Communication Based Multilevel Synthesis for Multi-output Boolean Functions *

Paul Molitor

Department of Computer Science
Humbold–Universität zu Berlin
D 10099 Berlin, FRG

Christoph Scholl

Department of Computer Science
Universität des Saarlandes
D 66041 Saarbrücken, FRG

Abstract

A multilevel logic synthesis technique for multi-output boolean functions is presented which is based on minimizing the communication complexity. Unlike the approaches known from literature [1, 5, 6, 8] which in the final analysis decompose each single-output function f_i of a multi-output function $f = (f_1, \ldots, f_m)$ independently of the other single-output functions f_j ($j \neq i$), the approach presented in this paper gives special attention to the fact that there possibly exist some decomposition functions which can be used by different outputs during the decomposition of the single-output functions of f. The benchmarking results (taken from 1991 MCNC multilevel logic benchmarks) which close the paper are promising.

1 Introduction

Most of the approaches attacking the multilevel logic synthesis problem use gate count as optimization criterion. This is based on the belief that gate count is a good estimator for layout area. However, in many cases, this criterion may not be the best estimator. Some recent papers [5, 6, 8] propose an approach different from the one addressed above. This approach to multilevel logic synthesis which originates from Ashenhurst [1], Curtis [3], Hotz [4], and Karp [7] is based on minimizing communication complexity. The methods used to reduce communication complexity employ functional decomposition. A decomposition of a boolean function $f : \{0,1\}^n \to \{0,1\}$ with respect to the input partition $\{X, Y\}$ ($X = \{x_1, \ldots, x_p\}$, $Y = \{y_1, \ldots, y_q\}$, $X \cap Y = \emptyset$, $p+q = n$) is a representation of the form

$$f(x_1, \ldots, x_p, y_1, \ldots, y_q) = g(\alpha_1(\mathbf{x}), \ldots, \alpha_r(\mathbf{x}), \beta_1(\mathbf{y}), \ldots, \beta_s(\mathbf{y}))$$

for all $(x_1, \ldots, x_p, y_1, \ldots, y_q) \in \{0,1\}^n$. α_i and β_j are called *decomposition functions of f*. g is called *composition function.*

With respect to a given input partition $\{X, Y\}$, a single-output function f can be represented as a $2^p \times 2^q$ matrix $M(f)$, the *decomposition matrix of f* or the *chart of f* with respect to $\{X, Y\}$. Each row and column of $M(f)$ is associated with a distinct assignment of values to the inputs in X and Y, respectively, such that $f(\mathbf{x}, \mathbf{y}) = M(f)[\mathbf{x}, \mathbf{y}]$ where $M(f)[\mathbf{x}, \mathbf{y}]$ represents the element of $M(f)$ which lies in the row associated with $\mathbf{x}$ and the column associated with $\mathbf{y}$. Then $r \geq \lceil \log p_1 \rceil$ ($s \geq \lceil \log p_2 \rceil$) where p_1 (p_2) is the number of distinct row patterns (column patterns) in $M(f)$. In the following we always assume $r = \lceil \log p_1 \rceil$ and $s = \lceil \log p_2 \rceil$. Of course the required number of decomposition functions depends strongly on the choice of the input partition $\{X, Y\}$.

In many cases the efficiency of good realizations of boolean functions is based on a clever reuse of subcircuits. Our approach takes this observation into account by a special processing of multi-output functions[1]. The decomposition of a multi-output function $f = (f_1, \ldots, f_m) : \{0,1\}^n \to \{0,1\}^m$ has the following form:

$$f_i(x_1, \ldots, x_p, y_1, \ldots, y_q) = g_i(\alpha_1^{(i)}(\mathbf{x}), \ldots, \alpha_{r_i}^{(i)}(\mathbf{x}), \beta_1^{(i)}(\mathbf{y}), \ldots, \beta_{s_i}^{(i)}(\mathbf{y}))$$

for all $1 \leq i \leq m$. In order to reuse subcircuits our algorithm tries to choose as many decomposition functions $\alpha_j^{(i)}$ as possible as identical functions. Unlike other approaches our synthesis method doesn't decompose each single-output function f_i independently of the other single-output functions f_j ($j \neq i$) and only tests whether some $\alpha_j^{(i)}$ are identical by accident. On the contrary our approach tries to *compute* common decomposition functions for different f_i.

