utilizes the ESPRESSO minimization routine. The initial network of TEMPLA is required to be an 8-bounded circuit. We apply command *tech_decomp -a 2 -o 2* to the input network to make sure that the network is 8-bounded circuits.

Table 5.2 shows the results of PLA block counts and the circuits depth. The results show that for 8 out of 11 circuits, our results are superior to those of TEMPLA in terms of the number of PLAs and 7 out of 11 circuits in terms of depth. On average, our algorithm is 6% better in terms of the number of PLAs and 5% in terms of depth as compared to TEMPLA.

Table 5.2: Results of comparison with TEMPLA in area minimization

	(10,12,4)-PLA	
	OURS	TEMPLA
Benchmarks	area/depth	
apex6	47/7	49/5
dalu	64/9	85/10
rot	48/11	47/11
x3	46/6	50/6
frg2	54/8	54/7
i8	85/10	88/7
pair	98/10	92/12
apex5	71/7	73/6
C499	37/6	39/9
duke2	47/7	45/10
C2670	42/9	52/12
Total	639/90	677/95
Average	58.1/8.2	61.6/8.64
Ratio		+6%/+5%

Table 5.3: The effect of varying value k for level

	$k = 8$		$k = 7$		$k = 6$	
Benchmarks	LUT/lev	PLA/lev	LUT/lev	PLA/lev	LUT/lev	PLA/lev
apex6	170/3	170/3	157/3	157/3	198/4	198/4
dalu	131/3	131/3	229/4	229/4	237/4	237/4
rot	208/5	231/5	220/5	220/5	275/6	275/6
x3	146/3	146/3	175/3	175/3	198/3	198/3
frg2	271/4	271/4	269/4	269/4	290/4	290/4
i8	288/3	288/3	355/4	355/4	359/7	359/4
pair	328/5	341/5	393/5	393/5	460/6	460/6
apex5	285/3	285/3	306/3	306/3	377/4	377/4
C499	70/3	108/5	76/4	88/4	60/4	76/5
duke2	131/3	131/3	154/3	154/3	185/4	185/4
C2670	145/5	161/5	217/5	227/5	195/6	204/6
Total	2173/40	2245/42	2551/43	2573/43	2834/49	2859/50
		+3%/+5%		+.8%/+0%		+.8%/+2%

LUT: number of single-output LUT nodes.
PLA: number of signal-output (10,12,1)-PLA.

5.2 Delay Minimization

5.2.1 Different Input-size k for LUT Mapping in Delay Minimization

This experiment is to find out how the value of input-size k for LUT mapping affects the level of the network. Table 5.3 shows the results. The columns labeled LUT/lev are the numbers of single-output LUT blocks produced and the level for each circuit after the mapping phase is performed. The columns labeled PLA/lev are the number of single-output PLA blocks after weighted functional decomposition is performed. We can see from the table that most of entries for LUT/lev and PLA/lev are the same. Equal value means that there is no p-infeasible node after step 2 is performed. Therefore, we do not need to further decompose any node. Moreover, even for the small number of p-infeasible nodes, the area and delay increased is very small after functional decomposition is performed. This shows that our decomposition algorithm is efficient.

5.2.2 Comparison with TEMPLA in Delay/Depth

This experiment is to compare our algorithm with TEMPLA and FlowMap in delay/depth. Note that FlowMap can produce a circuit with optimum level for LUT-based FPGA. The (10,12,4)-PLA was selected as our target PLA logic block architecture. In this experiment, k is set to 7 in LUT Mapping step.

The value of α to compute the cost function for bound set is set to 0.5 in this experiment. Table 5.4 shows the results of PLA block counts and the circuits depth. The columns labeled OURS is our algorithm, and **TEMPLA** is TEMPLA PLA mapping tool. The result shows for all of circuits, our results have smaller delay. Although TEMPLA is an area minimization algorithm, but on average, our algorithm is 54.7% improvement in delay and with only 24.7% increase in area. As compared to FlowMap, our algorithm produce the same depth.

Table 5.4: Results of comparison with TEMPLA in delay optimization

	(10,12,4)-PLA		
	OURS	TEMPLA	FlowMap
Benchmarks	area/depth		depth
apex6	51/3	49/5	3
dalu	84/4	85/10	4
rot	69/5	47/11	5
x3	57/3	50/6	3
frg2	81/4	54/7	4
i8	115/4	88/7	4
pair	132/5	92/12	5
apex5	104/3	73/6	3
C499	60/4	39/9	4
duke2	48/3	45/10	3
C2670	78/5	52/12	5
Total	879/43	677/95	43
Average	79.9/3.9	61.6/8.64	3.9
Ratio		-29.7%/+54.7%	+0%

5.3 Parameters for (i, p, o)-PLA block

In our experiments, we found that in the bin-packing stage, the number of LUT blocks can be packed is limited by the number of inputs instead of product terms. To have better multiple-output utilization, we should have a PLA block which has a larger number of inputs and a smaller number of product terms. To verify our observation, we conducted experiments on the $(12, 10, 4)$-PLA architecture. Note that the architecture is chosen so that it has the same area as that of $(10, 12, 4)$-PLA (the area of the and-plane in a PLA is computed as $2 \times the\ number\ of\ inputs\ \times the\ number\ of\ product\ term$). The results is shown in Table 5.5. As expected, the results for $(12, 10, 4)$-PLAs is better than those for $(10, 12, 4)$-PLAs.

