ALTO: An Iterative Area/Performance Tradeoff Algorithm for LUT-Based FPGA Technology Mapping

Juinn-Dar Huang, *Student Member, IEEE*, Jing-Yang Jou, *Member, IEEE*, and Wen-Zen Shen, *Member, IEEE*

***Abstract*—In this paper, we propose an iterative area/performance tradeoff algorithm for look-up table (LUT)-based field programmable gate array (FPGA) technology mapping. First, it finds an area-optimized, performance-considered initial network by a modified area optimization technique. Then, an iterative algorithm consisting of several resynthesizing techniques is applied to trade the area for the performance in the network gracefully. Experimental results show that this approach can efficiently provide a complete set of mapping solutions from the area-optimized one to the performance-optimized one for the given design. Furthermore, these two extreme solutions produced by our algorithm outperform the results provided by most existing algorithms. Therefore, our algorithm is very useful for the timing-driven, LUT-based FPGA synthesis.**

***Index Terms*—Critical-path, performance tradeoffs, programmable gate array, technology mapping.**

I. Introduction

FIELD programmable gate arrays (FPGAs) are modern logic devices that can be programmed by the users to implement their own logic circuits. Because of the short turnaround time compared with that of the standard ASIC process, they have become very popular in rapid system prototyping recently. Many FPGA architectures have been proposed and the look-up table (LUT)-based architecture is the most popular one. It consists of many configurable k-LUTs, and a k-LUT can implement an arbitrary function with up to k inputs. For example, in Xilinx XC3000 architecture [1], k is equal to 5.

Many FPGA technology mapping algorithms have been proposed in previous studies. According to the objectives, they can be roughly classified into three categories.

1) *Area optimization [2]–[10]:* These algorithms minimize the number of LUTs used to implement the given circuit based on the assumption that the number of LUTs in the FPGA design is a good measurement of the area of FPGA implementations.
2) *Performance optimization [11]–[21]:* These algorithms minimize the circuit delay time of the specified design. Because the propagation delay for every LUT is almost identical, the most popular delay model used in FPGA synthesis is the unit delay model. That is, the circuit delay is estimated by the maximum level of LUTs in the synthesized circuit. In general, the smaller number of levels always results in the better performance, though some algorithms take the extra interconnection delay into account to get the more precise estimation [12]–[14], [17], [18].
3) *Routability optimization [22], [23]:* These algorithms maximize the routability for easy placement and routing at the later stage of the technology mapping. It works well for the FPGA architectures, which lack routing resources.

The common limitation of the previously described algorithms is that only one extreme mapping solution is produced. That is, these algorithms can provide relatively good results for their own objectives. but may not provide a solution based on designers' specific requests. In general, the area-optimized mapping solution has more levels while the performance-optimized solution uses more LUTs. So, an area-optimization algorithm may provide a compact solution but may not meet the performance specification, and a performance-optimization algorithm may provide a fast solution but uses too many LUTs. Therefore, a set of mapping solutions positioned at the comprehensive area/level tradeoff curve should be generated to provide the maximum flexibility for the designers.

An algorithm called *FlowMap-r* has been proposed to provide such capability [24]. It starts from a level-optimal[1] mapping solution produced by *FlowMap* [16]. Then, it performs a number of *depth* relaxation operations to get the area/level tradeoff curve. In this paper, we will use an alternative approach to achieve this goal. Instead of beginning from a level-optimal solution, our new approach starts from an area-optimized solution with level consideration. Then, it applies a series of resynthesizing operations to gradually reduce the number of levels without increasing too many LUTs. Our strategy seems more similar to the typical approaches used in the timing-driven logic synthesis targeting for standard cells or gate arrays [25]. Experimental results show that our algorithm can provide better solutions than those of *FlowMap-r*. Moreover, our approach not only can produce a comprehensive area/level tradeoff curve but also can provide competitive level-optimized solutions compared with those produced by current existing performance-optimization algorithms.

This paper is organized as follows. Section II introduces some basic terminologies and definitions used in this paper.

Manuscript received September 10, 1998; revised December 29, 1998. This work was supported by the R.O.C. National Science Council under Grant NSC87-2215-E-009-040.

The authors are with the Department of Electronics Engineering, National Chiao Tung University, Hsinchu 300, Taiwan, R.O.C. (e-mail: jyjou@bestmap.ee.nctu.edu.tw).

Publisher Item Identifier S 1063-8210(00)00757-5.

[1]The optimality is achieved under some assumptions. See [16] for details.

Section III describes how we get the area-optimized initial solution with level consideration for a given circuit. In Section IV, our iterative area/performance tradeoff algorithm is presented in detail. Section V shows the extensive experimental results, and the concluding remarks are given in Section VI.

II. Preliminaries

Some basic terminology and definitions used in this paper are given in this section. A combinational Boolean network can be represented by a directed acyclic graph (DAG), $G(V, E)$. Each node, $v \in V$, represents a logic function and each directed edge, $e(i, j) \in E$, i and $j \in V$, represents that node i is a fanin of node j. A **fanin** of a node f is a node which the function of f explicitly depends on. $fanin(f)$ is the set consisting of all fanins of f. A **fanout**, g, of f is a node where $f \in fanin(g)$. $fanout(f)$ is the set consisting of all fanouts of f. A **primary input** (PI) of the network is a node without any incoming edge, and a **primary output** (PO) of the network is a node without any outgoing edge. Node i is a **transitive fanin** of node j if there exists at least one path from node i to node j. A node is k-**feasible** if the number of its fanins is no more than k. A network is k-feasible if all nodes are k-feasible in the network. The level of a node v, $l(v)$, is the number of nodes in the longest path from a PI node to v. So the level of a PI node is defined to be 0. The level of the other node, v, is defined to be the maximum level of its fanins plus 1, i.e.,

$$l(v) = \max_{u \in fanin(v)} l(u) + 1.$$

Thus, the level of each node in the network can be computed in the topological order. The level of the network N, $l(N)$, is defined to be the maximum level of the PO nodes. The **required level** of a network N, which is user-specified and denoted as $rl(N)$, indicates the maximum level of the desired resultant network. Thus, for each PO node v of the network N, the required level, $rl(v)$, is defined to be $rl(N)$. The required level of any other node v is defined to be the minimum required level of its fanouts minus 1, i.e.,

$$rl(v) = \min_{u \in fanout(v)} rl(u) - 1.$$

Thus, the required level of each node in the network can be computed in the reverse topological order. A node v is **critical** if $rl(v)$ is less than $l(v)$. A **critical fanin** of f is a fanin of f, which is critical. $cfi(f)$ is the set containing all critical fanins of f. A **cone**, $cone(v, l)$, is a subset of V, which contains the root node v as well as its transitive fanins whose level are no less than l.

