An Integrated Approach for Synthesizing LUT Networks

Shigeru Yamashita Hiroshi Sawada Akira Nagoya
NTT Communication Science Laboratories
2-4 Hikaridai, Seika-cho, Soraku-gun,
Kyoto 619-0237 Japan
{ger, sawada, nagoya}@cslab.kecl.ntt.co.jp

Abstract

This paper presents a method for synthesizing look-up table (LUT) networks. The strategy employed by our method is very different from the strategies of previous methods; many decomposition methods that are not only algebraic but also functional are integrated very well. Our method can be thought of as a general framework for LUT network synthesis integrating various decomposition methods. The experimental results are very encouraging.

1. Introduction

In multi-level logic synthesis, the decomposition of logic representations is very important. Accordingly many decomposition methods have been proposed. Most of them are based on transforming algebraic **expressions** of switching formulas; below, we call them **algebraic** decomposition methods. For example, kernel extraction [1] is a supreme method. The above decomposition methods appear to be reasonable in conjunction with the technology mapping phase for the standard technology libraries.

To realize combinational logic functions using a lookup table (LUT) based field programmable gate array (FPGA), we must generate an LUT network where each LUT is a special node which can realize any function with K (typically 4 or 5) inputs. Most LUT network synthesis methods can be divided into the following two categories.

The methods in the first category are extended methods for the standard technology libraries:

- First, a logic optimizer performs decomposition and technology-independent optimization. In this phase, algebraic decomposition methods are usually used, and the number of literals is used for the cost considering the standard technology libraries.
- Next, a technology mapper covers nodes to K-input nodes.

In this category, there are state-of-the-art methods such as Chortle-d [3], MIS-pga-delay [6][1], and FlowMap [2]. For the covering phase, optimal algorithms have been developed under specified conditions [3, 2]. However, the intermediate networks before the covering phase often affect the final results; in such cases, the final results are sometimes not so good.

The methods in the second category consist of only one phase: they directly transform primary output **functions** (not expressions) in terms of primary inputs represented by an ordered binary decision diagram (OBDD) [8, 5]. (Below, we call transformations of functions **functional** decomposition methods.) Therefore, the final results are not affected by intermediate results and are usually better than the results of the methods in the first category.

The decomposition form of the functional decomposition methods used in the methods in the second category is limited to a specified form based on Roth-Karp decomposition [7]. Therefore, in some cases the decomposition does not work so well, and another type of functional decomposition method may be better [11]. However, it is very difficult to utilize various decomposition methods together in synthesizing LUT networks because the decomposition forms are very different between some methods.

In this paper, we propose a new method which overcomes the drawbacks of the methods mentioned above. Various decomposition methods, such as Disjoint Decomposition [7], Non-Disjoint Bi-Decomposition [11], Weak Division by Kernels, and Davio Expansion can be integrated into our method. Our method can be thought of as an extension of methods in the second category and a general framework for LUT network synthesis integrating various decomposition methods. Although it is rather heuristic in nature, the experimental results are very encouraging.

2. Our LUT Network Synthesis Method

Our problem is to generate the lowest cost network where all nodes are **K-feasible** (the number of fanins $\leq$ K). The cost of a decomposed network is defined as (the number of nodes in the network) + (W $\times$ the levels of the network), where W is the user defined weight.

[1] MIS-pga partially uses a functional decomposition method.

2.1. Concept of Our Method

Our strategy is based on the following concept: "Suppose we have various decomposition methods. We can find the best decomposed network from the search space by considering all of the possible combinations of the decomposition methods and the covering effect." However, performing an exhaustive search for all of the possible combinations is not practical. Therefore, we instead select a "probably best" decomposition at an intermediate decomposition.

If we must think of the covering effect after the decomposition phase, it becomes very difficult to determine a "probably best" decomposition at an intermediate decomposition, because the decomposition forms are likely to be very different between some of the decomposition methods, and so it is very difficult to predict the covering effect when the decomposition is being done.

Considering the above discussion, we evaluate the "cost" of a decomposition form with the following strategy.

