FPGA Technology Mapping Optimization by Rewiring Algorithms

TANG Wai Chung

A Thesis Submitted in Partial Fulfilment
of the Requirements for the Degree of
Master of Philosophy
in
Computer Science and Engineering

June 2005

Abstract of thesis entitled:
FPGA Technology Mapping Optimization by Rewiring Algorithms
Submitted by TANG Wai Chung
for the degree of Master of Philosophy
at The Chinese University of Hong Kong in June 2005

FPGA Technology Mapping is an important design automation problem which affects floorplanning, placement and routing dramatically. Depth-optimal technology mapping algorithms were proposed and produced quality mapping solution for delay minimization. However such algorithms fail to further reduce area consumption by applying powerful logic transformation techniques.

Our work is the first and successful attempt to apply rewiring techniques, state-of-the-art logic transformation techniques, on FPGA technology mapping problem. We focused on reduction on the number of LUTs used while keeping the depth optimality. Our approach is based on a greedy but effective heuristic to choose good alternative wires to transform the network for less LUTs. Experiment results show our approach can effectively reduce 20% of LUTs over initial mapping solutions by FlowMap, FlowSYN and CutMap. Our final optimized mapping solutions are the best among all network-flow-based mapping algorithms and also very comparable to logic-based approach such as BDS-pga.

Our result also provides solid recommendations on the choice of rewiring techniques which can provide flexible trade-offs between optimization effort and runtime constraints. REWIRE allows

complete transformation ability while REWIRE can carry out the optimization process at high throttle. The proposed greedy LUT minimization approach is highly practical and further alleviate burdens on placement and routing for FPGA technology.

論文題目：基於換綫法的現場可編程門陣列的技術映射最優化算法
學生名字：鄧偉宗
學位：計算機科學及工程學哲學碩士
日期：二零零五年六月

現場可編程門陣列的技術映射是一個對布圖規劃、布局與布綫具重大影響的重要設計自動化問題。 縱然最優選深度技術映射算法為延遲最小化提供了極好質量的映射解答，它們並沒有進一步以強而有力的邏輯變換技術來減少芯片面積消耗量。

我們成功嘗試使用頂尖的換綫法技術在現場可編程門陣列的技術映射問題上。我們集中於保持深度最優性的同時減少硬件查找表的數目。我們的方法根據一貪婪但有效的探索法來選擇良好的導線來變換網路，以減少硬件查找表的數目。實驗結果展示這方法可有效地在 FlowMap，FlowSYN 及 CutMap 最初的映射解答中減少兩成硬件查找表。我們的優化映射解答已是在所有網路流動根據映射算法之中最佳的，同時還可跟邏輯根據方法如 BDS-pga 等再同價比較。

我們的結果並且提供堅實的建議來提供在優化努力和運行時間限制之間權宜換綫法技術的選擇：REWIRE 允許完整變換能力而 RAMFIRE 能以高速執行整個優化過程。我們所提出的貪婪硬件查找表最少化方法既具高度實用性，且能進一步緩和布圖規劃與布綫的負擔。

Acknowledgement

I would like to thank my labmates for inspiring discussions on different rewiring techniques and they provide me a lot of insight on this work.

Also I would like acknowledge UCLA VLSI CAD lab for providing the source code of FlowMap, FlowSYN and CutMap.

For Winky, my family and all my friends.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Rewiring techniques had been developed by EDA research community in the past decade. Those techniques are able to find out alternate wires for a given target wire, and allow logic transformation on the network without changing any behaviours of the circuit at primary outputs. The techniques were applied successfully in various design automation problem such as circuit partitioning and proves the power of the techniques for efficient and flexible logic synthesis.

We studied FPGA technology mapping problem and reviewed several depth-optimal mapping algorithms. Statistics showed that the logic block components in the FPGA is not fully utilized by current algorithms. Moreover, logic transformation is not in-

cluded or fully used in these algorithms. This suggests us to investigate the feasibility of adding in rewiring techniques into technology mapping algorithms.

The thesis is organized as follows: chapter 2 introduces rewiring algorithms and compares their abilities and speed; chapter 3 reviews network-flow-based technology mapping algorithms which guarantee depth-optimal mapping solutions; chapter 4 explains our approach and discusses our extensive experiments. This chapter also includes our analysis on the experimental results; chapter 5 give the final conclusion.

Chapter 2

Rewiring Algorithms

Redundancy addition and removal (RAR), or rewiring, can modify the structure of a logic circuit by adding redundant wire(s) or gate(s) to force removable redundancies on another part of the circuit, and throughout the whole process the function implemented by the circuit is remained unchanged. We call the wire which is assigned to be removed the *target wire* and the new wire to be added to the circuit the *alternate wire*. Rewiring algorithms are referring to the techniques to identify any target / alternate wire pair for a given circuit. We denote a wire by a duple of gates - (s, d) refers to a wire from gate s to gate d.

Consider the example circuit shown in figure 2.1, where the target wire is $(g1, g6)$. We can see that the circuit functionality

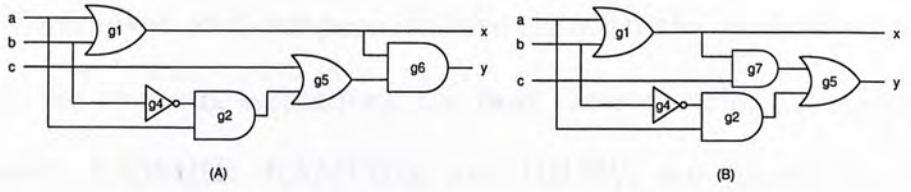

Figure 2.1: (A)Original Circuit (B) Modified Circuit

will be maintained by:

1. Replacing $(c, g5)$ with the output of an AND gate whose inputs are $g1$ and c respectively.

2. Removing $(g1, g6)$.

The modified circuit is shown in figure 2.1. Clearly, the output function y remains to be $(a + b)(a\bar{b} + c) \equiv (a + b)c + a\bar{b}$ after rewiring.

