[11] Y. Lai, M. Pedram, and S. Vrudhula. BDD based decomposition of logic functions with application to FPGA synthesis. In *ACM/IEEE Design Automation Conference (DAC93)*, pages 642–647, 1993.

[12] P. Molitor and Ch. Scholl. Mehrstufige Logiksynthese unter Ausnutzung von Symmetrien und nichttrivialen Zerlegungen. Technical Report TR-02/1993, Sonderforschungsbereich 124 *VLSI Entwurfsmethoden und Parallelität*, Fachbereich Informatik, Universität des Saarlandes, Im Stadtwald, W–6600 Saarbrücken 11, FRG, 1993.

[13] J. Slansky. Conditional-sum addition logic. *IEEE Transactions on Electronic Computers*, EC-9:226–231, 1960.

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, C.T. McMullen G.D. Hachtel, and A.L. Sangiovanni-Vincentelli. *Logic Minimization Algorithms for VLSI Synthesis*. The Kluwer International Series in Engineering and Computer Science. Kluwer Academic Publishers, 1984.

[3] R.K. Brayton, G.D. Hachtel, and A. L. Sangiovanni-Vincentelli. Multilevel logic synthesis. *Proceedings of the IEEE*, 78(2):264–300, February 1990.

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

[5] S. Chang and M. Marek-Sadowska. BDD representation of incompletely specified functions. In *Notes of the International Workshop on Logic Synthesis held in Tahoe City, California*, 1993.

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

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

[8] T. Hwang, R.M. Owens, and M.J. Irwin. Exploiting communication complexity for multilevel logic synthesis. *IEEE Transactions on Computer Aided Design*, CAD-9(10):1017–1027, October 1990.

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

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

Circuit	No. of gates			Sum of the cell areas			Layout size			Signal delay		
	misII	*mulop*	ratio	*misII*	*mulop*	ratio	*misII*	*mulop*	ratio	*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
b1	4	9	0.44	12160	20520	0.59	24000	33600	0.71	3.0	5.6	0.53
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.

in the realization of different outputs.

The benchmarking results are promising. However, they also confirm the observation already made in [8, 9] that some boolean functions seem to be not suited for being decomposed with respect to disjoint input partitions. This is the reason why, for certain benchmarks, *misII* produces smaller circuits than our approach. Nevertheless, for nearly almost benchmarks our tool leads to better realizations than *factor* does. For every benchmark we have considered, 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.

Our system will be extended so that it will make use of 'don't cares'. As shown in [5, 11, 12], this problem is equivalent to the problem of partitioning graphs into a minimum number of cliques. Thus, we can apply the matured heuristics for the clique cover problem during multilevel synthesis of partially defined boolean functions.

Circuit	Number of inputs	Number of outputs	No. of gates *factor*	No. of gates *mulop*	ratio
9symm1	9	1	82	38	2.15
C17	5	2	11	6	1.83
b1	3	4	8	8	1.00
cm138a	6	8	19	21	0.90
cm151a	12	2	55	43	1.28
cm152a	11	1	30	29	1.03
cm162a	14	5	89	44	2.02
cm163a	16	5	49	37	1.32
cm82a	5	3	20	16	1.25
cm85a	11	3	56	44	1.27
cmb	16	4	31	29	1.07
majority	5	1	12	12	1.00
parity	16	1	30	40	0.75
z4m1	7	4	26	25	1.04

Table 1: Comparison between our tool *mulop* and *factor* with respect to the number of gates used. The technology file used is taken from the paper of Hwang et al.

consists of the set of the 2-input gates.[‡]

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) although the layout sizes of half of the benchmark set are smaller (or equal) than those of *misII*. However, on the other hand the results confirm the observation already made in [8, 9] that some circuits, e.g., $b1$, $cm151a$, and $cm152a$, are not suited for being decomposed with respect to disjoint input partitions.