The computation of common decomposition functions for some single-output functions f_i requires a decomposition of all f_i with respect to the same input partition $\{X, Y\}$. However we have to take into consideration that there possibly exist single-output functions f_i and f_j such that there does not exist an input partition good for both f_i and f_j. That's why we have to divide our algorithm into two steps: In the first step we partition $\{f_1, \ldots, f_m\}$ into disjoint sets $Y_1, \ldots, Y_u$ (see section 2). In the second step

*This work was supported in part by DFG grant SFB 124 and the Graduiertenkolleg of the Universität des Saarlandes

[1] Even if the original function f is a single-output function, applying functional decomposition recursively to the decomposition functions $\alpha = (\alpha_1, \ldots, \alpha_r)$ and $\beta = (\beta_1, \ldots, \beta_s)$ demands a generalization to multi-output boolean functions.

we decompose all single-output functions f_i of the *same* set Y_k with respect to the *same* input partition giving special attention to generate these functions in such a way that many can be used in the decomposition of different elements of Y_k (see section 3).

2 Output/input partitioning

In this section we present a heuristic which performs the first step of our algorithm by partitioning $F = \{f_1, \ldots, f_m\}$ into disjoint sets $Y_1, \ldots, Y_u$. To each set Y_k, the algorithm assigns an input partition $\{X^{(k)}, Y^{(k)}\}$ which is 'near-optimal' for each $f_i \in Y_k$. The size $\mid X^{(k)} \mid$ of $X^{(k)}$ is given by a number p[2].

Let IP_p be a subset of all possible input partitions $\{X, Y\}$ with $\mid X \mid = p$ computed in a preprocessing step. Furthermore, let $df_i(A)$ be the minimum number of decomposition functions required by a decomposition of f_i with respect to the input partition $A \in IP_p$. We define $df_i^{min} = \min\{df_i(A);\ A \in IP_p\}$ to be the minimum of these values $df_i(.)$. An input partition $A \in IP_p$ is said to be *near-optimal* for f_i if $df_i(A) - df_i^{min} < parameter \cdot (n - df_i^{min})$, where $0 < parameter \leq 1$ is given by the designer[3]. Then, the following heuristic algorithm solves the output/input partitioning problem as defined above.

1. `Let` $u = 0$, $F = \{f_1, \ldots, f_m\}$.
2. `For all` $A \in IP_p$ `and for all` $f_i \in F$, `compute` $df_i(A)$, df_i^{min}, `and the difference` $diff_i(A) = df_i(A) - df_i^{min}$ `between the number of decomposition functions required by a decomposition of` f_i `w.r.t.` A `and the minimum number of decomposition functions required by a decomposition of` f_i `w.r.t.` IP_p.
3. `If there is an` $f_i \in F$ `with` $df_i^{min} = n$, `i.e., which cannot be decomposed with respect to any partition of` IP_p `with less than` n `decomposition functions, then let`
   $Y_1 = \{f_i \in F;\ df_i^{min} = n\}$, $F = F \setminus Y_1$; $u = 1$.
   (These single-output functions will be decomposed by applying the well-known Shannon expansion which is a nontrivial decomposition[4] for $n \geq 4$.)
4. `Determine the input partition` A^* `such that` $\sum_{f_i \in F} diff_i(A^*)^L$ `is minimal.`
   (If $L = 1$ ($L = +\infty$), then A^* is the input partition for which the sum (maximum) of the deviations is minimal. L is a parameter of the heuristic.)
5. `Let` $u = u + 1$, `determine` Y_u `which is the set of those single-output functions` $f_i \in F$ `for which` A^* `is near-optimal, i.e.,` $diff_i(A^*) < parameter \cdot (n - df_i^{min})$, `and let` $F = F \setminus Y_u$.
6. `If` $Y_u = \emptyset$, `then let`
   $Y_u = \{f_j \in F;$ $diff_j(A^*)$ `is minimal`$\}$, $F = F \setminus Y_u$.
7. `If` $F \neq \emptyset$, `then goto 4.`

