

Exploiting Communication Complexity for Multilevel Logic Synthesis

TING-TING HWANG, ROBERT MICHAEL OWENS, AND MARY JANE IRWIN, SENIOR MEMBER, IEEE

Abstract—A multilevel logic synthesis technique based on minimizing communication complexity is presented. Intuitively, this approach is believed viable because for many types of circuits the area needed is dominated by interconnections. By minimizing communication complexity and interconnect, thus area is reduced. This approach performs especially well for functions that are hierarchically decomposable (e.g., adders, parity generators, comparators, etc.). Unlike many other multilevel logic synthesis techniques, a lower bound can be computed to determine how well the synthesis was performed. Also presented is a new multilevel logic synthesis program based on the techniques described for reducing communication complexity.

I. INTRODUCTION

LOGIC SYNTHESIS can be viewed as the transformation of a functional (i.e., algebraic) specification into a gate (i.e., netlist) specification. It is relatively straightforward to transform a functional specification into a nonoptimal gate specification. The difficulty arises when an optimal, near optimal, or probabilistically optimal (i.e., optimal with some probability) gate description is desired. The optimal gate specification is the one which can be used to synthesize an optimal (i.e., smallest) layout. Thus to produce truly optimal layouts, logic synthesis cannot be divorced from layout synthesis. However, for purely pragmatic reasons it is often desirable to separate logic synthesis and layout synthesis into distinct design steps. If this is done, criteria other than actual layout area must be used to define gate specification optimality. Historically, gate count has been used. This is based on the belief that gate count is a good estimator for layout area; that is, the smaller the gate count, the smaller the layout. Unfortunately, in many cases, this and other such criteria may not be the best estimators.

For two-level logic synthesis, tools like *espresso* [4] can be used. Approaches that have been advocated for multilevel synthesis include 1) algebraic decomposition of logic functions to a multilevel design [5]; 2) translation using a number of rewriting rules [7], [11]; and 3) exhaustive search [2]. While none of these approaches is a complete panacea, each has its place in the design process. For example, several available multilevel logic syn-

thesis tools (e.g., *misII* [5] and *bold* [3]) cannot handle certain types of functions well [15]. While our exhaustive search tool, *logician* [2], produces optimal gate specifications that can be used to synthesize good layouts, it is useful only for relatively small circuits (up to six inputs and six outputs).

We propose in this paper an approach to multilevel logic synthesis based on minimizing communication complexity. This approach is different from any of those mentioned in the previous paragraph. Intuitively, we believe this approach should be considered for the following reason. For many types of circuits, lower bounds on the area to implement those circuits have been obtained considering only communication complexity [9], [16], [19], [22]. Hence, it would appear that at least for some types of circuits communication complexity could be a good area estimator; that is, the smaller the communication complexity, the smaller the layout. The method we use to reduce the communication complexity employs functional decomposition. This decomposition approach is not new. It has been considered before [1], [6], [8], [13], [18], [20].

We first overview our definition of communication complexity as it relates to the logic synthesis problem. The goal is to find a partitioning of the inputs which results in the fewest number of interconnections between subcircuits. Several partitioning schemes are presented and compared. Using one of the partitioning schemes proposed in the second section, the third section discusses various implementation issues. In particular, partitionings where the partitioned sets are of equal size (which tends to lead to fast tree circuits) and partitionings where one of the partitioned sets has bounded size (which tends to lead to smaller, slower, serial circuits) are discussed. Finally, we present some results for our prototype multilevel logic synthesis tool, *factor*, which is designed to reduce communication complexity.

II. COMMUNICATION COMPLEXITY

Succinctly, the problem of logic synthesis can be stated as follows. Given a function

$$y = f(x) \quad f: Z^n \rightarrow Z^m \quad Z = \{0, 1\}$$

where the elements of x are the n binary inputs, the elements of y are the m binary outputs, and f is a function that maps the input vector into the output vector, find a circuit C which computes this function. The problem of

Manuscript received November 1, 1988; revised July 21, 1989. This work was supported in part by the National Science Foundation under Grant MIP-8701367. This paper was recommended by Associate Editor R. K. Brayton.