7 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

[‡]For the technology file itself see *stdcell2_2.genlib* available in *octtools*.

uses another code than f_6 and f_7. The reason is that f_5 can use the identical decomposition function as f_4.

6 Benchmarking results

Several examples of the 1991 MCNC multilevel logic benchmark set were synthesized to compare *factor* [8, 9] and *misII* [4] to our tool, which we will call *mulop* in the following.

First, we compared our tool *mulop* to *factor*. We ran the experiments with the technology mapping used in [8]. The technology file can be found in [8]. A comparison to *factorII* was not possible because the corresponding technology file has not been specified in [9]. However, a comparison of *factor* and *factorII* is done in [9] showing that the quality of the layouts synthesized by *factorII* approximately equals the quality of the layouts synthesized by *factor*. However, the running time of *factorII* is much better than that of *factor*. Table 1 summarizes the results of the comparison of *mulop* and *factor*. For every circuit, the entries of the table correspond to the minimal number of gates obtained by the synthesis tools. Compared to *factor*, our approach generates realizations with a smaller (or equal) number of gates for every circuit but $cm138a$ and *parity*. The improvement the most dramatic has been reached for circuit $9symm1$ which is a symmetric function. It confirms the approach of searching simple decomposition functions as explained in section 4.1. The circuit *parity* is a symmetric function, too. The reason why the number of gates of the realization synthesized by our approach is greater than the number of gates of the realization synthesized by *factor* is due to the technology mapping used, because *mulop* (without technology mapping) realizes *parity* by a balanced *exor*-tree which is optimal at the design level considered.

As layout considerations motivate the approach of minimizing communication complexity, we compared layout sizes in the following. A comparison to *factor* with respect to cell area and layout size was not possible because Hwang et al. used in their paper a layout tool named *artist* which was not at our disposal. We used the standard cell place and route package *wolfe* which is integrated in *octtools*. Table 2 shows the comparison between *mulop* and *misII* with respect to the number of gates, the cell area, and the layout size. The technology library used

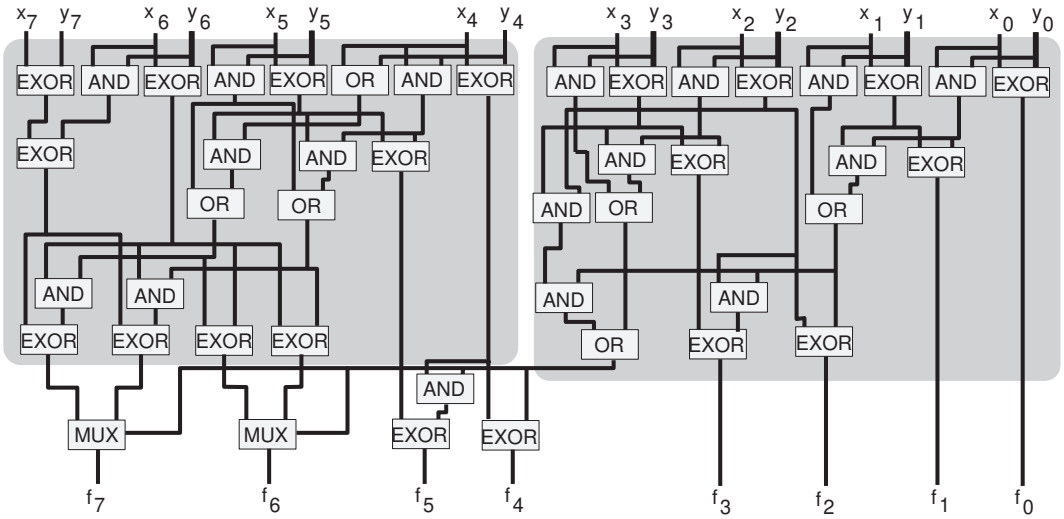

Figure 1: Realization of an 8-bit adder by using balanced input partitions. The dark faces illustrate the decomposition functions of the first recursion level.

input partitions. By having a closer look at figure 1, one recognizes that its overall structure is that of the well-known conditional-sum adder (CSA) [13]. Our adder consists of 49 2-input gates (if a multiplexer is realized by 3 2-input gates) whereas CSA requires 90 2-input gates. The depth of both adders is 7.