Although rewiring was first designed for logic minimization, the technique is prove to be powerful in areas like post-layout timing optimization [13], circuit partitioning [7] [15] and FPGA routing [4]. From all the examples, we can understand that rewiring algorithms provide flexible and powerful logic transformation which can be used to optimized circuits' performance for different goals. This strongly motivates us to apply rewiring techniques in optimization for FPGA technology mapping.

To analyze and compare extensively with the performance of different rewiring techniques, the best three rewiring techniques, namely REWIRE, RAMFIRE and GBAW, are chosen for our purpose. The techniques were applied and tested with our technology mapping optimization in order to characterize their abilities for our problem. They are briefly introduced in the following sections.

2.1 REWIRE

REWIRE [5] is based on the idea of MA (Mandatory Assignments). These are the assignments that are required in order for a test of a fault to exist. If the set of MA for a fault is inconsistent, no test is possible for that fault. REWIRE tries to add wires that force the set of MA for the stuck-at fault test of the target wire to become inconsistent, that is, after the addition of the wire, the target wire's stuck-at fault will become untestable and the wire will become redundant. It then ascertains the added wire itself is also redundant so its addition will not change the circuit functionality. There are fast filters in REWIRE to screen out those wires that cannot become redundant so that less redundancy tests are needed.

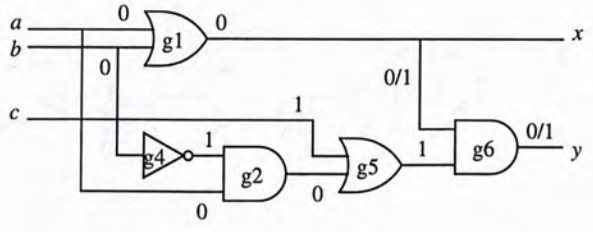

Figure 2.2: MA values for s-t-1 fault on $(g1, g6)$

To illustrate the idea of mandatory assignments, we consider again the circuit shown in 2.1A. If we perform a stuck-at-1 fault on the wire $(g1, g6)$, the values of MA are shown on 2.2. We can see that if we insert an AND gate with inputs $g1$ and c in front of $g5$ the MA values become inconsistent, making the stuck-at-fault not testable. In this way, the target wire is hence removable after the addition of the alternate wire. REWIRE identifies alternate wires for different target wires based on this simple idea of creating conflicts in MA.

REWIRE is found to be the most powerful rewiring algorithm which can find the largest number of target / alternate wire pairs. However, it consumes large amount of computation time due high computational complexity in doing logic propagation and justification.

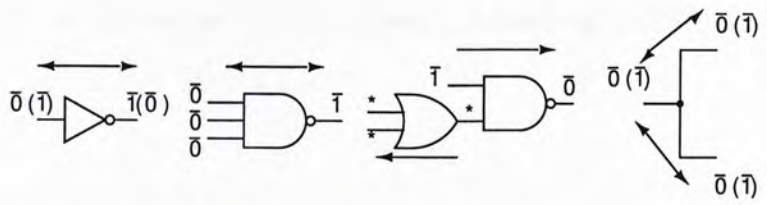

Figure 2.3: Unobservability and Uncontrollability Propagation Rules (Basic)

2.2 RAMFIRE

RAMFIRE[2][3] is another ATPG[1]-based rewiring algorithms. It utilizes a polynomial-time redundancy identification technique called FIRE[12] to reduce unnecessary redundancy checking. FIRE applied the concept of uncontrollability and unobservability directly on implications. A signal is assigned $\bar{1}(\bar{0})$ if we cannot control the signal to be 1(0), and a signal will be marked as $*$ when any faults on this signal line is not able to be propagated to primary output for observations. During the implications, uncontrollability will propagate based rules on gate types and unobservability will propagate mainly backwards. The propagation rules from [2] is reproduced in figure 2.3 for reference.

The algorithm first starts implication of $\bar{1}$ and $\bar{0}$ on the target wire and get two sets of uncontrollability and unobservability values $S(\bar{1})$ and $S(\bar{0})$. And the alternate wire(s) will be identified

[1] automatic test pattern generation

checking the potential of redundancy based on the following definition:

A gate is *potentially redundant* in S if

- the gate output is assigned unobservable ($*$) in S, or
- the gate output has a MA of $0(1)$ in stuck-at-fault test and a value of $\bar{0}(\bar{1})$ in S.

given that the gate is in the transitive fanout cone of the target wire.

Experimentally, RAMFIRE has a speed up of 20 times over REWIRE but REWIRE can find 70% more alternative wires than RAMFIRE.

2.3 GBAW

GBAW [16] does not apply any logic implications. What it depends on is a set of graph configurations, which are called "patterns". Patterns are pre-defined graph representations of subcircuits which contains alternative wires. An example pattern is shown in figure 2.6. The target wire to the NOR gate can be re-

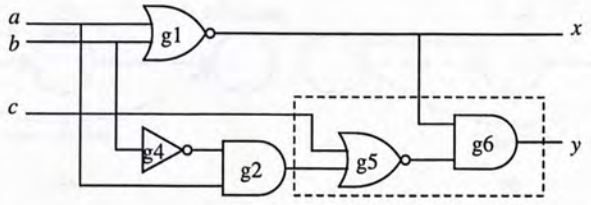

Figure 2.4: Circuit Containing a GBAW LOCAL 13 Pattern

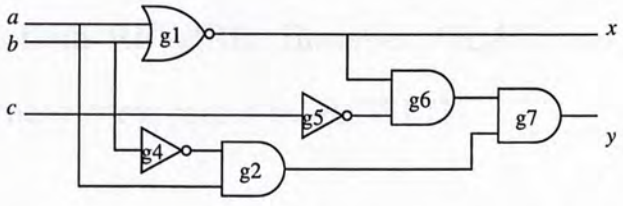

Figure 2.5: Resultant Circuit After Rewiring based on GBAW LOCAL 13 Pattern

placed by the alternative wire to the AND or NAND gate. Figure 2.4 shows a circuit containing a pattern *local 13* (the pattern is embraced with the dashed box). Thus when GBAW finds alternative wires on this circuit, *local 13* pattern will be matched and then we can identify $(c, g5)$ or $(g2, g5)$ as the target wire. Suppose we replace $(g2, g5)$ by a new alternative wire $(g2, g6)$ (equivalent adding a new AND gate after $g6$), we get a logically equivalent circuit shown in figure 2.5.