Table 5.5: Different paremeters for (i, p, o)-PLA block

Benchmarks	(10,12,4)-PLA	(12,10,4)-PLA
apex6	47/7	44/7
dalu	64/9	60/9
rot	48/11	42/11
x3	46/6	46/6
frg2	54/8	46/8
i8	85/10	69/10
pair	98/10	95/10
apex5	71/7	58/7
C499	37/6	52/7
duke2	47/7	37/7
C2670	42/9	42/9
Total	639/90	591/91
Average	58.1/8.2	53.73/8.3

Chapter 6

Conclusions

In this thesis we have presented an efficient algorithm to map a logic network into CPLD. Our algorithm proceeds in two phases: single-output mapping and multiple-output packing. Based on the result presented in [4], a LUT-based mapping algorithm was proposed. We have also studied for a given PLA block architecture how to set the input and product term constraints in mapping phase. The experimental results show that we have 6% improvement in area and 5% in delay as compared to TEMPLA in area minimization, and have 54.7% improvement in delay but only have more 29.8% more area as compared to TEMPLA in delay minimization.

Bibliography

[1] Jason Helge Anderson and Stephen Dean Brown, “Technology Mapping for Large Complex PLDs”, *Design Automation Conference*, pp. 698-703, June 1998.

[2] Jason Cong, Hui Huang, and Xin Yuan, “Technology Mapping for k/m-macrocell Based FPGAs”, *ACM/SIGDA International Symposium on Field Programmable Gate Array*, pp. 51-59, 2000.

[3] Zafar Has an, David Harrison, and Maciej Ciesielski, “A Fast Partitioning Method for PLA-based FPGAs”, *IEEE Design and Test of Computers*, Vol. 9, pp. 34-39, Dec. 1992.

[4] Jack L. Kouloheris and Abbas EL Gammal, “FPGA Performance versus Cell Granularity”, *IEEE Custom Integrated Circuits Conference*, pp. 6.2/1-6.2/4, 1991.

[5] Jack L. Kouloheris and Abbas EL Gammal, “PLA-based FPGA Area versus Cell Granularity”, *IEEE Custom Integrated Circuits Conference*, pp. 4.3.1-4.3.4, 1992.

[6] Jack L. Kouloheris and Abbas EL Gammal, “FPGA Area versus Cell Granularity - Lookup Tables and PLA Cells”, *ACM Workshop on Field Programmable Gate Array*, pp. 9-14 1992.

[7] Jack L. Kouloheris, “Empirical Study of the Effect of Cell Granularity on FPGA Density and Performance”, PhD thesis, Stanford University, 1993.

[8] Jason Cong and Yuzheng Ding, “FlowMap: An Optimal Technology Mapping Algorithm for Delay Optimization in Lookup-Table Based FPGA Designs”, *IEEE Transaction on Computer-Aided Design of Integrated Circuit and Systems*, Vol. 13, No. 1, pp. 1-11, January 1994.

[9] Rajeev Murgai, Youshihito Nishizaki, Narendra Shenoy, Robert K. Nrayton, and Sangiovanni Vincentelli, "Logic Synthesis for Programmable Gate Arrays", *27th ACM/IEEE Design Automation Conference*, pp. 620-625, June 1990.

[10] Robert J. Francis, Jonathan Rose and Zvonko Vranesic, "Chortle-crf: Fast Technology Mapping for Lookup Table-Based FPGAs", *28th ACM/IEEE Design Automation Conference*, pp. 227-233, June 1991.

[11] Jason Cong and Yuzheng Ding, "Combinational Logic Synthesis for LUT Based Field Programmable Gate Arrays", *ACM Transaction on TODAES*, Vol. 1, No. 2, pp. 145-162, April 1996.

[12] Ellen M. Sentovice et al., "SIS: A System for Sequential Circuit Synthesis", *Technical Report UCM/REL M92/41, Electronics Research Lab., Department of Electrical Engineering and Computer Science, University of California, Berkeley*, 1992.

[13] Kuang-Chien Chen, Jason Cong, Yuzheng Ding, Andrew B. Kahng, and Peter Trajmar, "DAG-Map: Graph-Based FPGA technology Mapping for Delay Optimization", *IEEE Design and Test of Computers*, Vol. 9, pp. 7-20, 1992.

[14] Jonathan Rose, Robert J. Francis, David Lewis, and Paul Chow, "Architecture of Field-Programmable Gate Arrays: the effect of logic block functionality on area efficiency", *IEEE Journal of Solid-State Circuits*, Vol. 25, No. 5, pp. 1217-1225, Oct. 1990.

[15] Jonathan Rose, Abbas EL Gammal, and Sangiovanni-Vincentelli, "Architecture of Field-Programmable Gate Arrays", *Proceedings of the IEEE*, Vol. 81, No. 7, pp. 1013-1029, July 1993.