In the first recursion level, the algorithm partitioned the inputs into the sets $X_1 = \{x_i, y_i;\ 0 \leq i \leq 3\}$ and $X_2 = \{x_j, y_j;\ 4 \leq j \leq 7\}$. There is one decomposition function defined on X_1, namely the carry function of the binary addition of $x_3x_2x_1x_0$ and $y_3y_2y_1y_0$, which is used as decomposition function of f_4, f_5, f_6 and f_7.

The main difference between our adder and CSA is that, for every subfunction f_i, the algorithm tries to realize f_i with the minimum number of decomposition functions. For illustration, consider the decomposition of f_4 on the first recursion level. f_4 is realized by using only one decomposition function defined on X_2 whereas CSA uses two decomposition functions, namely that computing the less significant bit of the binary addition of $x_7x_6x_5x_4$ and $y_7y_6y_5y_4$ and that computing the less significant bit of the binary addition of $x_7x_6x_5x_4$, $y_7y_6y_5y_4$, and 1. Both CSA and the adder generated by our tool use two decomposition functions defined on X_2 for the decomposition of f_5, f_6 and f_7 which encode four different informations. CSA always uses the same encoding scheme, namely the 4 informations are encoded by the corresponding bits of the sum and the sum plus 1. Our adder changes the encoding scheme. Function f_5

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 whether f_i and f_j have a common decomposition function, at all.

Another aspect of our synthesis algorithm will only be sketched: Functions which possess special properties like independence of some variables or symmetry in some variables often occur in practical applications and often have 'simple' realizations. As the decomposition functions $\alpha_1^{(k)}, \ldots, \alpha_{r_k}^{(k)}$ of such a single-output function f_k are allowed to assign different values to rows with identical row pattern, they do not need to have these special properties too. In order to compute 'simple' decomposition functions, we are first looking for decomposition functions which assign an identical value to all those $v \in \{0,1\}^p$ for which the corresponding row patterns of $M(f_k)$ are identical. Thus, we first make use of a modified branch and bound algorithm to compute common decomposition functions, which have this property. Subsequent to this step, the original branch and bound algorithm is applied to compute the lacking decomposition functions. The modified branch and bound algorithm has still another advantage concerning the running time of the synthesis tool. Common decomposition functions $\alpha_1, \ldots, \alpha_h$ of $f_1, \ldots, f_m$ have to assign an identical value to v and $v' \in \{0,1\}^p$ whenever there is a $k \in \{1, \ldots, m\}$ such that the rows of $M(f_k)$ corresponding to v and v' have identical row patterns. Note that maximal subsets $E_1, \ldots, E_l \subseteq \{0,1\}^p$ can easily be computed such that $\alpha_1, \ldots, \alpha_h$ have to assign an identical value to each $v \in E_i$. Thus, the modified branch and bound algorithm assigns values to the subsets $E_1, \ldots, E_l$. Because l mostly is much smaller than 2^p, this approach considerably reduces the running time compared to the original branch and bound algorithm described in the subsection above.

5 An efficient adder generated by the synthesis tool above

We have applied the algorithm above to several multi-output boolean functions. An overview of the benchmarking results can be found in section 6. The example we comment in detail for illustration is the 8-bit adder generated by our synthesis tool when considering balanced

$$\alpha_{h+1}^{(k)}(x_1,\ldots,x_p),\ldots,\alpha_{r_k}^{(k)}(x_1,\ldots,x_p),x_{p+1},\ldots,x_n)$$

($\forall k \in \{1,\ldots,m\}$) 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. q.e.d.

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 $int(v) < int(x)$,[†] 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.

4.1 Integration of CDF in the multilevel 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 $A \in IP_p$.

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.

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

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.