The authors are with the Department of Computer Science, Pennsylvania State University, University Park, PA 16802.

IEEE Log Number 9036679.

finding the circuit C can be decomposed into a number of subproblems in the following way. First, partition the inputs x into two nonoverlapping sets, x_l and x_r . Then, for this partitioning, find the functions $\tilde{f}$, f_l , and f_r such that

$$f(x) = \tilde{f}(f_l(x_l), f_r(x_r)) \quad f_l: Z^{n_l} \rightarrow Z^{m_l}$$

$$f_r: Z^{n_r} \rightarrow Z^{m_r} \quad \tilde{f}: Z^{m_l} \times Z^{m_r} \rightarrow Z^m.$$

Trivially, such functions exist. Next, find circuits, L , R , and T , which compute f_l , f_r , and $\tilde{f}$, respectively. Finally, interconnect these circuits as illustrated in Fig. 1.

An interesting and obviously important question now arises. For a given partitioning π of the inputs, what is the minimum number of interconnections which must exist between circuit L and circuit T and between circuit R and circuit T ? The partitioning with the minimum number of interconnections should lead to an optimal design based on communication complexity. Since we are considering combinational circuits, each interconnection represents the ability to transfer 1 b of information.

This question can be answered exactly in the following way. With respect to a given partitioning π , the function f can be represented as a $2^{n_l} \times 2^{n_r}$ matrix M . Each row of the matrix M is associated with a distinct assignment of values to the inputs in x_l , and each column is associated with a distinct assignment of values to the inputs in x_r , such that

$$f(x_l, x_r) = M[x_l, x_r]$$

where $M[x_l, x_r]$ represents the element of M which lies in the row associated with x_l and the column associated with x_r . Then the minimum number of interconnections between circuit L and T is $\log_2(p_l)$, where p_l is the number of distinct row patterns [16]. Likewise, the minimum number of interconnections between circuit R and T is $\log_2(p_r)$, where p_r is the number of distinct column patterns.

For example, consider the comparison function as defined by

$$z = f(x_1, x_2, x_3, x_4, y_1, y_2, y_3, y_4)$$

$$= \begin{cases} 1, & \text{if } x_i \equiv y_i, \quad i = 1, 2, 3, 4 \\ 0, & \text{otherwise.} \end{cases}$$

Suppose the inputs are partitioned into the sets (x_1, x_2, x_3, x_4) and (y_1, y_2, y_3, y_4) . Then the comparison function can be represented as the $2^4 \times 2^4$ matrix M given in Fig. 2, where (x_1, x_2, x_3, x_4) and (y_1, y_2, y_3, y_4) are the row index and column index, respectively.

Matrix M has 16 different row patterns and 16 different column patterns. Hence, if the problem of finding a circuit that computes the comparison function was subdivided using this partitioning, there must be at least four (i.e., $\log(16)$) interconnections between circuit L and circuit T , and at least four (i.e., $\log(16)$) interconnections between circuit R and circuit T .

Suppose instead the inputs are partitioned into sets (x_1, y_1, x_2, y_2) and (x_3, y_3, x_4, y_4) . Then the comparison func-

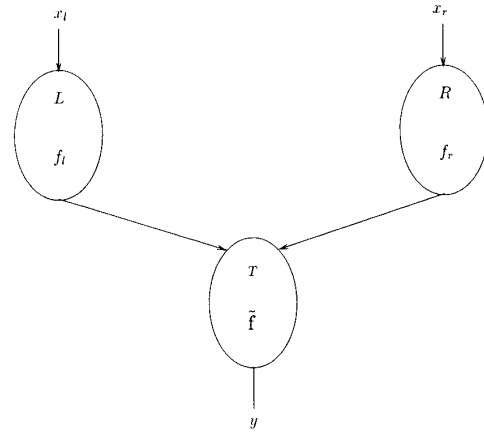


Fig. 1. Simple decomposition.

1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Fig. 2. Input partitioning with many different patterns.