III. Initial Network Generations

As mentioned in Section I, our area-performance tradeoff algorithm starts from an area-optimized mapping solution with level consideration. To generate such a good initial network, two key points have to be concerned.

1) The initial network should also be as compact as the one obtained by other area-optimization algorithms and should be generated as fast as possible.
2) The level should also be considered while generating the area-optimized initial network.

Considering these two reasons, the *chortle-crf* algorithm [3] is selected to be enhanced. It can generally produce a good area-optimized solution in a short time. Moreover, though it is a pure area-optimization algorithm, we will show later that it is relatively easy to be enhanced to take the level information into account.

The *chortle-crf* algorithm first performs the AND–OR decomposition to transform the original network into the one containing nodes representing AND or OR functions only. Then, it traverses all nodes in the network in the topological order. For each node, two major decompositions, namely, the two-level decomposition and the multilevel decomposition, are applied. In the two-level decomposition phase, the bin-packing technique with the heuristic reconvergent path optimization is applied to pack the fanin nodes into a set of k-feasible nodes (bins with the maximum size k). Then, it applies the multilevel decomposition to further reduce the number of k-feasible nodes required to implement the function represented by this node. In these two phases, minimizing the number of k-feasible nodes is the only objective, i.e., no attempt on level-optimization is made. For example, consider two mapping results for the same function shown in Fig. 1, where the number labeled outside the gate represents the level and the number labeled inside the gate represents the ID of the node. Both of them could be generated by the *chortle-crf* algorithm[2] because these two results are identical in terms of the number of LUTs. But they are quite different from the performance point of view. The level of Fig. 1(a) is 6, while the level of Fig. 1(b) is 4. Thus, if the level information is properly considered during area-optimized mapping, the performance could also be improved at the same time. Fig. 1 gives such an example that the level can be reduced without paying any extra LUT.

To achieve this kind of level reduction, two modifications should be made to the original *chortle-crf* algorithm.

1) In the two-level decomposition phase, the bin-packing algorithm incorporated with the heuristic maximum sharing decreasing (MSD) algorithm[3] is used. The MSD algorithm selects the fanin nodes to be packed under some criteria targeting for the area optimization. At this time, if two candidate fanin nodes have the same priority, the one with the lower level is chosen. Thus, the resultant nodes are potentially with the smaller level.
2) In the multilevel decomposition phase, the modifications are as follows.
   a) An ordered list of packed fanin nodes is obtained by sorting the number of their fanin nodes in decreasing order. When two candidates have the same number of fanin nodes, the one with the smaller level is ordered before the one with the larger level.
   b) The first node in the list is moved out to connect to the first $(k-1)$-feasible packed fanin node with the maximum number of fanins in the list.
   c) Repeat a) and b) until only one node is in the list.

[2]Details about the mapping procedures can be found in [26].

[3]Details about MSD algorithms can also be found in [26].

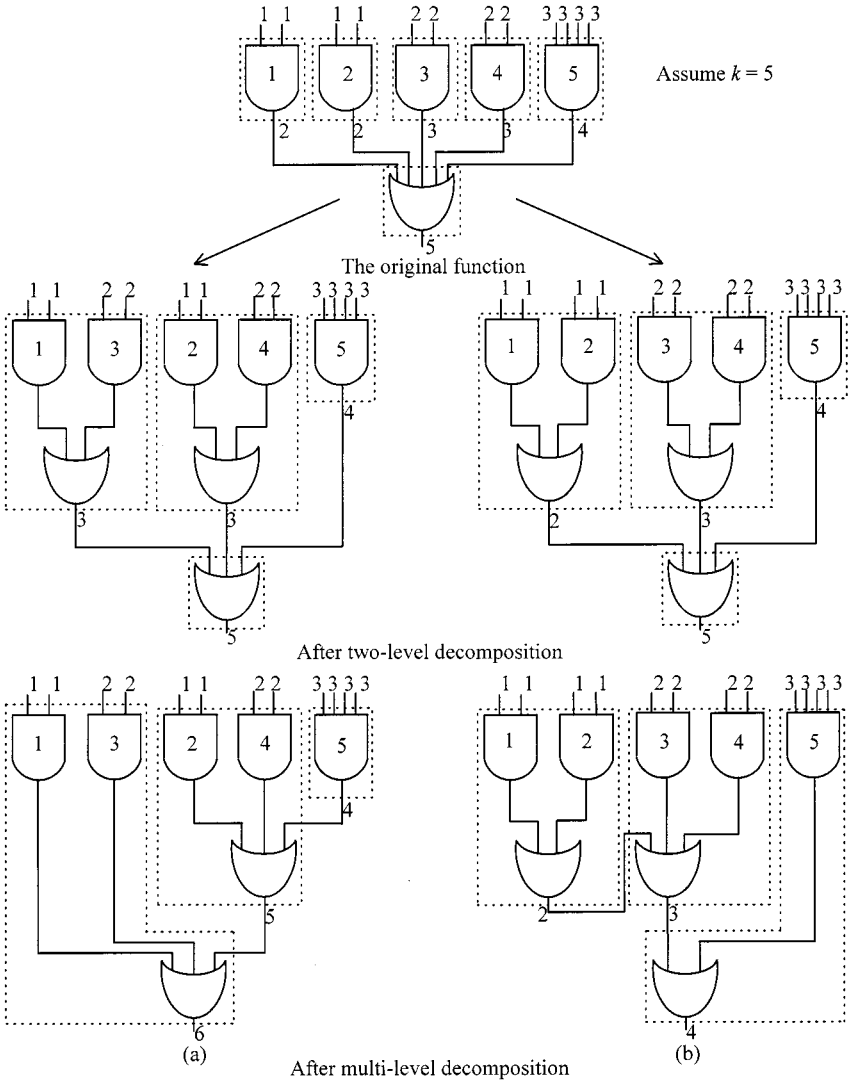


Fig. 1. Two possible mapping results of *chortle-crf*.