Note that the decomposition algorithm is recursively applied to the decomposition functions and the composition functions. This makes step 3 of the algorithm reasonable as subfunctions obtained by Shannon expansion often are nontrivially decomposable even though the function itself is not nontrivially decomposable.

4 An algorithm for CDF

We now concentrate on the problem of finding common decomposition functions. In the following, let $f = \{f_1, \ldots, f_m\} \in B_{n,m}$ be a multi-output function. Each of the single-output functions f_k has to be decomposed with respect to the same given input partition $A = \{X_1, X_2\} \in IP_p$ which will be fixed in the following.

We start by a theoretical result working towards a solution to CDF. It gives a condition necessary and sufficient that h single-output functions $\alpha_1, \ldots, \alpha_h \in B_p$ are common decomposition functions of $f_1, \ldots, f_m$. It is a generalization of a lemma shown by Karp [10]. 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 $\theta^{(k)} : \{0,1\}^p \rightarrow \{1, \ldots, p_1^{(k)}\}$ 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 $\alpha_{1,\ldots,h}$. 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 *$\alpha_1, \ldots, \alpha_h \in B_p$ are decomposition functions of $f_1, \ldots, f_m$ with respect to A, i.e., there is a representation of f of the form*

$$f_k(x_1, \ldots, x_n) \;=\; g^{(k)}(\alpha_1(x_1, \ldots, x_p), \ldots, \alpha_h(x_1, \ldots, x_p),$$

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. 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* 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 heavily depends on the choice of subset IP_p of all possible input partitions $\{X_1, X_2\}$ with $|\ X_1\ | = p$. For the computation of IP_p we use heuristics like

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

The values p and *parameter* are determined by the designer. p defines the size $|X_1^{(k)}|$ of $X_1^{(k)}$ $(\forall k)$. 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. The value of *parameter* determines the notion of 'near-optimal' and, thus, influences the value of u (see section 3).

After output partitioning, we have to compute decomposition functions of f which are used by different single-output functions f_i of a class Y_k. Thus, we have to pose the following problem which we denote by CDF (**c**ommon **d**ecomposition **f**unctions problem).

Given: Let $f=\{f_1,\ldots,f_m\}\in B_{n,m}$, $\{X_1,X_2\}$ with $X_1=\{x_1,\ldots,x_p\}$ and $X_2=\{x_{p+1},\ldots,x_n\}$ an input partition, and h be a natural number with $h \leq r_i$ $(=\lceil \log p_1^{(i)} \rceil)$ $(\forall i)$.

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

$$\begin{aligned} f_i(x_1,\ldots,x_n) &= g^{(i)}(\alpha_1(x_1,\ldots,x_p),\ldots,\alpha_h(x_1,\ldots,x_p), \\ &\qquad \alpha_{h+1}^{(i)}(x_1,\ldots,x_p),\ldots,\alpha_{r_i}^{(i)}(x_1,\ldots,x_p),x_{p+1},\ldots,x_n) \end{aligned}$$

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

CDF can be posed for X_2 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 [12].

3 A heuristic for OP

In this section, we present a heuristic which partitions $F=\{f_1,\ldots,f_m\}$ in disjoint sets $Y_1,\ldots,Y_u$. To each set Y_k, the algorithm assigns an input partition $\{X_1^{(k)},X_2^{(k)}\}$ which is 'near-optimal' for each $f_i \in Y_k$.

Let IP_p be a subset of all possible input partitions $\{X_1,X_2\}$ with $\mid X_1 \mid = p$ 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

$\{f_i, \ldots, f_j\}$.

As already mentioned, the first step of the algorithm partitions $\{f_1, \ldots, f_m\}$ into disjoint sets $Y_1, \ldots, Y_u$. Single-output functions of the same set Y_k will be decomposed with respect to the same input partition. Thus, we need the following definition.