The running time of the algorithm strongly depends on the choice of subset IP_p of all possible input partitions $\{X, Y\}$ with $\mid X \mid = p$. For the computation of IP_p we use heuristics like iterative improvement (or simulated annealing) in the following way: start with any input partition A_0; form a new input partition A_{k+1} by exchanging a pair of variables in A_k as long as there is an $f_i \in F$ such that $df_i(A_k)$ can be decreased by such an exchange. Then, the set IP_p of the input partitions which are processed by the algorithm above consists of the input partitions $A_0, A_1, \ldots, A_k, \ldots$ computed.

3 Common decomposition functions

In this section we describe the second step of our algorithm. Suppose that the first step gives us a set Y_i of single-output functions which have to be decomposed with respect to the same input partition $A = \{X, Y\} \in IP_p$ which will be fixed in the following. In order to choose common decomposition functions in the decomposition of the elements of Y_i we have to solve the following subproblem, which we denote by CDF (**c**ommon **d**ecomposition **f**unctions problem):

Given: A set $Z = \{f_1, \ldots, f_l\}$ of single-output boolean functions[5], $\{X, Y\}$ with $X = \{x_1, \ldots, x_p\}$, $Y = \{y_1, \ldots, y_q\}$, and a natural number $h \leq r_i$ $(\forall i)$. ($r_i = \lceil \log p_1^{(i)} \rceil$), where $p_1^{(i)}$ is the number of distinct row patterns in the chart $M(f_i)$.)

Find: h single-output boolean functions $\alpha_1, \ldots, \alpha_h$, which can be used as decomposition functions of every single-output function f_i for $i = 1, \ldots, l$ such that there is a decomposition of the form

$$f_i(\mathbf{x}, \mathbf{y}) = g_i(\alpha_1(\mathbf{x}), \ldots, \alpha_h(\mathbf{x}), \alpha_{h+1}^{(i)}(\mathbf{x}), \ldots, \alpha_{r_i}^{(i)}(\mathbf{x}), \beta_1^{(i)}(\mathbf{y}), \ldots, \beta_{s_i}^{(i)}(\mathbf{y}))$$

$(\forall i \in \{1, \ldots, l\})$.

CDF can be posed for Y in an analogous manner.

Unfortunately, there is no great hope of finding an efficient algorithm solving CDF.

Lemma 1 *CDF is NP-complete.*
Proof: Reduction from the 3-partition problem [9].

We start by a theoretical result working towards a solution to CDF. It gives a condition necessary and sufficient that h single-output functions $\alpha_1, \ldots, \alpha_h$ are common decomposition functions of $f_1, \ldots, f_l$. It is a generalization

[2] If p is about $\frac{n}{2}$, the partition scheme tends to lead to fast tree circuits. If p is about 1 or $n-1$, it tends to lead to smaller and slower circuits.

[3] If *parameter* is chosen closer to 1, the probability for an input partition A to be accepted as 'near-optimal' for a function f_i increases, i.e., possibly f_i is decomposed with respect to an input partition which doesn't lead to a minimal number of decomposition functions. On the other hand the size of the sets Y_k then tends to be larger and we possibly have an increased potential for reusing subcircuits by choosing common decomposition functions for elements of Y_k.

[4] A decomposition is said to be *nontrivial* if its composition function g has less than n inputs.