3) In order to evaluate the quality of our modified *chortle-crf* algorithm *MODIFIED*, several different approaches are introduced for extensive comparisons. The algorithm *ORIGINAL* represents the original *chortle-crf*, which ignores the level information. The resultant network produced by this approach will have the average performance in terms of levels. The approach *WORST* is specially designed to find the possibly worst case of *ORIGINAL*. It is modified in the opposite directions we propose.

The algorithms described above have been implemented in an SIS environment developed by the University of California, Berkeley [2], [27], [28]. An experiment over a set of MCNC and ISCAS benchmark circuits is performed to evaluate all these approaches. All benchmark circuits are first optimized by the MIS standard multilevel optimization script [27]. Then, all approaches are independently applied to make them five-feasible. Thus, each node in the network can be implemented by a 5-LUT. The mapping results of different approaches are shown in the first three columns of Table I. Over 25 benchmark circuits, all three approaches use almost the same number of LUTs. That is because all three implement the identical *chortle-crf* algorithm from the area point of view. However, from the performance point of view, they produce quite different results. On average, *MODIFIED* uses 10% fewer levels than that of *ORIGINAL*. Moreover, *MODIFIED* uses 15% fewer levels than that of *WORST*. By the way, the amounts of CPU time consumed by these three approaches are almost identical.

According to previous experiments, some circuits must be collapsed into the two-level form, then be decomposed by Roth–Karp decomposition to get the better mapping solutions both in area and performance [2]. However, the time complexity of the collapsing could be exponential for the circuits with the large number of PIs. So we only apply this collapsing operation to small circuits with a limited number of PIs, e.g., ten. Therefore, we develop an approach *MIXED*, which applies not only the modified *chortle-crf* algorithm, but also a modified Roth–Karp decomposition algorithm [9], [10] to those collapsed circuits. The results of the initial network generation are shown in column *MIXED* of Table I. In order to evaluate the quality of our level-considered area-optimized initial networks, the mapping results generated by one of the most popular area-optimization algorithms, mispga [2], are

TABLE I
EXPERIMENTAL RESULTS OF OBTAINING INITIAL NETWORKS BY DIFFERENT ALGORITHMS

	MODIFIED		*ORIGINAL*		*WORST*		*MIXED*		*mispga*	
CKT	#LUT	#LVL	#LUT	#LVL	#LUT	#LVL	#LUT	#LVL	#LUT	#LVL
5xp1	25	4	26	4	26	4	19	2	18	3
9sym	54	8	53	9	53	9	7	3	7	3
9symml	57	7	59	10	59	9	7	3	7	3
C499	70	7	70	7	70	7	70	7	70	7
C880	93	10	88	10	88	10	93	10	81	10
alu2	107	15	110	16	109	20	61	6	102	14
alu4	162	14	173	17	168	20	162	14	167	19
apex6	209	8	196	8	196	9	209	8	174	7
apex7	62	5	59	6	59	6	62	5	56	6
b9	35	4	35	4	35	4	35	4	35	4
clip	31	5	32	5	33	5	31	5	27	6
count	31	5	31	6	31	6	31	5	31	5
des	849	13	877	19	878	22	849	13	867	21
duke2	116	7	115	6	115	7	116	7	115	7
e64	80	17	80	17	80	17	80	17	80	17
f51m	27	4	30	4	30	4	15	3	24	5
misex1	15	3	16	3	16	3	15	3	17	3
misex2	32	3	30	4	30	4	32	3	31	3
misex3	144	13	151	14	151	15	144	13	153	16
rd73	19	4	22	4	23	4	8	2	6	2
rd84	53	7	54	7	56	8	13	3	10	3
rot	187	12	183	13	184	13	187	12	182	15
sao2	37	5	38	6	38	6	37	5	42	5
vg2	22	4	21	5	21	5	22	4	21	5
z4ml	5	2	5	2	5	2	5	2	5	2
Total	2522	186	2554	206	2554	219	2310	159	2328	191

shown at column mispga of Table I.[4] On average, our *MIXED* algorithm uses slightly fewer LUTs and 17% fewer levels than that of mispga, respectively. Moreover, *MIXED* only takes 521 s to complete this experiment, while mispga takes 4215 s on a Sun SPARC 20 workstation. The experimental results clearly show that our *MIXED* algorithm can efficiently provide an excellent area-optimized starting point of a given circuit for the later area/performance tradeoff operations.

IV. ITERATIVE AREA/PERFORMANCE TRADEOFF ALGORITHM

After introducing the algorithm to get the level-considered area-optimized initial network, we will present our iterative area/performance tradeoff algorithm. Starting with an area-optimized k-feasible network, our goal is to reduce the level of the network without increasing too many extra k-feasible nodes. The delay model used here is the unit delay model, i.e., the delay time is estimated by the level of the resultant network. The outline of our algorithm is presented in Fig. 2.

Given a network, our algorithm reduces its level by one each time until the desired target level is achieved or no further improvement can be made. If an unachievable low value is set as the target level, say 0, then the complete set of area/level tradeoff mapping solutions from the area-optimized one to the level-optimized one can be obtained. In the following, we will describe this algorithm in detail.

In our algorithm, the required level of the given network is assigned to its current level minus 1 at each iteration while the current level is still larger than the target level. The level of each node in the network is labeled in the topological order. Then, under the given required level of the network, the required level of each node is calculated in the reverse topological order. Hence, critical nodes can be easily identified. A function $Gain\_Calculation$ is then defined to calculate the gain for each critical node. Conceptually, this gain is designed to represent how much the performance of the entire network can be improved if the level of the corresponding node can be reduced by one. Hence, the critical node with the largest gain will be selected to be resynthesized first. The principle of $Gain\_Calculation$ is based on the fact that reducing the level of each critical fanin by one for a node v is equivalent to reducing the level of v by one. So, the gain of a critical node v is distributed to all of its critical fanins by the following formula:

$$gain(u) += gain(v)/|cfi(v)|, \quad \forall u \in cfi(v).$$

Note that the gain of a node does not directly propagate backward to all of its critical fanins. That is because if the gain is just simply propagated backward to its critical fanin nodes, the node with the largest gain will always appear near PI nodes, and obviously, those nodes are not always the best candidates. Conceptually, those critical nodes that fanout to more critical nodes will have higher priority to be chosen for level reduction.

```
Iterative_Area/Level_Trade-Off( network Net,
                                required_level Target_Level) {
  Original_Level ← l(Net);
  N ← Duplicate(Net);
  L1: while(l(N) > Target_Level) {
    rl(N) ← l(N) - 1;
    Label_Node_Level(N);
    Critical_Node_List ← Identify_Critical_Node(N);
    Gain_Calculation(Critical_Node_List);
    Sort_Gain_In_Decreasing_Order(Critical_Node_List);
    L2: foreach(candidate node v in Critical_Node_List) {
      for(Remap_Level ← l(v) - 1; Remap_Level > 0;
                                    --Remap_Level){
        New_Cone_List ← Remap cone(v, Remap_Level)
                        with several resynthesis techniques;
        cone(v', New_Level) ← The one with the minimum level
                              in New_Cone_List;
        if(l(v') < l(v)) {
          N ← replace cone(v, Remap_Level) with
                cone(v', New_Level);
          mark Local_Success;
          exit L2;     // exit foreach loop
        }
      }
    }
    if( Local_Success is marked ) {
      if(l(N) < l(Net)) replace  Net with N; // also a global success
    }
    else
      exit L1;   // exit while loop
  }
  if(l(Net) == Target_Level)
    return Success;
  else if(l(Net) < Original_Level)
    return Partial_Success;
  else
    return Failure;
}
```

Fig. 2. Iterative area/level tradeoff algorithm.

[4]The resultant networks are obtained by applying the optimization script recommended in [28].

The gain of each critical PO node is assigned to one by default. Then, the gain of each other critical node can be calculated in the reverse topological order. All critical nodes are then sorted by their gains in decreasing order. If two nodes have the same gain, the one with smaller level is selected first. That is because the effect of the level reduction on nodes near PI could potentially have a bigger impact.

For each critical node in the sorted list, a number of performance optimization techniques are applied to reduce its level. Unfortunately, most performance optimization techniques developed for the semicustom design, such as buffer insertion, gate sizing and fanout replication, etc., cannot be directly applied to the LUT-based FPGA architecture under the unit delay model. Thus, the partial resynthesis is the best possible way to reduce the level of a node, i.e., to resynthesize the critical node and some of its transitive fanin nodes together targeting for level reduction. In our algorithm, a greedy strategy is used to select the transitive fanin nodes to be resynthesized. At first, only the candidate node v and its critical fanin nodes, i.e., $cone(v, l(v)-1)$, are resynthesized to reduce the level of v. If the attempt fails, $cone(v, l(v) - 2)$ is selected to be resynthesized next. This process is not terminated until the level of v is reduced or the attempt fails even for $cone(v, 1)$.

Currently, three performance-driven resynthesizing techniques are applied to the selected partial network for level reduction. Detailed discussions on these three techniques are shown in Section IV-A–C.

After applying all resynthesizing techniques, the best mapping solution, in which the level of the root node is minimum, is selected. If the level of the root node is identical in two different solutions, the one with smaller number of increasing nodes is selected. If the level of the root node in the newly synthesized cone is not smaller than that of the original root node v, the next candidate node in the critical node list is selected and the process is repeated. Otherwise, the level of the root node in the newly synthesized cone is smaller than that of the original root node v, the new cone replaces the old one in the duplicated network N, and a local success is marked. If a local success results in a global success, i.e., the level of the modified network N is smaller than that of the original network net, then the original network is updated. After a local or a global success, the whole procedure starting from the level labeling is repeated because the network has been modified. This process is continued until the level of the final resynthesized network is no more than the target level, or is terminated after an iteration in which no local success can be obtained by resynthesizing all candidate critical nodes. Finally, the algorithm returns the last saved network as the resultant network as well as a status flag, which is set according to the given target level, the level of the original network, and the level of the resultant network.

The next three sections describe the resynthesis techniques currently employed in our algorithm.

A. Modified Chortle-d Algorithm

The first technique is based on *chortle-d* algorithm [11]. It performs the AND–OR decomposition first. For each node in the topological order, its fanin nodes of the same level are grouped into separate strata. The bin packing technique with the reconvergent path optimization is then applied to minimize the number of nodes in each stratum. Finally, it connects the outputs of nodes in stratum l to unused inputs of nodes in stratum $l + 1$. Note that additional nodes may be added to stratum $l + 1$ to provide unused inputs. This process is completed when there is only one node in the highest stratum.[5] Fig. 3 shows an example of this approach. The level of the root node in Fig. 3 is five while the level of the root node synthesized by *chortle-crf* is six. To avoid increasing the level of the network, some additional nodes are added, e.g., the node marked with the asterisk in Fig. 3. However, we find out that some of those extra nodes can be collapsed to its fanout nodes while the network is still k-feasible. Therefore, an extra pass, which finds those nodes and collapses them into their fanout nodes, is appended to the original *chortle-d* algorithm to reduce the number of nodes required. Thus, the major drawback of *chortle-d*, using too many nodes to trade the levels, is partially improved.

B. Modified Chortle-crf Algorithm

The second technique is our modified *chortle-crf* algorithm, *MODIFIED*, described in Section III. The area overhead is generally smaller than that of *chortle-d* based algorithm if it can successfully reduce the level of the candidate node.

C. Timed Roth–Karp Decomposition

The third technique is a newly developed technique called the timed Roth–Karp decomposition. As we described before, some networks should be collapsed into the two-level form, then be decomposed by Roth–Karp decomposition to get the better mapping solutions. So if the number of PI nodes of the selected partial network is smaller than a predefined upper bound, it is first collapsed into a two-level logic, then decomposed to a k-feasible network by the timed Roth–Karp decomposition algorithm. Our timed Roth–Karp decomposition is based on a modified Roth–Karp decomposition algorithm [9], [10] and is enhanced with the level-minimization technique.

The original Roth–Karp decomposition can decompose a function into several subfunctions. For example,

$$F(X, Y) = G(\alpha_1(X), \alpha_2(X), \cdots, \alpha_m(X), Y)$$
$$X \cup Y = fanin(F)$$

where X is called the bound set and Y is called the free set. Thus, if $m < |X|$ and $X \cap Y = H$, Roth–Karp decomposition breaks a function F with a large number of fanins into several subfunctions, G and α's, with fewer fanins.[6] From the area point of view, m should be made as small as possible—especially for the LUT-based FPGA architecture. In general, m strongly depends on what the bound set is. Therefore, a modified algorithm that selects a good bound set for area optimization is developed [9]. However, this modified algorithm only focuses on minimizing m and ignores the level information during the decomposition. Thus, our timed Roth–Karp decomposition is then developed to minimize the level of G. The outline of this algorithm is presented in Fig. 4.