Definition 1 *A decomposition of a multi-output boolean function $f \in B_{n,m}$, with respect to the input partition $\{X_1, X_2\}$ ($X_1 = \{x_1, \ldots, x_p\}$, $X_2 = \{x_{p+1}, \ldots, x_{p+q}\}$, $p+q=n$) is a representation of f of the form*

$$\begin{aligned} f_i(x_1, \ldots, x_n) \;=\; & g^{(i)}(\alpha_1^{(i)}(x_1, \ldots, x_p), \ldots, \alpha_{r_i}^{(i)}(x_1, \ldots, x_p), \\ & \beta_1^{(i)}(x_{p+1}, \ldots, x_{p+q}), \ldots, \beta_{s_i}^{(i)}(x_{p+1}, \ldots, x_{p+q})) \end{aligned}$$

($\forall i \in \{1, \ldots, m\}$), where $\alpha_k^{(i)} \in B_p$ ($\forall i, k$), $\beta_j^{(i)} \in B_q$ ($\forall i, j$), and $g^{(i)} \in B_{r_i+s_i}$. $\alpha_k^{(i)}$ ($\forall i, k$) and $\beta_j^{(i)}$ ($\forall i, j$) are called decomposition functions of f. *$g^{(i)}$ ($\forall i$) is called* composition function.

With respect to a given input partition $\{X_1, X_2\}$, a single-output function f_i can be represented as a $2^p \times 2^q$ matrix $M(f_i)$, the *decomposition matrix of f_i* or the *chart of f_i* with respect to $\{X_1, X_2\}$. Each row and column of $M(f_i)$ is associated with a distinct assignment of values to the inputs in X_1 and X_2, respectively, such that $f_i(X_1, X_2) = M(f_i)[X_1, X_2]$ where $M(f_i)[X_1, X_2]$ represents the element of $M(f_i)$ which lies in the row associated with X_1 and the column associated with X_2. Then the minimum number r_i of interconnections between $\alpha^{(i)}$ and $g^{(i)}$ is $\lceil \log p_1^{(i)} \rceil$ where $p_1^{(i)}$ is the number of distinct row patterns in $M(f_i)$. Likewise, the minimum number s_i of interconnections between $\beta^{(i)}$ and $g^{(i)}$ is $\lceil \log p_2^{(i)} \rceil$ where $p_2^{(i)}$ is the number of distinct column patterns.

Then, the **o**utput **p**artitioning problem (OP) is defined as follows.

Given: Let $f = \{f_1, \ldots, f_m\} \in B_{n,m}$, p be a natural number with $1 \leq p \leq n-1$, and *parameter* a real number with $0 < \mathit{parameter} \leq 1$.

Find: a partition of $\{f_1, \ldots, f_m\}$ in disjoint sets $Y_1, \ldots, Y_u$ such that, for all $k \in \{1, \ldots, u\}$, there is an input partition $\{X_1^{(k)}, X_2^{(k)}\}$ where the size of $X_1^{(k)}$ equals p, and which is 'near-optimal' for every $f_i \in Y_k$, i.e., for which there exists a decomposition of every $f_i \in Y_k$ where $r_i + s_i$ is 'near-minimal'.

partition good for both f_i and f_j. Furthermore, even in case that there is an input partition good for every single-output function of f, it is unlikely that there is a decomposition where function g has much less inputs than f (if m is large enough), i.e., g is not much easier than f to synthesize. The drawback of the second method is clear. In the final analysis, it does not use the potential of reusing subcircuits for different outputs of f.

In this paper we present a multilevel synthesis method for multi-output boolean functions based on communication complexity which avoids both drawbacks. The method can be divided into two steps. In the first step, output partitioning is performed, i.e., $\{f_1, \ldots, f_m\}$ is partitioned into disjoint sets $Y_1, \ldots, Y_u$. Single-output functions f_i and f_j of the same set Y_k will be decomposed with respect to the same input partition. The partitioning is executed such that for every Y_k there is an input partition which is 'near-optimal' for every $f_i \in Y_k$. The crucial point is the choice of u. If u is about n, each set Y_k only consists of 1 or 2 single-output functions so that there are no great possibilities of finding common decomposition functions. If u is too small, many of the single-output functions have to be decomposed with respect to the same input partition. We meet with this problem by introducing a variable *parameter* which is set by the designer, and which defines the notion 'near-optimal' above and thus determines value u. In the second step, a branch and bound algorithm constructs the decomposition functions of the single-output functions of each class Y_k 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.

