A New Decomposition of Boolean Functions

Elena Dubrova*

Electronic System Design Lab
Department of Electronics
Royal Institute of Technology
Kista, Sweden
elena@ele.kth.se

Abstract

This paper introduces a new type of decomposition of Boolean functions, partitioning the original function into disjoint chains of cubes. We show that using this type of decomposition as a preprocessing step in logic synthesis algorithms may lead to a considerable reduction in the run-time thus allowing to process larger functions, which otherwise cannot be handled due to the time limit.

1 Introduction

Decomposing a graph into its strongly connected components is a classical problem in graph theory. Many algorithms that work with directed graphs begin with such a decomposition, which allows the original problem to be divided into subproblems, one for each strongly connected component. This paper shows how a similar type of decomposition can be defined for logic functiond and demonstrates its application to logic synthesis.

Generally, the problem of functional decomposition can be formulated as follows. Given a function f, express it as a composite function of some set of new functions. Often a composite expression can be found in which the new functions are significantly simpler than f.

Usually, a function can be decomposed in several different ways, depending on the criteria chosen. For example, a classical type of decomposition is the *simple disjunctive* decomposition, where the original function is replaced by two functions, which are disjoint in variables [1], [2]. We define a different type of disjoint decomposition, in which not the *variables*, but the *domain space* of the the function is partitioned into disjoint areas. Our experimental results show that using this type of decomposition as a preprocessing steps in logic synthesis algorithms may lead to a considerable reduction in the run-time, thus allowing to process larger functions, which otherwise cannot be handled due to the time limit.

The paper is organized as follows. Section 2 describes the basic notation and definitions which are used in the sequel. Section 3 defines the new type of decomposition. In Section 4 demonstrate the application of the new decomposition to logic synthesis. Section 5 concludes the paper.

*Supported in part by Research Grant No 240-98-101 from the Swedish Research Council for Engineering Sciences and by a fellowship from the Knut and Alice Wallenbergs foundation of Sweden.

2 Notation

We use the standard definitions and notation in the area of logic synthesis ([4]). The most important notions are briefly summarized in this section.

Let $f(x_1, x_2, \ldots, x_n)$ be an completely specified Boolean function of type $f : \{0,1\}^n \rightarrow \{0,1\}$, of the variables $x_1, \ldots, x_n$.

A point in the domain $\{0,1\}^n$ of the function f is called a *minterm*. The *on-set* and the *off-set* of f are the sets of minterms that are mapped by f to 1 and 0, respectively.

A *product-term* is a Boolean product (AND) of one or more variables $x_1, \ldots, x_n$ or their complements. A convenient representation for a product-term is *cube* [5]. We use the terms *cube* and *product-term* interchangeably. A *sum-of-products* is a Boolean sum (OR) of product-terms.

Let p_i, p_j be cubes of f. The *distance* between p_i and p_j, denoted by $distance(p_i, p_j)$, is the number of empty fields in the intersection of p_i and p_j. When the distance is zero, the two cubes intersect, otherwise they are disjoint.

3 Definition of Decomposition

Recall from the graph theory, that a strongly connected component of a graph $G = (V, E)$ is a maximal set of vertices $U \subseteq V$ such that for every pair of vertices u and v in U, vertices u and v are reachable from each other. There is a linear-time $O(V + E)$ algorithm [6] computing the strongly connected components of a graph.

Now we show how a similar type of decomposition can be defined for logic functions. Let p_i, p_j be cubes of an n-variable Boolean function f. Then we define a relation $\mathcal{R}_1$ as follows:

$$(p_i, p_j) \in \mathcal{R}_1 \quad \text{if and only if} \quad distance(p_i, p_j) \leq k,$$

where $0 \leq k \leq n$.