- We evaluate the cost of a decomposition including the covering effect at the same time.
- We predict the cost of nodes whose supports are more than K by using a "cost database file," which describes decomposition costs of functions from past design results.

From the above strategy, we can utilize various decomposition methods in our method.

2.2. Outline of Our Method

The overall procedure to generate a network whose nodes are all K-feasible is as follows.

Step 1: Construct nodes corresponding to the primary output functions of the network. We prepare an expression for each node from the given specifications of the input network in order to utilize algebraic decomposition methods. We also prepare an OBDD, which represents the primary output function for each node, to utilize functional decomposition methods.

Step 2: As long as there remains a node that is not K-feasible, we decompose the node by using a decomposition method selected from the prepared methods.

Step 3: Generate a new "cost database file" from the decomposed network as will be mentioned in Section 2.4.

2.3. How to Handle Various Decomposition Methods Uniformly

We characterize a decomposition form of various decomposition methods used in our method as follows: a decomposition form of a node n_i is characterized as a node n_i', which is a replacement of n_i, and newly introduced nodes $n_{i1}, \cdots n_{in}$ which are fanins of n_i'. We can treat most decomposition methods in this form. Figure 1(b) shows an example of this for Bi-Decomposition based methods. We call the set of nodes introduced at the decomposition "DecompArea" (the dotted rectangle in Fig. 1(b)).

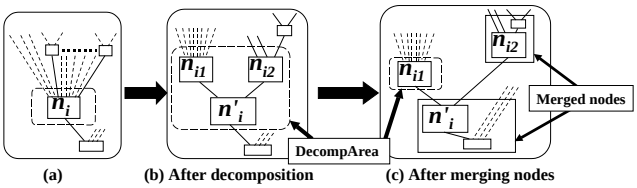

Figure 1. Decomposition Form of a Node.

We select a "possibly best" decomposition form of a node n_i by evaluating the "cost" of the decomposition. Because we want to treat various decomposition methods, we consider the case where the number of fanins of a node in the DecompArea is less than K. For example, the number of fanins of n_i' is two when a decomposition method based on Bi-Decomposition is used. Such a node may be able to be merged into a node not in the DecompArea. Since our strategy does not perform the covering phase after the decomposition phase, we try to merge such a node, which is at the boundary of the DecompArea, into a node not in the DecompArea to form a newly merged node if the merged node is still K-feasible as shown in Fig. 1(c). In this example, n_i' and n_{i2} can be merged into other nodes, and therefore we do not consider them in the decomposition cost. Accordingly, the cost evaluation after the merging of the nodes includes the covering effect at the same time.

For the DecompArea after the merging (the dotted rectangle in Fig. 1(c)), our cost is defined as follows:

$$cost\ of\ a\ decomposition =$$

$$\{\sum_{n_j \in DecompArea} CostLUT(n_j)\} + W \times \{\max_{n_j \in DecompArea} LEV(n_j)\},$$

where W is the user defined weight. $LEV(n_i)$ is recursively defined as follows, and it becomes 0 for a primary input node.

$$LEV(n_j) = \{\max_{n_k\ is\ a\ fanin\ of\ n_j} LEV(n_k)\} + CostLEV(n_j).$$

$CostLUT(n_i)$ and $CostLEV(n_i)$ denote the predicted numbers of K-LUTs and levels to implement the internal function of n_i, respectively. They become 1 for a K-feasible node. However, we cannot know the precise values of $CostLUT(n_i)$ and $CostLEV(n_i)$ if n_i is not K-feasible. Therefore, we determine their values by looking up a cost database file as mentioned in Section 2.4.

2.4. Cost Database File

In most of the previous logic synthesis methods, the cost of a function is usually measured only by the number of supports of the function or literals in the logic expression of the function. Our strategy differs greatly from this strategy: we prepare a cost database file to store the statistical relationships between the three parameters characterizing the output function of a node n_i, and $CostLUT(n_i)$ and $CostLEV(n_i)$. In the present implementation of our method, we use the number of supports of the function, the numbers of cubes and literals in an expression for the function.

We generated a cost file as follows, but clearly this is not the only method.

