An Iterative Area/Performance Trade-Off Algorithm for LUT-Based FPGA Technology Mapping

Juinn-Dar Huang, *Jing-Yang Jou*, and *Wen-Zen Shen*
Department of Electronics Engineering, National Chiao Tung University, Taiwan

Abstract

In this paper, we propose an iterative area/performance trade-off algorithm for LUT-based 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 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, the area-optimized one and the performance-optimized one, produced by our algorithm outperform the results of most existing algorithms. Therefore, our algorithm is very useful for the timing driven FPGA synthesis.

1. Introduction

Field Programmable Gate Arrays (FPGA's) are modern logic devices which can be programmed by the users to implement their own logic circuits. Because of the short turnaround time, they 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*-LUT's which 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 two categories:

(1) Area Optimization [2-6]: These algorithms minimize the number of LUT's used to implement the given circuit based on the assumption that the number of LUT's in the FPGA design is a good measurement of the area of FPGA implementations.
(2) Performance Optimization [7-12]: 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 LUT's in the synthesized circuit. In general, the smaller number of levels always results in the better performance.

The common limitation of the previously described algorithms is that only one extreme mapping solution is produced. These algorithms can provide the relatively good results for their own objectives but may not provide a solution based on the designers' specifications. Thus, a set of mapping solutions positioned at the comprehensive area/level trade-off curve should be generated to provide the maximum flexibility for the designers.

An algorithm, named *FlowMap-r*, has been proposed to provide this capability [13]. It starts from a level-optimal mapping solution produced by *FlowMap* [11]. Then, it performs a number of **depth relaxation** operations to get the area/level trade-off 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 LUT's. 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 trade-off curve but also can provide competitive level-optimized solutions compared with those produced by most existing performance-optimization algorithms.

This paper is organized as follows. Section 2 introduces some basic terminologies and definitions used in this paper. Section 3 describes how we get the area-optimized initial solution with level consideration for a given circuit. In Section 4, our algorithm performing iterative area/performance trade-off is presented in detail. Section 5 shows the experimental results and the concluding remarks are given in Section 6.

2. Preliminaries

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 **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 fanin nodes 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, that is,

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

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, that is,

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

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 the fanin of f which is critical. $cfi(f)$ is the set containing all critical fanins of f. A **cone**, $cone(v, l)$, is a subgraph $H \subseteq G$ that contains the root node v as well as its transitive fanin nodes whose level is no less than l.

3. The Initial Network Generations

As mentioned in Section 1 that our area-performance trade-off 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 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 multi-level 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. Then, it applies the multi-level 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, that is, no attempt on level-optimization is made. If the level information is properly considered during area-optimized mapping, the performance could also be improved at the same time.

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 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 multi-level decomposition phase, the modifications are as follows:

(i) An ordered list of packed fanin nodes is obtained by sorting on 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.

(ii) 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.

(iii) Repeat (i) and (ii) until only one node is in the list.

In order to evaluate the quality of our modified *chortle-crf* algorithm *MODIFIED*, several different approaches are introduced for comprehensive 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 has been implemented in SIS environment which is developed by UC Berkeley [2, 14]. 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 multi-level optimization script [14]. Then, all approaches are independently applied to make them 5-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 twenty-five benchmark circuits, all three approaches use almost the same number of LUT's. It is because all of 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. However, the time complexity of the collapsing could be exponential for the circuits with the large number of PI's. So we only apply this collapsing operation to small circuits with a limited number of PI's, for example, 10. Therefore, we develop an approach *MIXED*, which applies not only the modified *chortle-crf* algorithm but also a modified Roth-Karp decomposition algorithm [5-6] to such collapsed circuits. The results of our initial network generation are shown at 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 shown at column *mispga* of Table I. On average, our *MIXED* algorithm uses a little fewer LUT's and 17% fewer levels than that of *mispga*, respectively. Moreover, *MIXED* only takes 520 seconds to complete this experiment while *mispga* takes 3950 seconds 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 trade-off operations.

4. Iterative Area/Performance Trade-Off Algorithm

After introducing the algorithm to get the level-considered area-optimized initial network, we will present our iterative area/performance trade-off algorithm. Starting with an area-optimized *k*-feasible network, our goal is to develop an algorithm 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, that is, the delay time is estimated by the level of the resultant network. The outline of our algorithm is presented in Figure 1.

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

```
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);
        Identify_Critical_Node(N);
        Critical_Node_List ⟵ Gain_Calculation(N);
        Sort_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 resynthesizing 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;
}
```

Figure 1 : Our iterative area/level trade-off algorithm.

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 a node by one is equivalent to reducing the level of each of its critical fanin nodes by one. So, the gain of a critical node *v* is distributed to all of its critical fanins by the following formula:

$$gain(u) \mathrel{+}= gain(v)/\left|cfi(v)\right|, \forall\, u \in cfi(v).$$

The gain of a node does not directly propagate forward to all of its critical fanins. It is because that if the gain is just simply propagated to its critical fanins, the node with largest gain will always appear near PI nodes, and obviously, this is not always true.