Let $\mathcal{R}_1^+$ be the transitive closure of $\mathcal{R}_1$. Then $\mathcal{R} \stackrel{df}{=} \mathcal{R}_1^+$ is the equivalence relation, dividing the cubes of f into equivalence classes. Thus, the equivalence classes of $\mathcal{R}$ form a partition of the set of all cubes into *disjoint clusters*. Two cubes are in relation $\mathcal{R}$ either when they on distance k, or they are are reachable from each other through a chain of cubes of distance k. Notice, that we can define such a decomposition with respect to the on-set as well as with respect to the off-set of the function.

As an example, consider the set of cubes $\{010, 011, 001, 100\}$, representing a 3-variable Boolean function

$$f(x_1, x_2, x_3) = \overline{x}_1 x_2 + \overline{x}_1 x_3 + x_1 \overline{x}_2 \overline{x}_3$$

If $k = 1$, then $(010, 011) \in \mathcal{R}_1$, $(011, 001) \in \mathcal{R}_1$ since this pairs of cubes are on distance 1. We also have $(p_i, p_i) \in \mathcal{R}_1$ for all $p_i \in \{010, 011, 001, 100\}$, because $distance(p_i, p_i) = 0$. Computing the transitive closure of $\mathcal{R}_1$, we obtain the following partition of the on-set of f into two disjoint clusters of cubes $\{010, 011, 001\}$ and $\{001\}$.

If $k = 2$, then the only pair of cubes not belonging to $\mathcal{R}_1$ is $(011, 100) \notin \mathcal{R}_1$, because $distance(011, 100) = 3$. Computing the transitive closure of $\mathcal{R}_1$, we get that all the cubes fall into the same equivalence class.

Since $\mathcal{R}$ is equivalence relation, the above decomposition is unique for a given sum-of-product expression of a function. However, a Boolean function can normally be represented by several different sum-of-product expressions, and therefore several different decompositions

Table 1: Number of clusters in some benchmarks functions.

benchmark function	size of on-set	size of of-set	$k=0$		$k=1$		$k=2$	
			N_{on}	N_{off}	N_{on}	N_{off}	N_{on}	N_{off}
5xp1	65	78	20	20	1	1	1	1
9sym	86	72	1	2	1	2	1	2
alu4	575	599	39	6	1	1	1	1
apex1	206	1234	77	2	2	1	1	1
apex2	1035	518	1	1	1	1	1	1
apex3	280	767	131	5	1	1	1	1
apex4	436	1462	340	7	1	1	1	1
apex5	1088	1622	188	23	1	1	1	1
b12	43	75	4	7	1	1	1	1
bw	22	75	18	3	1	1	1	1
clip	120	151	9	8	1	1	1	1
con1	9	9	3	5	1	1	1	1
cordic	914	1191	2	2	1	1	1	1
cps	163	718	79	3	1	1	1	1
duke2	86	317	38	2	1	1	1	1
ex1010	284	1465	8	51	1	1	1	1
e64	65	262	65	1	1	1	1	1
inc	30	61	16	2	1	1	1	1
misex1	12	28	7	1	2	1	1	1
misex2	28	73	7	1	1	1	1	1
misex3	690	807	34	1	1	1	1	1
misex3c	197	641	30	20	1	1	1	1
pdc	145	550	25	1	1	1	1	1
rd53	31	32	23	6	7	2	1	1
rd73	127	147	81	23	9	9	1	1
rd84	255	260	162	96	11	2	1	1
sao2	58	76	28	1	1	1	1	1
seq	336	1569	122	1	1	1	1	1
spla	260	509	118	2	4	1	1	1
squar5	25	31	18	4	1	1	1	1
t481	481	360	17	8	1	1	1	1
table3	175	1503	137	1	1	1	1	1
table5	158	194	20	1	4	1	1	1
vg2	110	194	20	1	1	1	1	1
xor5	16	16	16	16	16	16	1	1

of the same function may exist. For example, the function realizing OR, may be written as $f(x_1, x_2) = x_1 + x_2$ as well as $f(x_1, x_2) = x_1 + \overline{x}_1 x_2$. If $k = 0$, then in the first case, its decomposition has only one cluster, while in the second case it has two clusters.