[5] Details about the *chortle-d* algorithm can be found in [26].

[6] Details about Roth–Karp decomposition can be found in [9], [10], [29].

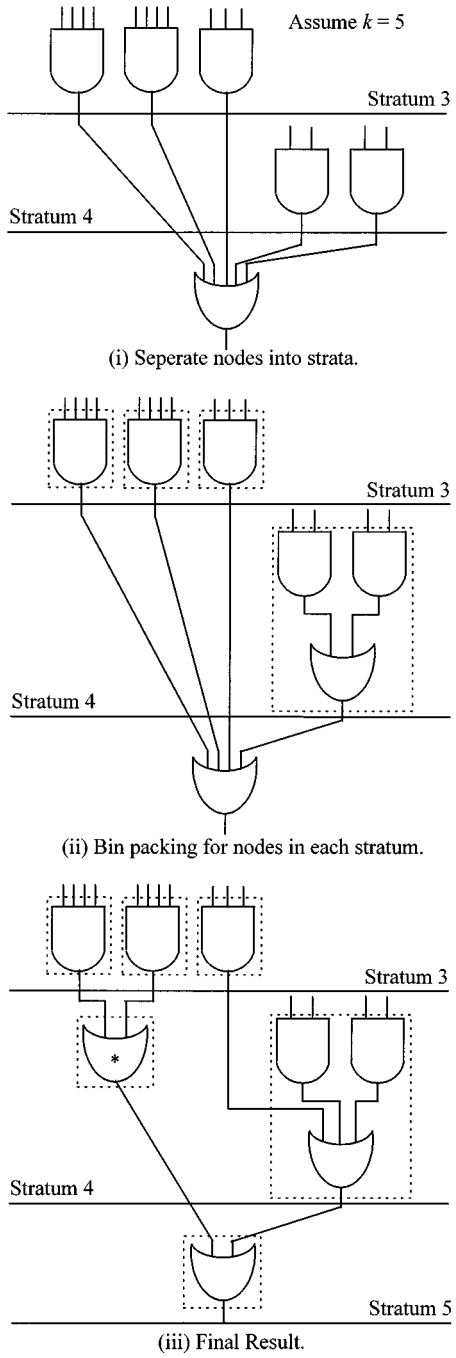


Fig. 3. Example of *chortle-d* algorithm.

The objective of this algorithm is to minimize the level of G as well as to reduce the number of LUTs required for decomposition. One might intuitively select a set of fanins with the smallest level as the bound set. However, as we mentioned before, Roth–Karp decomposition works only if $m < |X|$. Thus, in our algorithm, an initial bound set is first selected for area minimization by using the approach proposed in [9]. The target level of nodes in the bound set, $best\_bs\_lvl$, is set to the $|X|$th smallest level of the fanin node $\in fanin(F)$. Then, a procedure that properly exchanges the variables between the bound set and the free set is repeatedly invoked to reduce the largest level of nodes in the bound set to $best\_bs\_lvl$. In order to calculate the estimated level of G under a specific bound set, an additional procedure, ED, which calculates the worst-case estimated delay for a function F, is needed. Procedure ED is defined as

```
ED(Fanin_Number f, LUT_Size k) {
  if(f ≤ k)
    return 1;
  else
     return f − k + 1;
}
```

where f is set to $|fanin(F)|$ and k is the maximum number of inputs for a LUT. That is, if $f \leq k$, only a k-LUT is required to implement F and the estimated delay is one. But if $f > k$, it might require another $f-k$ iterations of Shannon cofactoring to decompose F into several k-feasible subfunctions in the worst case. So the estimated delay is $f - k + 1$ in such a case. Thus,

```
Timed_Roth-Karp_Decomposition
(Network Net, Node F, Bound_Set_Size bss, LUT_Size k){
  (bound_set, free_set) ← Bount_Set_Selection(F, bss);
  // proposed in [9]
  best_bs_lvl ← the bss-th smallest level of the fanin node
                ∈ fanin(F);
  fss ← |fanin(F)| - bss; // calculate the size of the free set
  L1:while(the largest level of the node∈bound set>best_bs_lvl){
    max_bs_lvl ← the largest level of the node w ∈ bound_set;
    max_fs_lvl ← the largest level of the node ∈ free_set;
    (G, α1, α2,..., αm) ←
              Roth-Karp_Decomposition(F, bound_set);
    initial_EL ← max(max_bs_lvl + ED(bss, k), max_fs_lvl)
                 + ED(m + fss, k);
    L2: foreach(variable v, Level(v) < max_bs_lvl,v ∈ free set) {
      (new_bound_set, new_free_set) ←
            exchange w ∈ bound_set and v ∈ free_set;
      max_new_bs_lvl ← the maximum level of the node
                       ∈ new_bound_set;
      max_new_fs_lvl ← the maximum level of the node
                       ∈ new_free_set;
      (G', α'1, α'2,..., α'n) ←
              Roth-Karp_Decomposition(F, new_bound_set);
      new_EL ← max(max_new_bs_lvl + ED(bss, k),
                   max_new_fs_lvl) + ED(n + fss, k);
      if(n < bss && new_EL ≤ initial_EL) {
        (bound_set, free_set) ← (new_bound_set, new_free_set);
        mark Success;
        exit L2; // exit foreach loop
      }
    }
    if( Success is not marked) exit L1;   // exit while loop
  }
  (G, α1, α2,..., αm) ←
              Roth-Karp_Decomposition(F, bound_set);
  replace F with (G, α1, α2,..., αm) in Net;
}
```

Fig. 4. Timed Roth–Karp decomposition algorithm.