- We make a first cost database file in which $CostLUT(n_i)$ and $CostLEV(n_i)$ take the same value as follows.

  $$\begin{cases} 1, \text{ if } n_i \text{ is K-feasible} \\ (the\ number\ of\ fanins\ of\ n_i) - K + 1, \text{ otherwise} \end{cases}$$

  This value is taken from [5]. In this first cost database file, the numbers of cubes and literals are not considered.

- Using the first cost database file, we generate various networks by our method. We then make a second cost database file in which each entry describes a statistical relationship between the above three parameters for the output function of each node in the decomposed networks, and the numbers of transitive fanins of the node and levels of the node, which correspond to $CostLUT(n_i)$ and $CostLEV(n_i)$, respectively.

From the cost database file, we calculate $CostLUT(n_i)$ and $CostLEV(n_i)$ as follows.

- Extract three parameters from the internal function of n_i.
- Find the values of $CostLUT(n_i)$ and $CostLEV(n_i)$ in the entry that produces the best fit for the three parameters in the cost database file.

2.5. Additional Operations in Our Method

Some decomposition methods should always be applied first if possible. For example, simple disjunctive decomposition provides good decomposition forms that can be found relatively fast [9]. Such decomposition methods should be applied before the decomposition of a node in Step 2. This process reduces the total computation time.

Our strategy takes almost no account of the sharing of common functions between some functions. Therefore, we have prepared operations called algebraic resubstitution and boolean resubstitution as the decomposition methods in our method, to check whether an existing node can be used for the decomposition of another node.

In our method, we have also prepared another operation to share common functions: after all decompositions, the minimization method proposed in [10] is performed to replace the output of a node with that of another node.

2.6. Advantages of Our Method

To sum up, our method has the following features.

- Various decomposition methods can easily be integrated into our method. If a new decomposition algorithm has been developed, we can easily check its effectiveness.
- We can get various results from the various implementations of our method; the implementation of our method varies depending on what types of decomposition methods are integrated and what cost database file is used. Therefore, we are able to obtain various decomposed networks for a given specification, and consequently we can explore a large design space.

Our method can treat various decomposition methods, and can generate various decomposed networks. This means that our strategy inherently takes a large amount of time. However, the checking of each decomposition method in Step 2 can be done independently; we can perform decomposition methods in parallel on different processors, and this reduces the computation time.

3. Experimental Results

Table 1 shows a comparison of mapping results for 5-LUT networks between our method and several of the well-known level-optimized LUT network synthesis methods. The sub-columns "♯lut" and "♯lvl" show the numbers of 5-LUTs and network levels, respectively. "CPU" indicates the CPU run-time (sec.) on a Sun Ultra 2 2200. To compare our results with the other results, the total numbers for the same circuits are presented in the lower part of the table. From the results, we can observe that our method is the most robust because various decomposition methods are utilized in it. It is very interesting that we can get various results from the various implementations of our method; an implementation of our method varies depending what type of decomposition methods are integrated and what cost database file is used. Our result shown in Table 1 were produced by the implementation as follows:

- Disjoint Decompositions, Non-Disjoint Bi-Decompositions and Davio Expansion were used.
- We used the second cost database file produced by the method in Section 2.4.

Table 1. Comparison of Mapping Results for 5-LUT Networks.