GBAW finds alternative wires by performing pattern matching on the circuit with the library of patterns. GBAW is able to find alternate wires with high speed, and on average it is around 150

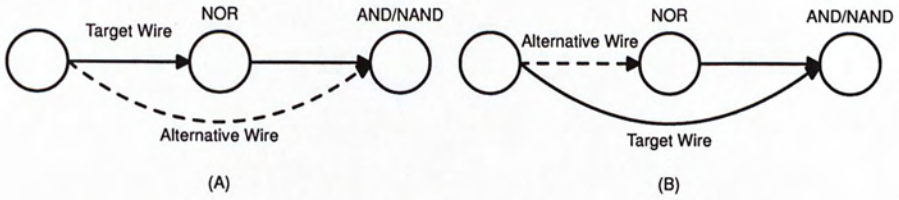

Figure 2.6: Local 13 Pattern(A) and its Backward Wire(B)

times faster than REWIRE. However, GBAW can only locate 23% of alternate wires found by REWIRE.

□ **End of chapter.**

Chapter 3

FPGA Technology Mapping

Field Programmable Gate Arrays (FPGA) is a popular technology for digital designers, especially in Application Specific Integrated Circuit (ASIC) area. The reuseablility and programmability of FPGA dramatically reduce the design turn-around time and development cost. FPGA is also widely used in benchmark testing and prototyping of digital circuits.

One of the most important components in FPGA are logic block, which is responsible for implementing all the combinatorial functions of the circuit. In conventional FPGA, the logic block is implemented by lookup tables (LUT), which is found to be an area-efficient method for Boolean function implementations on VLSI. Usually, fixed size LUTs are used among the whole

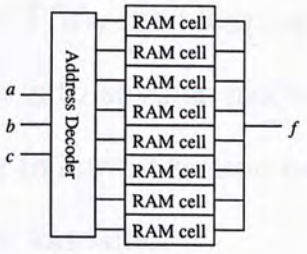

Figure 3.1: Schematic of a SRAM-based 3-LUT

FPGA chip and the size of every LUT is denoted by k, which is commonly chosen to be 4 or 5. Every LUT is implemented by 2^k memory cells with k-bit address decoder. Any inputs to a Boolean function will be taken as an address to read corresponding bit pre-loaded inside the memory cell. Therefore a k-LUT can implement any k-variable Boolean functions by saving the truth of the function inside the list of memory cells. Figure 3.1 shows a possible structure of a 3-LUT[1].

During the design automation process, after a circuit is synthesized by technology-independent optimization algorithms, we have to carry out a technology mapping to assign functions to be saved on the LUTs - when the circuit consists of small input gates, the mapping process needs to collect and group the gates to together to fit into a LUT; when the circuit is originally in com-

[1] *Wired-OR*: The outputs of the SRAM cells are connected through a shared BUS

plex form, for instance, PLA, the mapping process is required to break down the circuit into smaller pieces which can be implemented by LUTs. The following section explains the technology mapping problem more formally.

3.1 Problem Definition

A circuit is modeled as a Boolean Network. There is a set of nodes PI representing the primary inputs (PI) and another set of nodes PO representing the primary outputs (PO). All other nodes in the network is called *internal* and these nodes are associated with specific functions. The function type can be simple (AND, OR, NOT, XOR) or complex. Every wire in the circuit is represented by an edge between two nodes. All incoming edges to a node is called *fanin* of this node and all outgoing edges are called *fanout*; Nodes in PI has only fanouts while nodes in PO has only fanins. If the in-degrees of all nodes are less than or equal to k, the network is k-bounded. Clearly k-bounded network can be implemented by a FPGA using k-LUTs as logic block. A simple Boolean network and its corresponding circuit are shown in figure 3.2

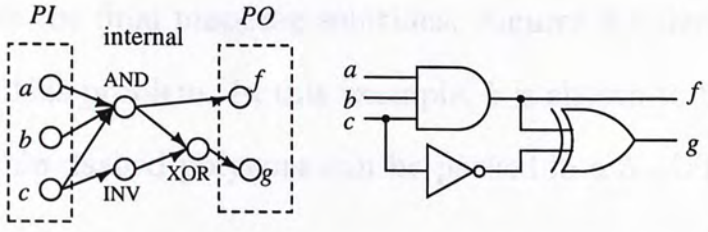

Figure 3.2: Simple Boolean Network & its Corresponding Circuit

The FPGA technology mapping problem is formulated as follows:

- INPUTS: A Boolean Network N, LUT size k, delay model D
- OUTPUT: A k-bounded network N'
- OBJECTIVES:
  1. Minimize the number of LUTs used to map the circuit, denoted by $|N'|$.
  2. Minimize the delay of N', $d(N')$, according to D.

The original network can be k-bounded or k-unbounded. For the latter case, we have to decompose the k-unbounded network into a k-bounded one before we can process it with the technology mapping algorithms. This process is usually referred as *gate decomposition*. We should notice that gate decomposition will alter the initial network structure supplied to the mapping algorithms,

thus affect the final mapping solutions. Figures 3.3 shows an example on this problem. In this example, k is chosen to be 3. The gates within dashed polygons can be packed in a 3-LUT.

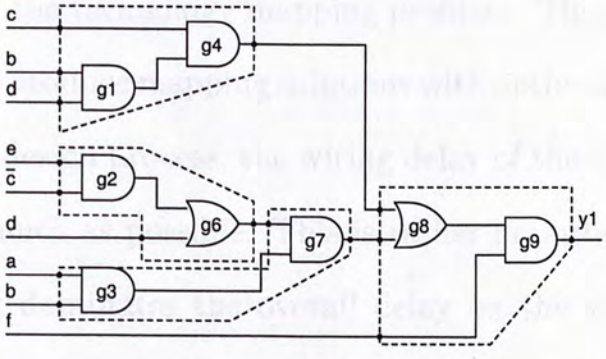

Figure 3.3: FPGA mapping example

For a quick but accurate delay estimation, unit delay model can be used to estimate the overall delay of the circuit. In unit delay model, every node is assumed to have delay 1 over the signal propagations, and therefore the total delay of the circuit is determined by the depth of the circuit. In other words, the number of nodes on the longest signal path (or critical path) is the total delay under this model. Depth of the network can be found by doing a depth first search (DFS).