The gain of each critical PO node is assigned to 1 by default. Then, the gain of each other critical node can be calculated in the reverse topological order. The critical nodes are then sorted by their gains in decreasing order. If two nodes have the same gain, the one with smaller level is listed first. It is because that the effect of the level reduction near PI nodes could potentially be propagated to the other part of the network and has bigger impact.

For each critical node in the sorted list, a number of performance optimization techniques are applied to reduce its level. However, most performance optimization techniques developed for the semi-custom 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 resynthesizing is the most possible way to reduce the level of a node. That is, to resynthesize the critical node and some of its transitive fanin nodes together. In our algorithm, a greedy strategy is used to select the transitive fanin nodes which should be resynthesized. At first, only the candidate node *v* and its critical fanin nodes, that is, *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 also fails even for *cone*(*v*, 1).

Currently, three resynthesizing techniques are applied to the selected partial network for level reduction. The first technique is based on *chortle-d* algorithm [7]. 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 $l + 1$. Notice 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. However, some of such additional 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 further reduce the number of nodes. Thus, the major drawback of *chortle-d*, using too many nodes to trade the levels, is partially improved.

The second technique is our modified *chortle-crf* algorithm, *MODIFIED*, described in Section 3. The area overhead is generally smaller than that of *chortle-d* based algorithm if it can successfully reduce the level of the candidate node.

The third technique is the modified 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 under a predifined upper bound, it is first collapsed then decomposed by a modified Roth-Karp decomposition algorithm proposed in [5-6].

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 smaller than that of the original root node *v*, the new cone is added to replace 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.

5. Experimental Results

Our area/level trade-off algorithm has also been implemented in 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 3. All solutions should retain 5-feasible to be implemented by the 5-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 5-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 and 9sym, etc., do not have a lot of trade-off design points because the best designs are found for both area and level. The most dramatic case is the mapping results of the benchmark circuit e64 in which 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* (stands for Area/Level Trade-Off), with those produced by another area/level trade-off algorithm named *FlowMap-r* [13], denoted *FM-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* [7], *mispga-delay* [8], *FlowMap* [11] and *FlowSYN* [10], are listed in Table IV for comparison. For 18 benchmark circuits, *ALTO* on average requires 17% and 56% fewer levels and LUT's than those of *chortle-d*, respectively. For 24 benchmark circuits, *ALTO* on average requires 19% fewer levels and 34% fewer LUT's than those of *mispga-delay*. For 17 benchmark circuits, *ALTO* on average requires 11% and 28% fewer levels and LUT's than those of *FlowMap*. The only exception is that *FlowSYN* on average requires 4% fewer levels but 25% more LUT's than those of *ALTO* for 17 benchmark circuits. However, our algorithm is iterative and constructive in nature, it can be applied to much more varieties of circuits than other algorithms.

For 25 benchmark circuits, *ALTO* totally takes 4931 seconds to obtain the level-optimized networks from their area-optimized ones in a SUN SPARC 20 workstation. The time varies from 2 seconds to 1906 seconds for an individual circuit.

The experimental results clearly show that *ALTO* can effectively produce a better set of area/level trade-off mapping solutions than those of *FlowMap-r* for most circuits. Furthermore, two extreme solutions, the area-optimized one and the level-optimized one, produced by *ALTO* either outperform or are competitive with those produced by other existing area-optimization algorithms and level-optimization algorithms, respectively.

6. Conclusions

In this paper, we propose an iterative area/level trade-off algorithm for LUT-based FPGA technology mapping. The approach begins with finding a level-considered area-optimized network for the given circuit by performing the modified *chortle-crf* algorithm. Our iterative area/level trade-off algorithm *ALTO* is then applied to get the set of complete area/level trade-off mapping solutions. Experimental results show that *ALTO* can provide not only an excellent area/level trade-off 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*, Xilinx Inc., San Jose, 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*, pp. 564-567, Nov. 1991.

[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.*, pp. 227-233, June 1991.

[4] Y. T. Lai, M. Pedram, and Sarma B. K. Vrudhula, "BDD based decomposition of logic functions with application to FPGA synthesis," in *Proc. 30th Design Automation Conf.*, pp. 642-647, June 1993.

[5] 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.*, pp. 65-69, June 1995.

[6] 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*, pp. 359-363, Nov. 1995.

[7] R. J. Francis, J. Rose, and Z. Vranesic, "Technology mapping of look-up table-based FPGAs for performance," in *Proc. Int. Conf. Computer-Aided Design*, pp. 568-571, Nov. 1991.

[8] 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*, pp. 572-575, Nov. 1991.