1	0	0	1	0	0	0	0	0	0	0	0	1	0	0	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	1	0	0	0	0	0	0	0	0	1	0	0	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	1	0	0	0	0	0	0	0	0	1	0	0	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	1	0	0	0	0	0	0	0	0	1	0	0	1

Fig. 3. Input partitioning with few different patterns.

tion can be represented as the matrix M given in Fig. 3, where (x_1, y_1, x_2, y_2) and (x_3, y_3, x_4, y_4) are the row and column indexes, respectively.

Matrix M now has only two different row patterns and two different column patterns. Hence, if the problem of

finding a circuit which computes the comparison function was subdivided using this partitioning, there could be as few as one (i.e., $\log(2)$) interconnection between circuit L and circuit T , and as few as one (i.e., $\log(2)$) interconnection between circuit R and circuit T . The question of theoretical interest has been: for some function f , what is the fewest number of interconnections which result from any partitioning? However, not unexpectedly, the question of interest to us is: for some function f , find a partition π that results in the fewest number of interconnections.

By enumerating all possible partitions, the procedure just outlined gives an exact lower bound on the number of interconnections. It is sometimes desirable and sufficient to use a procedure which is computationally more efficient but which may not achieve the absolute fewest number of interconnections, as will be explained in the next section. One such procedure is outlined as follows. With respect to some algebra $G = \langle +, \times \rangle$, the matrix M can be factored into three matrices:

$$M = LDR$$

where D is a diagonal matrix. Trivially, if there is a multiplicative identity in G , such factorings exist for M . Among all the factorings, there is at least one factoring where D has the same rank as M . Determination of such a factoring can proceed as follows. Initially, $D = M$, and L and R are identity matrices. We use the following row and column operations to cancel out all nonzero entries in D such that the product of LDR will remain the same.

1) *Row Operation:*

$$LMR = L'M'R$$

where the j th row of $M' = (j$ th row of $M) - a \times (i$ th row of $M)$ and the i th column of $L' = (i$ th column of $L) + a \times (j$ th column of $L)$.

2) *Column Operation:*

$$LMR = LM'R'$$

where the j th column of $M' = (j$ th column of $M) - a \times (i$ th column of $M)$ and the i th row of $R' = (i$ th row of $R) + a \times (j$ th row of $R)$.

For example, the row operation can be illustrated in Fig. 4.

We repeat the row and column operations until the diagonal matrix D is obtained. The preceding computation is similar in nature to Gaussian elimination.

Now, using some algebraic manipulation, we can obtain

$$M = LDR = \sum_{i=1}^k l_i r_i$$

where $\text{diag}(D) = \{1, 1, \dots, 1, 0, 0, \dots, 0\}$ with k 1's, where k is the rank of M and $l_i(r_i)$ is the i 'th column of L (the i 'th row of R). This is the factoring that minimizes k . For a minimum factoring, the l_i 's and the r_i 's are distinct. For example, the matrix M which represents the comparison function of two 2-b numbers with partitioned

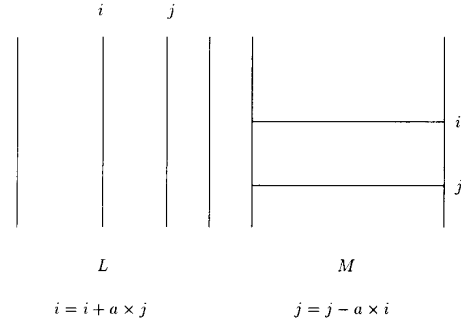


Fig. 4. Row operation.

sets (x_0, y_0) and (x_1, y_1) can be decomposed as follows:

$$M = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

(LMR)

$$= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

(row operation)

$$= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

(column operation; LDR)

$$= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} [1 \ 0 \ 0 \ 0]$$

$$\cdot \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (\text{algebraic manipulation})$$

$$= \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \end{bmatrix} [1 \ 0 \ 0 \ 1] \quad (l_1 r_1).$$

was assigned arbitrarily. The resulting subfunctions might be very complicated. This work was generalized by Karp [13]. Karp used a procedure to assign the communication code such that the resulting subfunctions fell into certain classes (e.g., unate functions, threshold functions, etc.). However, in general it is inefficient to carry out this procedure. Also, the subfunctions found are still not simple enough. Nevertheless, the exact procedure can be used to compute the lower bound of communication that is necessary, which in turn can be used to determine how well the factoring-based procedure performs. The factoring-based procedure is computationally more efficient, and the subfunctions derived are simpler. However, it may not achieve a minimum number of interconnections.