[5] Regard Z as a subset of Y_i.

of a lemma shown by Karp [7]. For this, we need the following notations. The rows of $M(f_k)$ induce a partition of $\{0,1\}^p$ into equivalence classes $K_1^{(k)}, \ldots, K_{p_1^{(k)}}^{(k)}$ such that $v, v' \in \{0,1\}^p$ belong to the same class $K_j^{(k)}$ if and only if the two corresponding row patterns of $M(f_k)$ are equal. Let $\theta^{(k)} : \{0,1\}^p \to \{1, \ldots, p_1^{(k)}\}$ be the function where $\theta^{(k)}(v)$ is the index j of the class $K_j^{(k)}$ to which v belongs. Furthermore, for all $a \in \{0,1\}^h$, let $S_a^{(k)}$ be the set $\{\theta^{(k)}(v);\ \alpha_{1,\ldots,h}(v) = a\}$ of those classes which contain a row mapped to a by $\alpha_{1,\ldots,h}$[6]. Note that, for given $\alpha_{1,\ldots,h}$, $S_a^{(k)}$ and $S_{a'}^{(k)}$ need not to be disjoint for $a \neq a'$, and that $|S_a^{(k)}|$ equals the number of *distinct* row patterns of $M(f_k)$ mapped to a by $\alpha_{1,\ldots,h}(v)$. Let $dr(A, f_k, \alpha_{1,\ldots,h})$ be defined as $\max\{|S_a^{(k)}|;\ a \in \{0,1\}^h\}$.

Lemma 2 *$\alpha_1, \ldots, \alpha_h$ are decomposition functions of $f_1, \ldots, f_l$ with respect to A, i.e., there is a representation of f of the form*

$$f_k(\mathbf{x}, \mathbf{y}) = g_k(\alpha_1(\mathbf{x}), \ldots, \alpha_h(\mathbf{x}), \alpha_{h+1}^{(k)}(\mathbf{x}), \ldots, \alpha_{r_k}^{(k)}(\mathbf{x}), \beta_1^{(k)}(\mathbf{y}, \ldots, \beta_{s_k}^{(k)}(\mathbf{y}))$$

$(\forall k \in \{1, \ldots, l\})$ if and only if $dr(A, f_k, \{\alpha_1, \ldots, \alpha_h\}) \leq 2^{r_k - h} (\forall k)$.

Proof: A decomposition of the above form exists if and only if $(\alpha_{1,\ldots,h}, \alpha_{h+1,\ldots,r_k}^{(k)})$ assigns different values to rows of chart $M(f_k)$ with different row patterns $(\forall k)$. As $\alpha_{h+1,\ldots,r_k}^{(k)}$ can produce at most $2^{r_k - h}$ different values, the statement of the lemma follows. ■

Then CDF can be solved by computing $\alpha_{1,\ldots,h}$ by a branch and bound algorithm. The sets $S_a^{(k)}$ are constructed step by step. In the initialization phase, $\alpha_{1,\ldots,h}(x)$ is set to $undef$ for all $x \in \{0,1\}^p$, and $S_a^{(k)}$ is set to the empty set for all a and k. Each time we enter the main loop there is an $x \in \{0,1\}^p$ and a vector $value_f \in \{0,1\}^h$ such that $\alpha_{1,\ldots,h}(v)$ is defined for all v with $int(v) < int(x)$,[7] and there is no extension of the present function table with $int(\alpha_{1,\ldots,h}(x)) < int(value_f)$ which does not violate the condition of lemma 2. In this step, we test whether the condition of lemma 2 is violated if $\alpha_{1,\ldots,h}(x)$ is set to $value_f$. If the condition is violated, we have to backtrack if $int(value_f) = 2^h - 1$, i.e., $value_f = (1, \ldots, 1)$. If $int(value_f) < 2^h - 1$, enter the loop once again with $value_f$ incremented by 1. The sets $S_a^{(k)}$ are updated in each step.

Integration of CDF in the synthesis tool We apply the branch and bound algorithm solving CDF in an heuristic manner in order to solve the multilevel synthesis problem for multi-output boolean functions.

Let $f = (f_1, \ldots, f_m)$ be a multi-output function where each of the single-output functions f_k has to be decomposed with respect to the same input partition $A \in IP_p$.