We have implemented a classical algorithm for computing the transitive closure [6] and tried it on benchmark functions to see how many disjoint clusters we obtain for different values of k. Table 1 shows the results. Columns 2 and 3 give the number of cubes in the covers for on- and off-set of the benchmark functions, correspondently, computed by the two-level AND-OR minimizer Espresso [7]. Columns 4-9 show the number of clusters obtained for the on-set (N_{on}) and for the off-set (N_{off}) of the functions, for different values of distance k between the cubes. One can see that very few benchmarks have more than one cluster for $k \geq 1$. Therefore, the decomposition with $k = 0$ seems to be the most suitable choice.

4 Application to logic synthesis

To demonstrate the application of the new decomposition to logic synthesis, we take an algorithm for three-level AND-OR-XOR minimization, presented in [3], and added clustering as a preprocessing step of the algorithm. First, we briefly describe the basic idea of the algorithm, and then give the experimental results showing the effect of clustering on its run-time and the quality of its solutions.

4.1 Algorithm AOXMIN-MV

In this section we briefly describe the basic idea of the algorithm for three-level AND-OR-XOR minimization AOXMIN-MV, presented in [3]. The target of the algorithm is a three-level logic expression which is an XOR of two sum-of-products. For some practical functions, such an AND-OR-XOR expression is shown to have up to 27 times less product-terms compared to the classical sum-of-products form.

The core of AOXMIN-MV is a group migration algorithm [8] which is an extension of Fiduccia/Mattheyses iterative improvement algorithm [9]. Group migration algorithm repeats the following: given an initial partitioning of objects into two groups, for each object determine the decrease in cost if the object is moved to the other group. Then, move the object that produces the greatest decrease or smallest increase in cost and mark it as *moved*. After all objects have been moved once, the lowest-cost partitioning is selected. If this partitioning has a higher cost then the initial one, then the algorithm stops. Otherwise it iterates taking the new partitioning for the initial partitioning.

Obviously, the run-time of the group migration algorithm depends on the number of the input objects. Without clustering, the objects are cubes from the cover of the on-set of the function. If clustering is added as a preprocessing step (with distance $k = 0$), then the objects are equivalence classes of the on-set of the function, determined by the algorithm for computing the transitive closure. As shown in Table 1, clustering greatly reduces the number of objects to consider. In the next section we demonstrate that this leads to a considerable reduction in the run-time of AOXMIN-MV algorithm.

4.2 Effect of clustering on AOXMIN-MV

We have applied the algorithm AOXMIN-MV with and without clustering to a set of benchmark functions. Table 2 shows the results of the comparison in terms of the total number of

Table 2: Comparison of AOXMIN-MV with and without clustering stage; x indicates timeout ($>$ 50 hours).