TABLE II
EXPERIMENTAL RESULTS OF OUR AREA/LEVEL TRADEOFF ALGORITHMS

CKT	L|Aopt #LVL	L #LUT	L-1 #LUT	L-2 #LUT	L-3 #LUT	L-4 #LUT	L-5 #LUT	L-6 #LUT	L-7 #LUT	L-8 #LUT
5xp1	2	19	-							
9sym	3	7	-							
9symml	3	7	-							
C499	7	70	70	78	-					
C880	10	93	93	96	96	-				
alu2	6	61	66	-						
alu4	14	162	180	181	199	208	209	259	-	
apex6	8	209	199	201	208	211	-			
apex7	6	62	62	66	81	-				
b9	4	35	36	-						
clip	5	31	32	33	-					
count	5	31	32	43	-					
des	13	849	852	859	876	858	845	801	784	-
duke2	7	116	117	128	156	-				
e64	17	80	81	82	83	84	85	86	87	88
f51m	3	15	-							
misex1	3	15	14	-						
misex2	3	32	37	-						
misex3	13	144	149	161	167	173	201	218	251	-
rd73	2	8	-							
rd84	3	13	-							
rot	12	187	188	193	199	204	214	-		
sao2	5	37	34	38	-					
vg2	4	22	26	-						
z4ml	2	5	-							

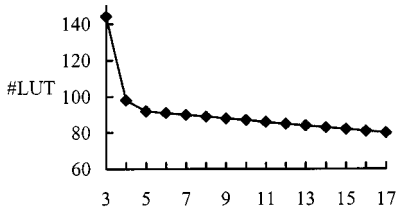
an estimated level of the corresponding G node for a specific bound set can be calculated as

$$\text{estimated level} = \max\left(\max_{x\in X} l(x) + ED(|X|, k), \max_{y\in Y} l(y)\right) + ED(m + |Y|, k).$$

For the initial bound set, $bound\_set$, the estimated level $initial\_EL$ is calculated. Then, for each candidate variable v of the initial free set, $free\_set$, whose level is less than the largest level of the node $w \in bound\_set$, v and w are exchanged to get the new bound set new_bound_set. For new_bound_set, the estimated level of the corresponding G' after Roth–Karp decomposition, new_EL, is also calculated. If $new\_EL \leq initial\_EL$, then new_bound_set replaces $bound\_set$ and the variable exchange process is restarted. This process is not terminated until the largest level of the node $\in$ bound set is reduced to $best\_bs\_lvl$ or all variable exchange attempts fail in the last iteration.

V. EXPERIMENTAL RESULTS

Our area/level tradeoff algorithm has been implemented in an SIS environment. In order to evaluate its quality, a set of comprehensive mapping solutions from the area-optimized one to the level-optimized one is produced for each benchmark circuit described in Section III. All solutions should retain the condition of being five-feasible to be implemented by the five-LUT FPGA architecture. The results are shown in Table II. The column $L|Aopt$ shows the level of the area-optimized initial network. The remaining columns represent the numbers of five-feasible nodes required to implement the circuit for the designated level. From Table II, it is found that our algorithm can really provide a wide range of mapping solutions to be chosen by the designers. Some benchmark circuits, such as 5xp1, 9sym, and 9symml, etc., do not have many tradeoff design points because the best designs are found for both area and level. The most dramatic



#LVL	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3
#LUT	80	81	82	83	84	85	86	87	88	89	90	91	92	98	144

Fig. 5. Various mapping results of the benchmark circuit e64.

TABLE III
COMPARISONS BETWEEN OUR ALGORITHM AND *FlowMap-r*

CKT	L #LVL	L *ALTO*	L *Fmap-r*	L-1 *ALTO*	L-1 *Fmap-r*	L-2 *ALTO*	L-2 *Fmap-r*	L-3 *ALTO*	L-3 *Fmap-r*	L-4 *ALTO*	L-4 *Fmap-r*
5xp1	4	-	22	-	23	19	-	-	-		
C499	7	70	-	70	130	78	151	-	-		
C880	11	-	172	93	179	93	195	96	211	96	-
alu2	9	-	140	-	148	-	-	61	-	66	-
alu4	12	181	240	199	244	208	245	209	-	259	-
apex6	8	209	-	199	-	201	220	208	221	211	232
apex7	6	62	-	62	76	66	80	81	-	-	-
count	5	31	57	32	73	43	-	-	-		
des	9	858	934	845	969	801	987	784	1003	-	1087
duke2	7	116	151	117	161	128	172	156	187	-	-
rd84	6	-	38	-	42	-	43	13	-	-	-
rot	10	193	-	199	210	204	213	214	218	-	243

case is the mapping results of the benchmark circuit e64 shown in Fig. 5. The levels of mapping results produced by our algorithm vary from 17 to 3. Table III shows the comparisons between the results produced by our algorithm, denoted as *ALTO* (area/level tradeoff), with those produced by another area/level tradeoff algorithm named *FlowMap-r* [24], denoted as *Fmap-r*. For most of the benchmark circuits, the mapping solutions of *ALTO* outperform those of *FlowMap-r* on the same level.

Finally, in order to show how good the level-optimized results *ALTO* can achieve, the results produced by previously proposed level optimization algorithms, including *chortle-d* [11], *mispga-delay* [12], *TechMap-D* [13], *FlowMap* [16], *FlowSYN* [15], *DOGMA* [20], and *BoolMap-D* [21], are listed in Table IV for extensive comparisons. For 18 benchmark circuits, *ALTO* on average requires 22% and 56% fewer levels and LUTs than those of *chortle-d*, respectively. For 24 benchmark circuits, *ALTO* on average requires 23% fewer levels and 34% fewer LUTs than those of *mispga-delay*. For 23 benchmark circuits, *ALTO* on average requires 13% and 38% fewer levels and LUTs than those of *TechMap*. For 17 benchmark circuits, *ALTO* on average requires 17% and 28% fewer levels and LUTs than those of *FlowMap*. For 17 benchmark circuits, *ALTO* on average requires 3% and 20% fewer levels and LUTs than those of *FlowSYN*. For 16 benchmark circuits, *ALTO* on average requires 16% and 17% fewer levels and LUTs than those of *DOGMA*. The only exception is that *BoolMap* on average requires 10% and 5% fewer levels and LUTs than those of *ALTO* for 17 benchmark circuits. For 25 benchmark circuits, *ALTO* takes a total of 5412 s to obtain the level-optimized networks from their area-optimized ones on a Sun SPARC 20 workstation. The time varies from 2 to 1906 s for various circuits.