3.2 Network-flow-based Algorithms for FPGA Technology Mapping

In this section, we will introduce three network-flow-based algorithms for the technology mapping problem. These algorithms guarantee to produce mapping solutions with optimal depth. Therefore in later design process, the wiring delay of the circuit can be reduced as much as possible. This is rather important when the wiring delay dominates the overall delay on the whole chip in deep sub-micron technology in VLSI. In our work, these three algorithms are used and compared for the best result.

3.2.1 FlowMap

FlowMap[9] is the first depth-optimal technology mapping algorithm developed. The algorithm will first apply DMIG[6] to decompose the network into network composed of small gates. The author of FlowMap believes small gates can be packed and grouped more efficiently by their algorithms than large input gates. And experimentally they showed that the depth of the mapped network is the smallest when the original was first decomposed into 2-input gates.

After gate decomposition, the algorithm enters the *labeling phase.* The algorithm calculate a label $l(t)$ for every node t in topological order. The label $l(t)$ gives the minimum depth of any mapping solution of the subnetwork rooted at node t, denoted by N_t. Moreover, $l(t)$ is either equal to the maximum label p of the nodes in fanin of t or one more than the maximum label. FlowMap first collapses all nodes with label p in N_t to get a new network N_t', then it continues with computation of the maximum volume min-cut of N_t' by the classic network flow technique. If the cut size is less than or equal to k, the label $l(t)$ is assigned to be p, otherwise $l(t) = p+1$, indicating a new LUT is used to map N_t.

After label calculation, FlowMap starts k-LUT generation with a list of PO nodes. It iteratively takes a non-PI nodes on list and generate a LUT to implement the function for all the nodes with the same label. The fanins to this newly generated LUT is then put on the list.

To illustrate the label calculation we show the network for the circuit in previous example in figure 3.4. For simplicity, we take

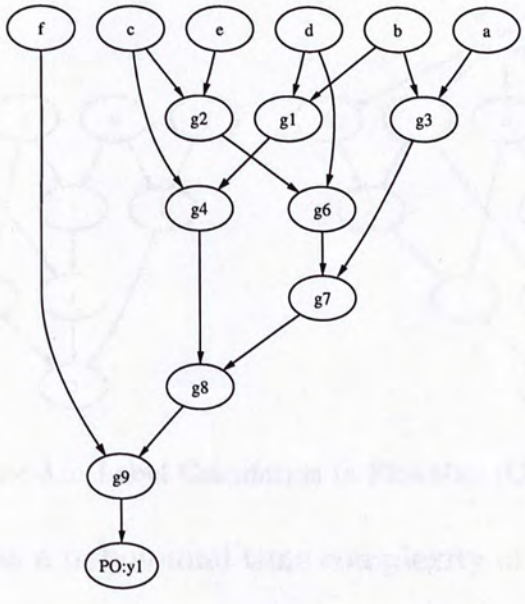

Figure 3.4: Label Calculation in FlowMap

$k = 3$. Suppose during the label phase we need to compute the label for node $g8$ with $p = 2$ ($l(g7) = 2$). Thus we collapse node $g7$ with $g8$ together and consider this collapsed node as the sink node. After addition of a dummy source node connecting to all PI nodes, we find a minimum cut on the network by network flow technique. Figure 3.5 shows the collapsed network and the graph for flow calculation. The min-cut simply separates the sink node with all other nodes, which implies that nodes $g7$ and $g8$ can be collected together and implemented by a 3-LUT. Since the cut size equals 3, the label of node $g8$ is 2, same as that of $g7$.

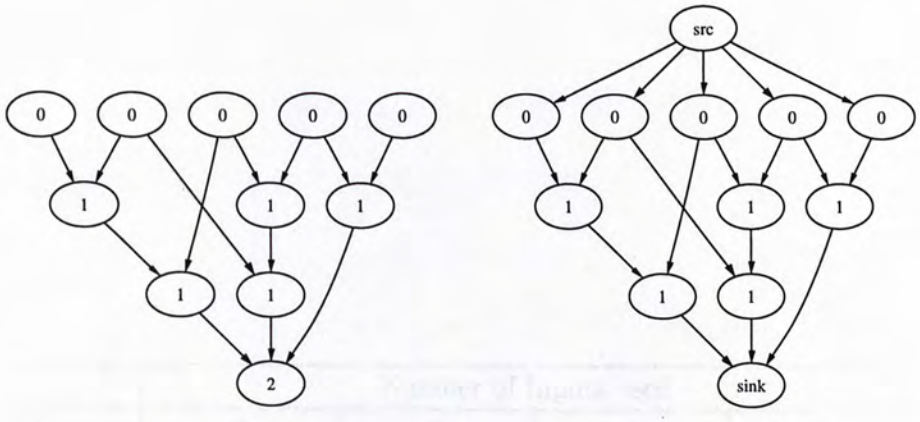

Figure 3.5: Label Calculation in FlowMap (Cont')

FlowMap has a polynomial time complexity of $O(kmn)$ where n and m are the number of nodes and the number of edges in N. Therefore the algorithm is extremely fast even for large circuits with thousands of gates. However, during our experiments we found that the mapping solutions by FlowMap does not guarantee high utilization of LUTs, despite its depth optimality. In other words, many k-LUTs do not have all k inputs used, and this situation goes worse when k increases. One prime reason is that when k increases, the utilization drops since the algorithm finds it harder and harder to push in nodes in an old LUT when no modification is allowed during mapping. Table 3.1 shows the statistics collected.