To use the factoring-based procedure two problems must be resolved. These problems are 1) how to determine a best partition and 2) for a given function f and partitioning π , how to find the functions $\tilde{f}$, f_l , and f_r . We will first consider the problem of finding a best partition. While the set of input variables can be partitioned many (i.e., exponentially) different ways, we are not interested in all possible partitionings. In this paper, we will consider only nonoverlapping partitionings where the partitioned sets are of equal size (which tends to lead to fast tree circuits), or where one of the partitioned sets has bounded size (which tends to lead to smaller, slower, serial circuits).

We will first consider the case where the inputs are partitioned into two partitions of approximately equal size. In this case, a balanced hierarchically structured circuit can be created by recursively decomposing the leaf circuits. For example, recursively decomposing the circuit with respect to the algebra $G = \langle \wedge, \& \rangle$ (bit-exclusive-OR and bit-AND) instead of $G = \langle +, \times \rangle$ produces the H-tree circuit given in Fig. 8. The choice of the algebra will be explained later.

The circuit given in Fig. 8 can be efficiently laid out using an H-tree structure [21]. We will now consider the case where the inputs are partitioned into two partitions, one of which has a bounded size. In this case, a linear structure can be created by recursively decomposing the leaf circuit associated with the unbounded partition. For example, consider the carry function defined by

$$c_i = (c_{i-1} \& (x_i | y_i)) | (x_i \& y_i), \quad i = 1, 2, 3, 4.$$

Recursively decomposing the unbounded partition with respect to $G = \langle |, \& \rangle$ produces the linear circuit given in Fig. 9.

Once the designer has decided whether to use balanced or unbalanced partitions, the best partition is found by enumerating all possible partitions and computing the rank for each one. Surprisingly, enumeration can be used for functions even of modest size (16 inputs and 8 outputs). The reason for this is twofold. First, it is usually not necessary to compute the exact rank for a given partition. Once it is known that a given partition has a rank larger than the best seen so far, it is not necessary to further

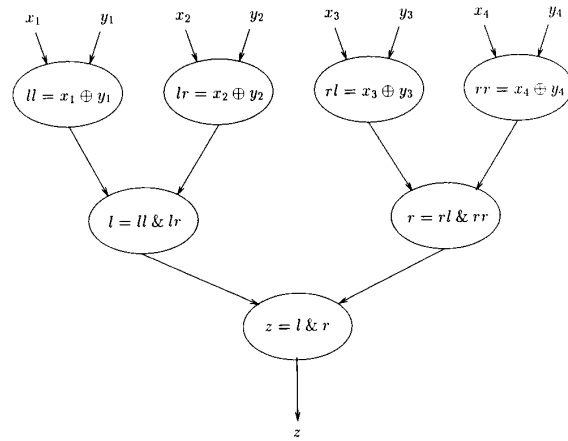


Fig. 8. Balanced decomposition.