Experimental results close the paper. They are promising. In particular, we present an 8-bit adder constructed by our algorithm which has the overall structure of the well-known conditional-sum adder [13], but which uses different encodings in different levels resulting in less decomposition functions, and thus, in a smaller number of gates although the optimization goal used is not gate count but communication complexity.

2 Basic definitions

A multi-output boolean function f with n inputs is represented as a set $\{f_1, \ldots, f_m\}$ of boolean-valued output functions. We denote the set of functions with n inputs and m outputs by $B_{n,m}$. Let B_n be an abbreviation for $B_{n,1}$. $f_{i,\ldots,j}$ $(i \leq j)$ denotes the multi-output function

1 Introduction

The long term goal for logic synthesis is the automatic transformation from a behavioral description of a boolean function to near-optimal netlists, whether the goal is minimum delay, minimum area, or some combination. The logic synthesis area is usually divided into two-level synthesis and multilevel synthesis. Solutions to the two-level logic minimization problem have matured and are used to synthesize PLAs for control logic (see e.g. [2]). Synthesizing multilevel logic (which is useful for both control and data-flow logic) at a level competitive with manual synthesis is much more difficult than in the PLA case because of the increased potential for reusing sublogic.

Most of the approaches attacking the multilevel logic synthesis problem use gate count as optimization criterion. A survey can be found in [3]. 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 [8, 9, 11] propose an approach different from the one addressed above. This approach to multilevel logic synthesis which originates from Ashenhurst [1], Curtis [6], Hotz [7], and Karp [10] is based on minimizing communication complexity. The methods used to reduce communication complexity employ functional decomposition, i.e., given a boolean function $f : \{0,1\}^n \rightarrow \{0,1\}$ they are looking for (multi-output) functions α, β und g, such that $f(x_1, \ldots, x_n) = g(\alpha(x_1, \ldots, x_p), \beta(x_{p+1}, \ldots, x_n))$ holds for all $(x_1, \ldots, x_n) \in \{0,1\}^n$. Applying functional decomposition recursively to the decomposition functions α and β demands a generalization to multi-output boolean functions. Of course the approaches of the papers above can also be applied to multi-output boolean functions $f = (f_1, \ldots, f_m) : \{0,1\}^n \rightarrow \{0,1\}^m$ either by considering multi-output boolean functions as single-output multi-value functions $f' : \{0,1\}^n \rightarrow \{0, \ldots, 2^m - 1\}$ defined by $f'(x_1, \ldots, x_n) = \sum_{i=1}^{m} f_i(x_1, \ldots, x_n) 2^{i-1}$ or by decomposing each single-output boolean function f_i independently of the other single-output functions f_j $(j \neq i)$ and only then testing whether they use some identical decomposition functions by accident. The first method has the drawback that it decomposes each single-output function f_i with respect to the same input partition which can result in poor realizations. It does not take into consideration that there possibly exist single-output functions f_i and f_j such that there does not exist an input

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, 8, 9, 10, 11] 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.

Key Words

Combinational Logic Synthesis, Multi-output Boolean Functions

An extended abstract of the paper will appear in the *Proceedings of the 4th Great Lakes Symposium on VLSI*, Notre Dame, Indiana, March 4–5, 1994.

Research has been supported in part DFG grant SFB 124 *Entwurfsmethoden und Parallelität* and the *Graduiertenkolleg* of the University of Saarbrücken.

Communication based multilevel synthesis for multi-output boolean functions

Paul Molitor Christoph Scholl

Informatik–Bericht Nr. 29
Humbold–Universität zu Berlin