	Number of Inputs used			
Circuit	2	3	4	5
3-LUT				
alu2	120	146	0	0
alu4	186	290	0	0
des	1528	982	0	0
Total	1834 (56.4%)	1418 (43.6%)	0	0
4-LUT				
alu2	68	59	87	0
alu4	108	107	179	0
des	346	292	1076	0
Total	552 (22.5%)	458 (19.7%)	1342 (57.8%)	0
5-LUT				
alu2	37	23	37	77
alu4	77	45	81	110
des	361	202	335	641
Total	475 (23.4%)	270 (13.3%)	453 (22.4%)	828 (40.9%)

Table 3.1: Relationship between k and LUT Utilization

3.2.2 FlowSYN

FlowMap does not take the logic information stored into consideration during the technology mapping process. It is obviously true by re-synthesis of the network the delay and the number of LUTs can be further reduced. FlowSYN[8] attempts to augment FlowMap with logic reconstruction technique. When FlowSYN finds a min-cut with size more than k, instead of incrementing the label by 1, it tries to re-synthesize the subnetwork based on functional decompositions. The principal decomposition form is $f(x_1, x_2, \ldots, x_r) = f'(g_1, \ldots, g_j, x_{k+1}, \ldots, x_r)$, where g_i for $i = 1$ to j are functions for $x_1, x_2, \ldots, x_k$, $j < k$ and $r > k$. If such decomposition is possible, all g_i functions can be implemented by k-LUT and resultant number of inputs from f to f' is reduced by 1. This follows to recursively decompose the function so that all subfunctions are fitted into k-LUTs. After all, such recursive decomposition can allow the label to remain p and thus there is higher chance to have a smaller depth in the final mapping solution.

The functional decompositions are carried out using ordered binary decision diagrams (OBDD) [1] efficiently. Nevertheless,

more memory usage and computation time are expected in FlowSYN.

3.2.3 CutMap

CutMap[10] introduced two important concepts - depth relaxation on non-critical nodes and node cost functions based on maximum fanout free cone.

CutMap relaxes the rule to assign depth-optimal labels on nodes which are not on the critical path in order to gain room for area minimization. Instead it keeps track on the difference between the current label and the depth-optimal label. During label calculation for non-critical nodes, the node collapse originated in FlowMap for depth optimization is not used, and this is replaced by the minimum cost k-feasible cut computation in CutMap.

The cost function for every node is determined by the maximum fanout free cone (MFFC) it belongs to. The root node of every MFFC is assigned the cost zero since they are more likely to be implemented by a single LUT and they have high number of fanouts. PI and PO nodes have cost zero and other nodes have cost one. Consequently this avoids cuts along the MFFCs and prevents more nodes to be implemented by unnecessary LUTs.

CutMap preserves the depth optimality of FlowMap but at the same time minimize the number of LUTs used through min-cost cut computation. Clearly, the computation for such cuts dominates the runtime of the algorithm. Although the algorithm is speed-up by careful case handling, its time complexity is $O(2kmn^{k/2+1})$. Taking $k = 4$ or 5, CutMap runs slowly than FlowMap by a factor of n^2.

□ **End of chapter.**

Chapter 4

LUT Minimization by Rewiring

From our discussion on several network-flow-based FPGA technology mapping algorithms, we can see that most of the logic information available in the network is not utilized for reduction of number of LUTs in the mapping solution. Though FlowSYN employs functional decomposition inside the algorithm, its main goal is still on depth optimization. On the other hand, CutMap tries to reduce the number of LUTs solely by taking better cuts along non-critical paths on the network and the algorithm does not consider the network contains digital logic information inside.

Therefore we explored the possibility of linking logic transformation with technology mapping together and work out a coherent collaboration so that we can on one hand keep the depth

optimality provided by FlowMap / FlowSYN / CutMap, but at the same time transform the network to allow less LUTs used in the mapping solution. We chose network-flow-based mapping algorithms since they provide optimal-depth solutions and we can thus focus on LUT reduction using logic transformation.

To illustrate the idea, we first present an example of LUT reduction by rewiring technique. Consider the sub-circuit shown in figure 4.1. Suppose initially we get 4 LUTs rooted at $g1$, $g3$, $g5$ and $g6$ respecitively, as bounded by dotted lines. More here we assume that $l(g1)$ equals $l(g5)$. If we find an alternate wire $(g2, g5)$ for the target wire $(g4, g5)$, we can allow more efficient packing with 3 LUTs only, and the overall depth of the circuit is unchanged. The final mapping is should in figure 4.2.

Rewiring was chosen to provide transformations available to the network. The first reason for this choice is that rewiring is powerful in which it is proved that any logic transformation can be archived by apply rewiring. Therefore rewiring technique provides us with high reachability in good starting network for the mapping algorithms. The second reason is that rewiring tech-