[6] We write $f_{i,\ldots,j}$ for the function $(f_i, \ldots, f_j)$.

[7] $int(y)$ denotes the natural number represented by the boolean vector y

Circuit	Number of inputs	Number of outputs	No. of gates *factorII*	No. of gates *mulop*	ratio
9symm1	9	1	75	38	1.97
cm138a	6	8	21	21	1.00
cm151a	12	2	37	43	0.86
cm162a	14	5	80	44	1.82
cm163a	16	5	47	37	1.27
cm82a	5	3	18	16	1.13
cmb	16	4	33	29	1.14
decod	5	16	31	32	0.97
f51m	8	8	107	63	1.70
x2	10	7	65	47	1.38
z4m1	7	4	25	25	1.00

Table 1: Comparison between our tool *mulop* and *factorII* with respect to the number of gates used.

First, we apply CDF to compute (by binary search) a maximum number h of common decomposition functions of $f_1, \ldots, f_m$. Then, CDF is applied to subsets of $\{f_1, \ldots, f_m\}$ beginning with subsets of size $m-1$, $m-2$ down to size 2. During this iteration, decomposition functions already found for f_k are used in the decomposition of f_k in any case in order to reduce the running time of the multilevel synthesis tool. Note that the branch and bound algorithm can be generalized in a canonical manner for the case that some of the decomposition functions $\alpha_i^{(k)}$ $(i > h)$ are already predetermined.

Obviously, we have not to consider every subset of $\{f_1, \ldots, f_m\}$. First of all, if r_k decomposition functions are already found for function f_k, we can omit the remaining subsets containing f_k. Furthermore, it often happens that there is no common decomposition function for two single-output functions f_i and f_j. Of course, in this case, we have not to consider subsets containing f_i and f_j. Therefore, in a first step, for all f_i, f_j, we test (by dynamic programming) whether f_i and f_j have a common decomposition function, at all.

4 Benchmarking results

Several examples of the 1991 MCNC multilevel logic benchmark set were synthesized to compare *factor*, *factorII* [5, 6] and *misII* [2] to our tool, which we will call *mulop* in the following.

First, we compared our tool *mulop* to *factor* and *factorII*. We ran the experiments with the technology mapping used in [6]. Since the quality of the layouts synthesized by *factorII* approximately equals the quality of the layouts synthesized by factor (see [6]), table 1 only reports the results of the comparison of *mulop* and *factorII*. For every circuit, the entries of the table correspond to the minimal number of gates obtained by the synthesis tools. Compared to *factorII*, our approach generates realizations with a smaller (or equal) number of gates for almost all circuits considered.

Table 2 shows the comparison between *mulop* and *misII* with respect to the number of gates, the cell area, and

Circuit	No. of gates *misII*	*mulop*	ratio	Sum of the cell areas *misII*	*mulop*	ratio	Layout size *misII*	*mulop*	ratio	Signal delay *misII*	*mulop*	ratio
9symm1	175	40	4.38	426360	110048	3.87	917928	183840	4.99	26.0	16.6	1.56
C17	7	6	1.17	16568	14288	1.16	28800	25704	1.12	4.2	4.2	1.00
cm138a	22	20	1.10	59280	58368	1.02	101528	98784	1.03	5.8	4.6	1.26
cm151a	26	48	0.54	57456	115520	0.50	102528	194712	0.53	12.6	17.0	0.74
cm152a	24	34	0.70	52896	81776	0.65	90360	144008	0.63	10.0	14.6	0.68
cm162a	33	46	0.72	93480	121904	0.76	149736	210912	0.71	13.8	11.0	1.25
cm163a	32	38	0.84	82080	97584	0.84	153272	165000	0.93	11.0	11.8	0.93
cm82a	13	13	1.00	41304	38760	1.07	83104	63360	1.32	8.2	7.0	1.17
cm85a	40	42	0.95	98496	106704	0.92	171584	189472	0.91	10.2	13.4	0.76
cmb	48	24	2.00	112480	59584	1.89	198616	103896	1.91	14.8	9.0	1.64
decod	32	31	1.03	72352	72504	0.99	133496	129000	1.03	6.2	6.0	1.03
f51m	123	64	1.92	289560	164160	1.76	536016	280160	1.91	51.0	23.0	2.21
majority	9	9	1.00	23560	25384	0.93	42200	42224	1.00	7.8	7.8	1.00
parity	15	15	1.00	61560	61560	1.00	96976	96976	1.00	6.2	5.0	1.24
x2	45	38	1.18	103208	89072	1.16	180776	143864	1.26	13.0	12.4	1.04
z4m1	44	20	2.20	111264	57760	1.93	176160	101088	1.74	18.0	9.8	1.74