[9] P. Sawkar, and D. Thomas, "Performance directed technology mapping for look-up table based FPGAs," in *Proc. 30th Design Automation Conf.*, pp. 208-212, June 1993.

[10] J. Cong, and Y. Ding, "Beyond the combinatorial limit in depth optimization for LUT-based FPGA designs," in *Proc. Int. Conf. Computer-Aided Design*, pp. 110-114, Nov. 1993.

[11] J. Cong, and Y. Ding, "FlowMap: an optimal technology mapping algorithm for delay optimization in lookup-table based FPGA designs," in *IEEE Trans. on Computer-Aided Design*, vol. 13, no. 1, pp. 1-12, Jan. 1994.

[12] C. Legl., B. Wurth, and K. Eckl, “A Boolean approach to performance-directed technology mapping for LUT-based FPGA designs,” in *Proc. 33rd Design Automation Conf.*, pp.730-733, June 1996.

[13] J. Cong, and Y. Ding, "On area/depth trade-off in LUT-based FPGA technology mapping," in *IEEE Trans. on VLSI Systems*, vol. 2, no. 2, pp. 137-148, June 1994.

[14] R. K. Brayton, R. Rudell, A. Sangiovanni-Vincentelli, and A. R. Wang, "MIS : a multi-level logic optimization system," in *IEEE Trans. on Computer-Aided Design*, vol. 6, no. 11, pp. 1062-1081, Nov. 1987.

Table III : Comparisons between *ALTO* and *FlowMap-r*.

	L	L		L-1		L-2		L-3	
CKT	#lvl	*alto*	*fm-r*	*alto*	*fm-r*	*alto*	*fm-r*	*alto*	*fm-r*
5xp1	4	-	22	-	23	19	-	-	-
C499	7	70	-	-	130	-	151	-	-
C880	11	-	172	93	179	93	195	96	211
alu2	9	-	140	-	148	-	-	61	-
alu4	11	199	244	208	245	209	-	259	-
apex6	7	216	-	216	220	227	221	229	232
apex7	6	60	-	60	76	77	80	-	-
count	5	31	57	32	73	47	-	-	-
des	9	858	934	845	969	801	987	784	1003
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

Table IV : Comparisons among *ALTO* and other level optimization algorithms.

	chortle-d		*mispga-d*		*FlowMap*		*FlowSYN*		*ALTO*	
CKT	#lut	#lvl	#lut	#lvl	#lut	#lvl	#lut	#lvl	#lut	#lvl
5xp1	26	3	21	2	22	3	20	2	19	2
9sym	63	5	7	3	60	5	7	3	7	3
9symml	59	5	7	3	55	5	7	3	7	3
C499	382	6	199	8	68	4	133	5	70	7
C880	329	8	259	9	124	8	232	8	96	8
alu2	227	9	122	6	155	9	113	6	61	6
alu4	500	10	155	11	253	9	249	9	259	8
apex6	308	4	274	5	238	5	257	4	229	4
apex7	108	4	95	4	79	4	89	4	77	4
b9			47	3					36	3
clip			54	4					33	3
count	91	4	81	4	31	5	75	3	47	3
des	2086	6	1397	11	1310	5	893	4	784	6
duke2	241	4	164	6	174	4	187	4	156	4
e64	139	7	212	5					144	3
f51m			23	4					15	3
misex1	19	2	17	2	16	2	15	2	14	2
misex2			37	3					37	2
misex3									251	6
rd73			8	2					8	2
rd84	61	4	13	3	46	4	13	3	13	3
rot	326	6	322	7	234	7	262	6	214	7
sao2			45	5					38	3
vg2	55	4	39	4	29	3	45	4	26	3
z4ml	25	3	10	2	5	2	6	2	5	2
chortle-d	5045	94							2228	78
mispga-d			3608	116					2395	94
FlowMap					2899	84			2084	75
FlowSYN							2603	72	2084	75

Table I : Initial networks obtained 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	212	9	204	10	204	10	212	9	201	10
apex7	60	6	60	7	60	7	60	6	57	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	2523	188	2563	209	2563	221	2311	161	2356	194

* alu4 and des are initially optimized by script.rugged.
Others are initially optimized by script.algebraic.

Table II : Area/level trade-off curves generated by *ALTO*.

	L|Aopt	L	L-1	L-2	L-3	L-4	L-5	L-6	L-7	L-8
CKT	#lvl	#lut	#lut	#lut	#lut	#lut	#lut	#lut	#lut	#lut
5xp1	2	19	-							
9sym	3	7	-							
9symml	3	7	-							
C499	7	70	-							
C880	10	93	93	96	-					
alu2	6	61	-							
alu4	14	162	180	181	199	208	209	259	-	
apex6	9	212	213	216	216	227	229	-		
apex7	6	60	60	77	-					
b9	4	35	36	-						
clip	5	31	32	33	-					
count	5	31	32	47	-					
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	-							