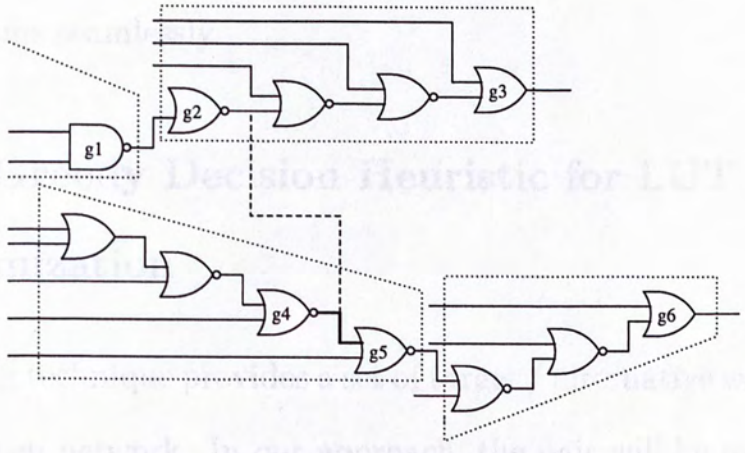

Figure 4.1: Initial Mapping Solution

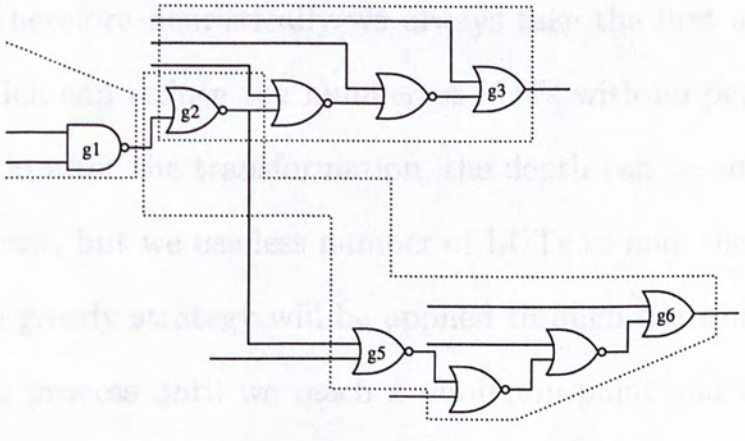

Figure 4.2: Final Mapping Solution after transformation

niques usually work better on network decomposed in small gates so we can link rewiring technique with the technology mapping algorithms seamlessly.

4.1 Greedy Decision Heuristic for LUT Minimization

Rewiring technique provides a set of target / alternative wire pairs for a given network. In our approach, the pair will be evaluated directly using the mapping algorithms to determine the number of LUTs and the depth after the transformation applied to the network. Even the evaluation can be completed in polynomial time[1], we would like to make the trial evaluation as least as possible. Therefore heuristically we always take the first alternate wire which can reduce the number of LUTs with no penalty on depth. So after the transformation, the depth can be smaller or at the least, but we use less number of LUTs to map the circuit. And the greedy strategy will be applied through the whole optimization process until we reach a minimum point and no more improvement can be done by transformation on the network. The

[1] when $k = 4$ or 5, FlowMap runs at $O(kmn)$ and FlowSYN and CutMap runs at $O(kmn^3)$ approximately.

Algorithm *ReMap*
Input: Boolean network N
Output: LUT minimization network N'
1. finding a set of alternate wires W of N
2. **for** each AW w in W
3. **do** transform N according to w
4. $N_f \leftarrow flowmap(N)$
5. **if** $lut(N_f) < lut(N)$ **and** $d(N_f) \leq d(N)$
6. **then** $N \leftarrow N_f$
7. goto step 1
8. /* no more possible reduction */
9. **return** N

Figure 4.3: Algorithm Flow of REMAP: Greedy LUT Minimization

algorithm is outlined in figure 4.3a.

We conducted a series of experiments in linking the rewiring technique with the technology mapping algorithms based on our greedy decision heuristic in order to verify the usefulness of this approach. REWIRE, RAMFIRE and GBAW are chosen to work with FlowMap, FlowSYN and CutMap. The experimental results and analysis are given in the following section.

4.2 Experimental Result

We conducted all the following experiments with 12 small to medium sized MCNC benchmark circuit on a personal computer with 2.4-GHz CPU and 1028 MB memory. The experiments were

written in C language with SIS[11] library on Fedora Core 3 Linux platform.

We started with REWIRE on the three map algorithms since REWIRE should be the most powerful which enables us to full understand the extent of effectiveness of the approach. Table 4.1, 4.2 and 4.3 show the initial depth, the initial number of LUTs, the new depth, the optimized number of LUTs, the reduction percentage and finally the execution time measured in minutes.

We can see that REWIRE can readily reduce the number of LUTs used for mapping into LUTs. The overall percentage reduction is 22.7%, 24% and 22.1% on FlowMap, FlowSYN and CutMap respectively. In general the depth of the circuit remains the same after the optimization. This can explain that REWIRE does not length the depth very much during the transformation for less LUTs and actually we have margins on each level of LUTs to accommodate a few more level of small gates so that the final mapping depth is kept unchanged. For small-sized circuits, REWIRE completed optimization within one to three minutes while for medium-sized circuits the time taken will be around

two to three hours on FlowMap and FlowSYN. The runtime will be longer on CutMap due to higher complexity in evaluation, and it takes around six to eight hours to optimize a medium-sized circuits.

Circuit	Init. d	Init. n	New d	Opt. n	Red. (%)	CPU
comp	4	27	4	26	3.7	0.4
f51m	3	28	3	23	17.9	0.4
5xp1	3	30	3	26	13.3	0.8
pcler8	3	47	3	35	25.5	0.2
b9_n2	3	47	3	38	19.1	0.3
ttt2	3	62	3	42	32.3	3.6
term1	4	77	4	54	29.9	10.6
C880	8	147	7	106	27.9	20.5
alu2	9	174	9	120	31.0	261.0
duke2	4	186	4	139	25.3	114.6
misex3	7	225	7	180	20.0	234.1
x3	5	252	5	217	13.9	63.7
Total		1302		1006	22.7	710.2

Table 4.1: Experimental Results: REWIRE on FlowMap

Then we proceeded with experiments with RAMFIRE and results are shown on table 4.4, 4.5 and 4.6. The reduction brought by RAMFIRE is quite satisfactory even when compared to results using REWIRE - the reduction percentages are 19.3%, 16.7% and 15.4% respectively and the ratios to results on REWIRE are 84.8%, 69.5% and 69.9%. However RAMFIRE used only one-fifth

Circuit	Init. d	Init. n	New d	Opt. n	Red. (%)	CPU
comp	4	27	4	26	3.7	0.4
f51m	3	25	3	16	36.0	1.0
5xp1	2	21	2	18	14.3	1.0
pcler8	3	47	3	35	25.5	0.2
b9_n2	3	47	3	39	17.0	0.3
ttt2	3	59	3	41	30.5	4.0
term1	4	76	4	52	31.6	10.7
C880	8	146	7	106	27.4	23.3
alu2	9	164	9	114	30.5	316.8
duke2	4	187	4	139	25.7	121.1
misex3	7	221	7	166	24.9	258.9
x3	5	253	5	216	14.6	59.1
Total		1273		968	24.0	796.8

Table 4.2: Experimental Results: REWIRE on FlowSYN

Circuit	Init. d	Init. n	New d	Opt. n	Red. (%)	CPU
comp	4	22	4	21	4.5	1.7
f51m	3	28	3	22	21.4	1.3
5xp1	3	27	3	24	11.1	1.2
pcler8	3	34	3	29	14.7	0.2
b9_n2	3	42	3	37	11.9	0.8
ttt2	3	48	3	42	12.5	2.7
term1	4	62	4	49	21.0	21.6
C880	8	127	7	92	27.6	171.8
alu2	9	150	9	118	21.3	1450.2
duke2	4	153	4	120	21.6	334.4
misex3	8	204	7	146	28.4	3573.6
x3	5	231	5	179	22.5	140.3
Total		1129		879	22.1	5699.8

Table 4.3: Experimental Results: REWIRE on CutMap

of CPU time to complete the set of experiments. From this we believe RAMFIRE is quite promising to produce substantial LUT reduction even if short amount of CPU time is allowed.