TABLE IV
COMPARISONS AMONG OUR ALGORITHM AND OTHER LEVEL OPTIMIZATION ALGORITHMS

	chortle-d		*mispga-d*		*TechMap*		*FlowMap*		*FlowSYN*		*DOGMA*		*BoolMap*		*ALTO*	
CKT	#LUT	#LVL	#LUT	#LVL	#LUT	#LVL	#LUT	#LVL	#LUT	#LVL	#LUT	#LVL	#LUT	#LVL	#LUT	#LVL
5xp1	26	3	21	2	17	2	22	3	20	2	24	3	13	2	19	2
9sym	63	5	7	3	9	3	60	5	7	3	59	5	7	3	7	3
9symml	59	5	7	3	9	3	55	5	7	3	50	4	7	3	7	3
C499	382	6	199	8	148	4	68	4	133	5	68	4	101	4	78	5
C880	329	8	259	9	213	7	124	8	232	8	98	8	146	7	96	7
alu2	227	9	122	6	197	8	155	9	113	6	138	9	43	4	68	5
alu4	500	10	155	11			253	9	249	9			268	7	259	8
apex6	308	4	274	5	252	5	238	5	257	4	231	5	189	4	211	4
apex7	108	4	95	4	86	4	79	4	89	4	68	4	78	3	81	3
b9			47	3	49	4									36	3
clip			54	4	40	3									33	3
count	91	4	81	4	71	4	31	5	75	3	31	5	42	2	43	3
des	2086	6	1397	11	1395	8	1310	5	893	4	938	5	594	3	784	6
duke2	241	4	164	6	175	4	174	4	187	4	173	4	193	5	156	4
e64	139	7	212	5	218	4									144	3
f51m			23	4	38	4									15	3
misex1	19	2	17	2	18	2	16	2	15	2	16	2	15	2	14	2
misex2			37	3	41	3									37	2
misex3															251	6
rd73			8	2	10	2									8	2
rd84	61	4	13	3	16	3	46	4	13	3	53	4	10	2	13	3
rot	326	6	322	7	315	6	234	7	262	6	210	7	228	6	214	7
sao2			45	5	45	4									33	3
vg2	55	4	39	4	36	4	29	3	45	4	27	3	30	4	26	3
z4ml	25	3	10	2	9	2	5	2	6	2	5	2	5	2	5	2
chortle-d	5045	94													2225	73
mispga-d			3608	116											2387	89
TechMap					3407	93									2128	81
FlowMap							2899	84							2081	70
FlowSYN									2603	72					2081	70
DOGMA											2189	74			1822	62
BoolMap													1969	63	2081	70

The experimental results clearly show that *ALTO* can effectively produce a better set of area/level tradeoff mapping solutions than those of *FlowMap-r* for most circuits. Moreover, the level-optimized solutions produced by *ALTO* outperform those produced by all existing level-optimization algorithms except *BoolMap-D*. Because *ALTO* is iterative in nature and is originally designed to produce the area/level tradeoff curve for a given design efficiently, it is not expected to find better solutions all the time than those produced by other algorithms, which take a global view of the whole circuit and target only for the level-optimized solution. So, it is very surprising that *ALTO* can outperform most existing level-optimization algorithms. Furthermore, *ALTO* provides a framework to resynthesize the critical part of the network. Although only three techniques are utilized currently, later developed techniques can be easily incorporated into this framework to further improve the quality of the algorithm.

VI. CONCLUSIONS

In this paper, we propose an iterative area/level tradeoff algorithm for LUT-based FPGA technology mapping. The approach begins with finding a level-considered area-optimized initial network for the given circuit by performing the modified *chortle-crf* algorithm and modified Roth–Karp decomposition. Our iterative area/level tradeoff algorithm *ALTO* is then applied to get the set of complete area/level tradeoff mapping solutions. Experimental results show that *ALTO* can provide not only an excellent area/level tradeoff curve, but also the level-optimized solutions, which compete favorably with those provided by most existing level optimization algorithms. Thus, this algorithm is working well on the timing-driven technology mapping for the LUT-based FPGA architecture.

REFERENCES

[1] *The Programmable Logic Data Book*. San Jose, CA: Xilinx, 1993.
[2] R. Murgai, N. Shenoy, R. K. Brayton, and A. Sangiovanni-Vincentelli, "Improved logic synthesis algorithms for table look up architectures," in *Proc. Int. Conf. Computer-Aided Design*, Nov. 1991, pp. 564–567.
[3] R. J. Francis, J. Rose, and Z. Vranesic, "Chortle-crf: fast technology mapping for lookup table-based FPGA's," in *Proc. 28th Design Automation Conf.*, June 1991, pp. 227–233.
[4] D. Filo, J. C. Yang, F. Mailhot, and G. D. Micheli, "Technology mapping for a two-output RAM-based field-programmable gate arrays," in *Proc. Eur. Design Automation Conf.*, Feb. 1991, pp. 534–538.