circuit	ALTO[4]		mispga-d		chortle-d		FlowMap-r		BoolMap-D[5]		Ours		
name	♯lut	♯lvl	♯lut	♯lvl	♯lut	♯lvl	♯lut	♯lvl	♯lut	♯lvl	♯lut	♯lvl	CPU
5xp1	19	2	21	2	26	3	23	3	13	2	11	2	0.33
9sym	7	3	7	3	63	5	61	5	7	3	5	4	0.55
alu2	61	6	122	6	227	9	148	8	43	4	33	4	5.44
alu4	259	8	155	11	500	10	245	10	268	7	85	7	77.33
apex4	-	-	-	-	1112	6	-	-	-	-	302	4	32.67
apex6	229	4	274	5	308	4	232	4	189	4	161	4	691.89
apex7	77	4	95	4	108	4	80	4	78	3	61	4	204.98
clip	33	3	54	4	-	-	-	-	-	-	11	3	2.39
count	47	3	81	4	91	4	73	4	42	2	30	4	732.5
duke2	156	4	164	6	241	4	187	4	193	5	150	4	162.75
f51m	15	3	23	4	-	-	-	-	-	-	10	3	0.34
misex1	14	2	17	2	19	2	15	2	15	2	10	2	0.22
misex3	251	6	-	-	-	-	-	-	-	-	166	6	196.64
rd73	8	2	8	2	-	-	-	-	-	-	6	2	0.21
rd84	13	3	13	3	61	4	43	4	10	2	7	3	0.54
sao2	38	3	45	5	-	-	-	-	-	-	21	3	3.56
vg2	26	3	39	4	55	4	38	4	30	4	21	4	120.16
z4ml	5	2	10	2	25	3	13	3	5	2	5	2	0.13
Total	1258	61	1128	67	2836	62	1158	55	893	40	1095	65	-
ALTO	1258	61									793	61	
mispga-d			1128	67							627	55	
chortle-d					2836	62					881	48	
FlowMap-r							1158	55			579	44	
BoolMap-D									893	40	579	44	

4. Conclusion

We have proposed an efficient method for synthesizing LUT networks. In our method, many decomposition methods that are not only algebraic but also functional are integrated very well. Our method can be thought of as a general framework for LUT network synthesis integrating various decomposition methods.

Our method inherently takes a large amount of time. However, this is not a serious problem because our method can easily be performed in parallel on different processors and the cost of processors has recently been decreasing.

As future work, we would like to evaluate the final routing results from the LUT networks generated by our method.

References

[1] R. K. Brayton, R. Rudell, A. Sangiovanni-Vincentelli, and A. R. Wang. MIS: A Multiple-Level Logic Optimization System. *IEEE Trans. CAD*, CAD-6(6):1062–1081, Nov. 1987.

[2] J. Cong and Y. Ding. An Optimal Technology Mapping Algorithm for Delay Optimization in Lookup-Table Based FPGA Designs. In *Proc. ICCAD*, pages 48–53, Nov. 1992.

[3] R. J. Francis, J. Rose, and Z. Vranesic. Technology Mapping of Lookup Table-Based FPGAs for Performance. In *Proc. ICCAD*, pages 568–571, Nov. 1991.

[4] J.-D. Huang, J.-Y. Jou, and W.-Z. Shen. An Iterative Area/Performance Trade-Off Algorithm for LUT-Based FPGA Technology Mapping. In *Proc. ICCAD*, pages 13–17, Nov. 1996.

[5] C. Legl, B. Wurth, and K. Eckl. A Boolean Approach to Performance-Directed Technology Mapping for LUT-Based FPGA Designs. In *33rd ACM/IEEE Design Automation Conference*, pages 730–733, June 1996.

[6] R. Murgai, N. Shenoy, and R. K. Brayton. Performance Directed Synthesis for Table Look Up Programmable Gate Arrays. In *Proc. ICCAD*, pages 572–575, Nov. 1991.

[7] J. P. Roth and R. M. Karp. Minimization Over Boolean Graphs. *IBM Journal*, pages 227–238, Apr. 1962.

[8] H. Sawada, T. Suyama, and A. Nagoya. Logic Synthesis for Look-up Table Based FPGAs Using Functional Decomposition and Support Minimization. In *Proc. ICCAD*, pages 353–358, Nov. 1995.

[9] H. Sawada, S. Yamashita, and A. Nagoya. Restructuring Logic Representations with Easily Detectable Simple Disjunctive Decompositions. In *Proc. of the Design, Automation and Test in Europe (DATE'98)*, pages 755–759, Feb. 1998.

[10] S. Yamashita, H. Sawada, and A. Nagoya. A New Method to Express Functional Permissibilities for LUT based FPGAs and Its Applications. In *Proc. ICCAD*, pages 254–261, Nov. 1996.

[11] S. Yamashita, H. Sawada, and A. Nagoya. New Methods to Find Optimal Non-Disjoint Bi-Decompositions. In *ASP-DAC '98*, pages 59–68, Feb. 1998.