Circuit	Init. d	Init. n	New d	Opt. n	Red. (%)	CPU
comp	4	27	4	26	3.7	0.1
f51m	3	28	3	26	7.1	0.1
5xp1	3	30	3	27	10.0	0.1
pcler8	3	47	3	35	25.5	0.0
b9_n2	3	47	3	39	17.0	0.1
ttt2	3	62	3	44	29.0	0.9
term1	4	77	4	59	23.4	1.7
C880	8	147	7	109	25.9	8.4
alu2	9	174	9	136	21.8	35.4
duke2	4	186	4	148	20.4	10.7
misex3	7	225	7	180	20.0	55.2
x3	5	252	5	222	11.9	21.8
Total		1302		1051	19.3	134.5

Table 4.4: Experimental Results: RAMFIRE on FlowMap

Finally we repeated the experiments with GBAW, the fastest rewiring technique. Obviously GBAW can finish the optimization in extremely short time, yet we need to analyze the power of GBAW in specifically reduction for LUTs. Table 4.7, 4.8 and 4.9 show the experimental results for our analysis. Here we show the CPU time in seconds since GBAW finished every circuit with 10 seconds. Nevertheless, the alternative wires found by GBAW is

Circuit	Init. d	Init. n	New d	Opt. n	Red. (%)	CPU
comp	4	27	4	26	3.7	0.1
f51m	3	25	3	16	36.0	0.2
5xp1	2	21	2	18	14.3	0.2
pcler8	3	47	3	35	25.5	0.0
b9_n2	3	47	3	39	17.0	0.1
ttt2	3	59	3	42	28.8	1.0
term1	4	76	4	60	21.1	1.9
C880	8	146	7	109	25.3	9.6
alu2	9	164	9	133	18.9	41.9
duke2	4	187	4	187	0.0	1.0
misex3	7	221	7	173	21.7	64.6
x3	5	253	5	223	11.9	18.9
Total		1273		1061	16.7	139.5

Table 4.5: Experimental Results: RAMFIRE on FlowSYN

Circuit	Init. d	Init. n	New d	Opt. n	Red. (%)	CPU
comp	4	22	4	21	4.5	0.6
f51m	3	28	3	27	3.6	0.3
5xp1	3	27	3	26	3.7	0.4
pcler8	3	34	3	29	14.7	0.1
b9_n2	3	42	3	36	14.3	0.5
ttt2	3	48	3	43	10.4	0.8
term1	4	62	4	59	4.8	3.8
C880	8	127	7	96	24.4	141.2
alu2	9	150	9	116	22.7	403.1
duke2	4	153	4	125	18.3	64.6
misex3	8	204	8	159	22.1	750.9
x3	5	231	5	217	6.1	34.5
Total		1128		954	15.4	1400.8

Table 4.6: Experimental Results: RAMFIRE on CutMap

not very useful in LUT reduction, and we can only reduce 1.8%, 1.6% and 1.2% of LUTs on FlowMap, FlowSYN and CutMap by GBAW respectively. The problem is especially serious when GBAW works on small-sized circuits, in most cases, no reduction is found.

Circuit	Init. d	Init. n	New d	Opt. n	Red. (%)	CPU(s)
comp	4	27	4	27	0.0	0.57
f51m	3	28	3	28	0.0	0.42
5xp1	3	30	3	29	3.3	0.1
pcler8	3	47	3	47	0.0	0.16
b9_n2	3	47	3	47	0.0	0.21
ttt2	3	62	3	61	1.6	0.07
term1	4	77	4	74	3.9	0.44
C880	8	147	8	146	0.7	0.4
alu2	9	174	9	171	1.7	0.81
duke2	4	186	4	179	3.8	0.8
misex3	7	225	7	221	1.8	1.42
x3	5	252	5	249	1.2	4.19
Total		1302		1279	1.8	9.6

Table 4.7: Experimental Results: GBAW on FlowMap

To sum up, we found that REWIRE and RAMFIRE is a strong optimization tool for the technology mapping algorithms under experiments. They are capable to reduce at least 15% of LUTs used by all three mapping algorithms. The final mapping solution is the best when we take REWIRE on CutMap and it uses only

Circuit	Init. d	Init. n	New d	Opt. n	Red. (%)	CPU(s)
comp	4	27	4	27	0.0	0.63
f51m	3	25	3	18	28.0	0.14
5xp1	2	21	2	21	0.0	0.58
pcler8	3	47	3	47	0.0	0.18
b9_n2	3	47	3	47	0.0	0.22
ttt2	3	59	3	56	5.1	0.08
term1	4	76	4	74	2.6	1.82
C880	8	146	8	145	0.7	6.14
alu2	9	164	9	164	0.0	8.55
duke2	4	187	4	187	0.0	5.54
misex3	7	221	7	217	1.8	6.07
x3	5	253	5	250	1.2	4.67
Total		1273		1253	1.6	34.6

Table 4.8: Experimental Results: GBAW on FlowSYN

Circuit	Init. d	Init. n	New d	Opt. n	Red. (%)	CPU(s)
comp	4	22	4	22	0.0	2.89
f51m	3	28	3	28	0.0	1.52
5xp1	3	27	3	27	0.0	2.13
pcler8	3	34	3	33	2.9	0.09
b9_n2	3	42	3	42	0.0	0.82
ttt2	3	48	3	48	0.0	5.26
term1	4	62	4	60	3.2	6.46
C880	8	127	8	127	0.0	74.46
alu2	9	150	9	150	0.0	105.32
duke2	4	153	4	148	3.3	11.46
misex3	8	204	7	201	1.5	34.9
x3	5	231	5	229	0.9	14.25
Total		1128		1115	1.2	259.6

Table 4.9: Experimental Results: GBAW on CutMap

879 LUTs to map all 12 circuits, when compared to 1302 LUTs used by FlowMap solely, we have already a 33% improvement. Such an improvement can readily ease floorplan, placement and routing in later design stages and also allow the whole design process to be completed in short time.