[5] Y. T. Lai, M. Pedram, and S. B. K. Vrudhula, "BDD based decomposition of logic functions with application to FPGA synthesis," in *Proc. 30th Design Automation Conf.*, June 1993, pp. 642–647.
[6] T.-T. Hwang, R. M. Owens, and M. J. Irwin, "Logic synthesis for field-programmable gate arrays," *IEEE Trans. Computer-Aided Design*, vol. 13, pp. 1280–1287, Oct. 1994.
[7] B. Wurth, K. Eckl, and K. Antreich, "Functional multiple-output decomposition: Theory and an implicit algorithm," in *Proc. 32nd Design Automation Conf.*, June 1995, pp. 54–59.
[8] H. Sawada, T. Suyama, and A. Nagoya, "Logic synthesis for look-up table based FPGA's using functional decomposition and support minimization," in *Proc. Int. Conf. Computer-Aided Design*, Nov. 1995, pp. 353–358.
[9] W.-Z. Shen, J.-D. Huang, and S.-M. Chao, "Lambda set selection in Roth–Karp decomposition for LUT-based FPGA technology mapping," in *Proc. 32nd Design Automation Conf.*, June 1995, pp. 65–69.
[10] J.-D. Huang, J.-Y. Jou, and W.-Z. Shen, "Compatible class encoding in Roth–Karp decomposition for two-output LUT architecture," in *Proc. Int. Conf. Computer-Aided Design*, Nov. 1995, pp. 359–363.
[11] R. J. Francis, J. Rose, and Z. Vranesic, "Technology mapping of look-up table-based FPGA's for performance," in *Proc. Int. Conf. Computer-Aided Design*, Nov. 1991, pp. 568–571.
[12] R. Murgai, N. Shenoy, R. K. Brayton, and A. Sangiovanni-Vincentelli, "Performance directed synthesis for table lookup programmable gate arrays," in *Proc. Int. Conf. Computer-Aided Design*, Nov. 1991, pp. 572–575.
[13] P. Sawkar and D. Thomas, "Performance directed technology mapping for look-up table based FPGAs," in *Proc. 30th Design Automation Conf.*, June 1993, pp. 208–212.
[14] C.-S. Chen, Y.-W. Tsay, T.-T. Hwang, A. C.-H. Wu, and Y.-L. Lin, "Combining technology mapping and placement for delay-optimization in FPGA designs," in *Proc. Int. Conf. Computer-Aided Design*, Nov. 1993, pp. 123–127.
[15] J. Cong and Y. Ding, "Beyond the combinatorial limit in depth optimization for LUT-based FPGA designs," in *Proc. Int. Conf. Computer-Aided Design*, Nov. 1993, pp. 110–114.
[16] ——, "FlowMap: An optimal technology mapping algorithm for delay optimization in lookup-table based FPGA designs," *IEEE Trans. Computer-Aided Design*, vol. 13, pp. 1–12, Jan. 1994.
[17] H. Yang and D. F. Wang, "Edge-Map: Optimal performance driven technology mapping for iterative LUT based FPGA designs," in *Proc. Int. Conf. Computer-Aided Design*, Nov. 1994, pp. 150–155.
[18] N. Togawa, M. Sato, and T. Ohtsuki, "Maple: A simultaneous technology mapping, placement, and global routing algorithm for field-programmable gate arrays," in *Proc. Int. Conf. Computer-Aided Design*, Nov. 1994, pp. 156–163.
[19] H. Shin and C. Kim, "Performance-oriented technology mapping for LUT-based FPGAs," *IEEE Trans. VLSI Syst.*, vol. 3, pp. 323–327, June 1995.
[20] J. Cong and Y.-Y. Hwang, "Structural gate decomposition for depth-optimal technology mapping in LUT-based FPGA design," in *Proc. 33rd Design Automation Conf.*, June 1996, pp. 726–729.
[21] B. Wurth, K. Eckl, and K. Antreich, "A Boolean approach to performance-directed technology mapping for LUT-based FPGA designs," in *Proc. 33rd Design Automation Conf.*, June 1996, pp. 730–733.
[22] N. Bhat and D. Hill, "Routable technology mapping for LUT FPGAs," in *Proc. Int. Conf. Computer Design*, Oct. 1992, pp. 95–98.
[23] M. Schlag, J. Kong, and P. K. Chan, "Routability-driven technology mapping for lookup table-based FPGA's," *IEEE Trans. Computer-Aided Design*, vol. 13, pp. 13–26, Jan. 1994.
[24] J. Cong and Y. Ding, "On area/depth trade-off in LUT-based FPGA technology mapping," *IEEE Trans. VLSI Syst.*, vol. 2, pp. 137–148, June 1994.
[25] J. P. Fishburn, "LATTIS: An iterative speedup heuristic for mapped logic," in *Proc. 29th Design Automation Conf.*, June 1992, pp. 488–491.
[26] S. D. Brown, R. J. Francis, J. Rose, and Z. G. Vranesic, *Field Programmable Gate Arrays.* Norwell, MA: Kluwer, 1992.
[27] R. K. Brayton, R. Rudell, A. Sangiovanni-Vincentelli, and A. R. Wang, "MIS: A multi-level logic optimization system," *IEEE Trans. Computer-Aided Design*, vol. CAD–6, pp. 1062–1081, Nov. 1987.
[28] E. M. Sentovich, K. J. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldanha, H. Savoj, P. R. Steven, R. K. Brayton, and A. Sangiovanni-Vincentelli, "SIS: A system for sequential circuit synthesis," Electron. Res. Lab. Memo. UCB/ERL M92/41, May 1992.
[29] J. P. Roth and R. M. Karp, "Minimization over Boolean graphs," *IBM J. Res. Develop.*, pp. 227–238, Apr. 1962.

Juinn-Dar Huang (S'99) received the B.S. and Ph.D. degrees in electronics engineering from National Chiao Tung University, Hsinchu, Taiwan, R.O.C., in 1992 and 1997, respectively.

His current work at Global Unichip Corporation, Taiwan, involves reusable soft silicon IP creation and system-on-a-chip (SoC) integration. His current research interests include IP reuse methodology and SoC design.

Jing-Yang Jou (S'82–M'83–S'84–M'85) received the B.S. degree in electrical engineering from National Taiwan University, Taipei, Taiwan, R.O.C., and the M.S. and Ph.D. degrees in computer science from the University of Illinois at Urbana-Champaign.

He is Professor in the Department of Electronics Engineering, National Chiao Tung University, Hsinchu, Taiwan. He has worked at GTE Laboratories and at Bell Laboratories. His research interests include behavioral and logic synthesis, VLSI designs and CAD for low power, design verification, synthesis and design for testability, and hardware/software codesign. He has published more than 70 papers.

Dr. Jou is a Member of Tau Beta Pi, and the recipient of the Distinguished Paper Award of the IEEE International Conference on Computer-Aided Design, 1990. He served as Technical Programmer Chair of the Asia-Pacific Conference on Hardware Description Languages (APCHDL'97).

Wen-Zen Shen (S'80–M'88) was born in Hsinchu, Taiwan, R.O.C., in 1950. He received the M.S. and Ph.D. degrees in electronics engineering from the National Chiao Tung University (NCTU), Hsinchu, in 1977 and 1982, respectively.

Upon his graduation, he joined the Department of Electronics Engineering, NCTU, where he has served as Director of the University Computer Center and Chairman of the Department of Electronics Engineering. In 1992, he initiated the Chip Implementation Area (CIC) at the National Science Council, where he has been Program Director of CIC since 1992. Currently, he is Professor and Dean of the College of Electrical Engineering and Computer Science, NCTU. His current research interests focus on VLSI designs and CAD for low-power, logic synthesis, and low-voltage, low-power mixed-mode integrated circuit designs.

Dr. Shen is a Member of Phi Tau Phi. He is the recipient of the 1985 Distinguished Teaching Award of the NCTU and the 1988 Distinguished Teaching Award of the Ministry of Education, R.O.C.