Example function	n	m	Espresso [7]		AOXMIN-MV with clustering		AOXMIN-MV without clustering	
			cubes	t,sec	cubes	t,sec	cubes	t,sec
5xp1	7	10	65	0.04	36	40.1	46	104
alu4	14	8	575	3.75	219	243	-	x
amd	14	24	66	0.22	68	1.13	68	7543
b9	16	5	119	0.07	43	118	73	1880
clip	9	5	120	0.18	82	17.2	93	854
dist	8	5	123	0.17	92	23.5	99	3656
in2	19	10	136	0.12	120	1053	133	21956
in5	24	14	62	0.05	62	2.24	62	29929
ex7	16	5	119	0.08	43	119	73	1883
exam	10	10	67	1.25	59	2.64	59	6288
f51m	8	8	77	0.12	33	55.2	63	130
life	9	1	84	0.13	60	200	66	316
p1	8	18	55	0.34	48	1.15	48	2580
radd	8	5	75	0.03	20	15.3	34	247
rd53	5	3	31	0.02	17	13.2	15	71.4
rd73	7	3	127	0.06	62	494	62	323
rd84	8	4	225	0.19	130	1566	127	3653
root	8	5	57	0.09	52	3.31	49	405
sao2	10	4	58	0.03	38	0.83	37	508
t1	21	23	102	0.33	83	101	89	3109
t2	17	16	53	0.06	53	0.87	53	525
t481	16	1	481	0.36	18	218	179	48954
tial	14	8	581	3.21	302	406	328	166142
x9dn	27	7	120	0.06	113	179	120	3106
xor5	5	1	16	0.001	6	7.31	6	7.08
z4	7	4	59	0.03	18	9.34	21	140
adder 2-bit	4	3	11	0.005	7	0.75	6	3.32
adder 3-bit	6	4	31	0.01	11	0.91	15	38.4
adder 4-bit	8	5	75	0.06	20	2.52	38	255
adder 5-bit	10	6	167	0.49	39	13.4	87	1470
mlp. 2-bit	4	4	7	0.007	6	0.63	6	1.12
mlp. 3-bit	6	6	32	0.03	26	1.74	26	68.3
mlp. 4-bit	8	8	128	0.19	103	9.12	107	6371
mlp. 5-bit	10	10	490	3.67	436	141	-	x

cubes in the resulting expressions (columns 6 and 8), and the time taken in seconds (columns 7 and 9). The time is user time measured using the UNIX system command *time*. The time limit was set to 50 hours. The experiments were run on a Sun Ultra 60 operating two 360 MHz CPU and 768 Mb memory.

The table also shows the number of cubes in the cover obtained by the two-level AND-OR minimizer Espresso [7] and the time t to compute it (columns 4 and 5, respectively). Columns 2 and 3 give the number of inputs n and the outputs m of the benchmark functions.

The experiments show that, for most benchmarks, the clustering improves both the quality of the solutions and the run-time of the algorithm.

5 Conclusion

This paper introduces a new type of decomposition of Boolean functions, similar to decomposition of graphs into strongly connected components. Our experiments show that using this type of decomposition as a preprocessing step may considerably reduce the run-time of the algorithm for three-level AND-OR-XOR minimization AOXMIN-MV [3], as well as improve the quality of its solutions.

In the future we plan to investigate which other logic synthesis algorithms could benefit from clustering.

References

[1] R. L. Ashenhurst, "The decomposition of switching functions", *Proc. International Symp. Theory of Switching* Part I vol. 29, 1959, pp. 74-116.

[2] H. A. Curtis, *A New Approach to the Design of Switching Circuits*, Van Nostrand, Princeton, 1962.

[3] E. V. Dubrova, D. M. Miller, J. C. Muzio, "AOXMIN-MV: A Heuristic Algorithm for AND-OR-XOR Minimization", *Proc. 4th International Workshop on the Applications of the Reed-Muller Expansion in Circuit Design,* 1999.

[4] G. De Micheli, *Synthesis and Optimization of Digital Circuits*, McGraw-Hill, 1994.

[5] Y. H. Su, P. T. Cheung, "Computer minimization of multi-valued switching functions", *IEEE Trans. Comput.*, vol. C-21, 1972, pp. 995-1003.

[6] T. H. Cormen, C. E. Leiserson, R. L. Rivest, *Introduction to Algorithms*, The MIT Press, Cambridge, England, 1997.

[7] R.K. Brayton, G. Hachtel, C. McMullen, A. Sangiovanni-Vincentelli, *Logic Minimization Algorithms for VLSI Synthesis*, Kluwer, 1984.

[8] D. Gajski, N. Dutt, A. Wu, S. Lin, *High Level Synthesis: Introduction to Chip and System Design*, Kluwer, 1992.

[9] C. M. Fiduccia, R. M. Mattheyses, "A linear time heuristic for improving network partitions", *Proc. 19th ACM/IEEE Design Automation Conference,* 1982, pp. 175-181.