Table 2: Comparison between *mulop* and *misII* with respect to gate count, cell area, layout size, and signal delay.

the layout size[8]. The technology library used during this comparison consists of the set of the 2-input gates[9]. For about half of the benchmark set, our approach dominates that of *misII* with respect to layout size which is the crucial optimization criterion. The signal delays of our realizations for most of the circuits considered are better (or almost equal) than those of the realizations synthesized by *misII* (see table 2). However, on the other hand the results confirm the observation already made in [5, 6] that some circuits, e.g. *cm*151*a*, and *cm*152*a*, are not suited for being decomposed with respect to disjoint input partitions.

5 Conclusion

We have presented a multilevel logic synthesis tool based on communication complexity which eliminates the drawbacks of similar approaches known from literature with respect to multi-output functions. Our method consists in two steps: output/input partitioning and constructions of decomposition functions while paying special attention that many of them can be used in the realization of different outputs.

The benchmarking results comparing our tool to *misII*, *factor* and *factorII* are promising both in respect to gate count and signal delay.

The running time of the tool itself (measured on a SPARC2) was much better than that of *factor* without however coming up to the excellent running time of *factorII*. Here, we are investing further work, especially, using BDDs in the implementation of our synthesis tool.

[8] The layouts were generated by *wolfe* which is integrated in *octtools*.

[9] The 2-input gates are taken from *stdcell2_2.genlib* available in *octtools*.

References

[1] R.L. Ashenhurst. The decomposition of switching functions. In *Proceedings on an International Symposium on the Theory of Switching* held at Comp. Lab. of Harvard University, pages 74–116, 1959.

[2] R.K. Brayton, R. Rudell, A. L. Sangiovanni-Vincentelli, and A.R. Wang. MIS: A multiple-level logic optimization system. *IEEE Trans. on CAD*, CAD-6(11), November 1987.

[3] H.A. Curtis. A generalized tree circuit. *J. Assoc. Comput. Mach.*, 8:484–496, 1961.

[4] G. Hotz. Zur Reduktionstheorie der booleschen Algebra. In *Colloquium über Schaltkreis- und Schaltwerk-Theorie*, 1960.

[5] T. Hwang, R.M. Owens, and M.J. Irwin. Exploiting communicaton complexity for multilevel logic synthesis. *IEEE Trans. on CAD*, CAD-9(10):1017–1027, October 1990.

[6] T. Hwang, R.M. Owens, and M.J. Irwin. Efficient computing communication complexity for multilevel logic synthesis. *IEEE Trans. on CAD*, CAD-11(5):545–554, May 1992.

[7] R.M. Karp. Functional decomposition and switching circuit design. *Journal of Society of Industrial Applied Mathematics*, 11(2):291–335, June 1963.

[8] Y. Lai, M. Pedram, and S. Vrudhula. BDD based decomposition of logic functions with application to FPGA synthesis. In *Proceedings of DAC'93*, pages 642–647, 1993.

[9] Chr. Scholl and P. Molitor. Mehrstufige Logiksynthese durch Ausnutzung von Symmetrien und nichttrivialer Zerlegungen. Technical report, FB Informatik, Universiät des Saarlandes, FRG, 1993.