On the other hand, we do not recommend the use of GBAW for the mapping optimization problem since the alternative wires found by GBAW are more type-limited when compared to ATPG-based rewiring technique (REWIRE, RAMFIRE). Despite its high speed, the reduction in LUTs is not very significant.

Besides network-flow-based mapping algorithm, logic-based algorithms are also proposed in literature. One of the logic-based algorithms is BDS-pga introduced in [14]. The algorithm first decomposes the network using OBDD and finalized the mapping with FlowMap. Table 4.10 shows the comparison between our approach using REWIRE on CutMap and BDS-pga. Our work is comparable with BDS-pga with better results in several circuits. Yet BDS-pga can perform better in XOR-intensive circuits like alu2 or comp.

	REWIRE + CutMap		BDS-pga + FlowMap	
Circuit	new d	opt. n	delay	LUTs
comp	4	21	3	14
b9_n2	3	37	3	40
C880	7	92	8	103
alu2	9	118	4	41
duke2	4	120	8	173
rot	7	213	10	223

Table 4.10: Comparison between REWIRE on CutMap and BDS-pga

□ **End of chapter.**

Chapter 5

Conclusion

We proposed a new approach to optimize FPGA technology mapping by using rewiring techniques. Our approach can effectively reduce the number of LUTs by 15 - 18% on average and it works even better with large circuits where a lot of alternative wires can be found. Our result is already better than all network-flow-based mapping algorithms found in literature and also very comparable to logic-based mapping algorithms.

We also conducted experiments to verify the power of various rewiring algorithms on the LUT minimization problem. We found that REWIRE is the most powerful technique which can on average reduce more than 20% of LUTs in initial mapping solution. However, when design time is a concern, we recommend the use of

RAMFIRE since it can reduce 15% of LUTs in one-fifth of CPU time used by REWIRE.

The thesis reveals our preliminary effort in apply logic transformation with technology mapping and the result is clearly encouraging. The LUT reduction is advantageous to all later physical design process after technology mapping. Therefore we believe this coherent approach is a very good and challenging research area and we truly hope more effort will be made in the future.

□ **End of chapter.**

Bibliography

[1] R. E. Bryant. Graph-based algorithms for Boolean function manipulation. *IEEE Trans. Comput.*, C-35(8):677–691, Aug. 1986.

[2] C. W. Chang and M. Marek-Sadowska. Single-pass redundancy addition and removal. In *ICCAD*, pages 606–609, 2001.

[3] C. W. Chang and M. Marek-Sadowska. Who are the alternative wires in your neighborhood (alternative wires identification without search). In *Great Lakes Symposium on VLSI*, pages 103–108, 2001.

[4] S. C. Chang, K. T. Cheng, N. S. Woo, and M. Marek-Sadowska. Post-layout logic restructuring using alternative wires. *IEEE Trans. Computer-Aided Design*, 6:587–596, June 1997.

[5] S. C. Chang, L. P. P. P. van Ginneken, and M. Marek-Sadowska. Fast boolean optimization by rewiring. In *Proc. IEEE Int'l Conf. on Computer-Aided Design*, pages 262–269, Nov. 1996.

[6] K. C. Chen, J. Cong, Y. Ding, A. B. Kahng, and P. Trajmar. Dag-map: Graph-based fpga technology mapping for delay minimization. pages 7–20, Sept. 1992.

[7] D. I. Cheng, C. C. Lin, and M. Marek-Sadowska. Circuit partitioning with logic perturbation. In *Proc. Int. Conf. on Computer Aided Design*, pages 650–655, Nov. 1995.

[8] J. Cong and Y. Ding. Beyond the combinatorial limit in depth minimization. In *Proc. IEEE Int'l Conf. on Computer-Aided Design*, pages 110–114, 1993.

[9] J. Cong and Y. Ding. Flowmap: An optimal technology mapping algorithm for delay optimization in lookup-table based fpga designs. *IEEE Trans. Computer-Aided Design*, 13:1–12, June 1994.

[10] J. Cong and Y. Hwang. Simultaneous depth and area minimization in lut-based fpga mapping. In *Proc. ACM/SIGDA International Symposium on FPGAs*, 1995.

[11] E. M. Sentovich, K. J. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldanha, H. Savoj, P. R. Stephan, R. K. Brayton and A. Sangiovanni-Vincentelli. SIS: A system for sequential circuit synthesis. Technical report, 1992.

[12] M. A. Iyer and M. Abramovici. Fire: A fault-independent combinational redundancy identification algorithm. *IEEE Trans. VLSI Syst.*, 4(2):259–301, June 1996.

[13] Y. M. Jiang, A. Krstic, K. T. Cheng, and M. Marek-Sadowska. Post-layout logic restructuring for performance optimization. In *Proc. of Design Automation Conf.*, pages 662–665, 1997.

[14] P. K. N. Vemuri and R. Tessier. Bdd-based logic synthesis for lut-based fpgas. *ACM Transactions on Design Automation of Electronic Systems*, 7(4):501–525, Oct. 2002.

[15] Y. L. Wu, C. C. Cheung, D. I. Cheng, and H. Fan. Further improve circuit partitioning using gbaw logic perturbation techniques. *IEEE Trans. VLSI Syst.*, 11(3):451–460, June 2003.

[16] Y. L. Wu, W. Long, and H. Fan. A fast graph-based alternative wiring scheme for boolean networks. In *International VLSI Design Conference*, pages 268–273, 2000.