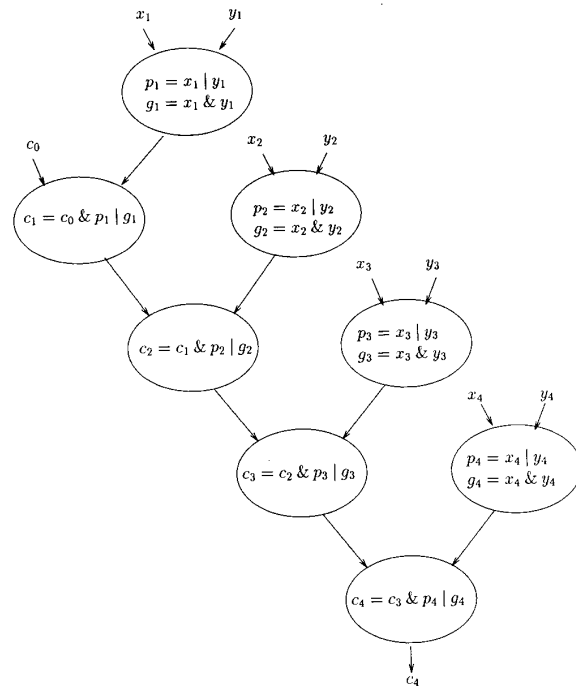


Fig. 9. Unbalanced decomposition.

quantize the rank for that partition. Second, for many functions of practical interest, enumeration of the partitions quickly finds a partitioning of low rank. Hence, determination of whether a given partition has rank less than the best partition seen so far can be relatively efficiently computed (as opposed to computing the exact rank for a given partition). With these restrictions, the number of partitions that must be enumerated, while large, is manageable.

Once the partitioning has been determined, the last problem that must be addressed is that of finding the functions $\tilde{f}$, f_l , and f_r for a given function f and partitioning π .

As shown earlier in this paper, this problem is equivalent to factoring M into LDR . If $G = \langle +, \times \rangle$ (integer addition and multiplication), this problem is equivalent to performing Gaussian elimination on M . However, the use of integer addition and multiplication implies that the overall circuit would be constructed from integer multipliers and adders. Considering the size and speed of integer adders, and especially multipliers, this choice does not lead to good designs except in some very special cases.

The algebra $G = \langle \wedge, \& \rangle$ (bit-exclusive-OR and bit-AND) is equivalent to integer multiplication and addition modulo 2. Like $G = \langle +, \times \rangle$, it also forms a ring. Thus for implementation, $G = \langle \wedge, \& \rangle$ is used as it was in the comparison circuit shown in Fig. 8. Hence L , D , and R can be found using a procedure similar to performing Gaussian elimination on M . The use of bit-exclusive-OR and bit-AND implies that the overall circuit can be constructed from exclusive-OR and AND gates. For example, consider the following function:

$$c_i = (c_{i-1} \& (x_i | y_i)) | (x_i \& y_i), \quad i = 1, 2, 3, 4.$$

A recursive balanced decomposition of this function produces the circuit given in Fig. 10.

It would seem that $G = \langle |, \& \rangle$ (bit-OR and bit-AND) would be an even better choice, since the overall circuit could be constructed from bit-OR and bit-AND gates. This was the algebra used in the carry circuit shown in Fig. 9. Unfortunately, this algebra does not form a ring because 1 has no additive inverse using OR as the addition operation. Hence, a procedure similar to performing Gaussian elimination on M cannot be used. Instead it is necessary to use a minimum-covering procedure. *SET BASIS* is a problem known to be *NP*-complete [10]. It is easy to see that *SET BASIS* reduces to this problem. Thus a factoring-based procedure with respect to $G = \langle |, \& \rangle$ is *NP*-complete as well.

A highly simplified overview of the preceding factoring procedure is presented in the following.

```

procedure factor( $M$ )
  for each balanced partition ( $X_l, X_r$ ) do
    compute rank of  $M$  with respect to ( $X_l, X_r$ );
    if smallest rank so far then remember ( $X_l, X_r$ );
  do end;
  compute  $LDR$  with respect to best partition;
  factor( $LD$ ); factor( $DR$ );
end procedure.
```

IV. FACTOR

At Penn State, we have a prototype multilevel logic synthesis program called *factor* that was designed along the lines presented in the previous sections. The program uses bit-exclusive-OR's and bit-AND's. The users first specify whether they want to use a balanced or an unbalanced decomposition. Then the best partitioning is found by enumerating all possible partitionings and computing the rank for each one. While this may seem to be a time-intensive process, *factor* can synthesize the gate speci-

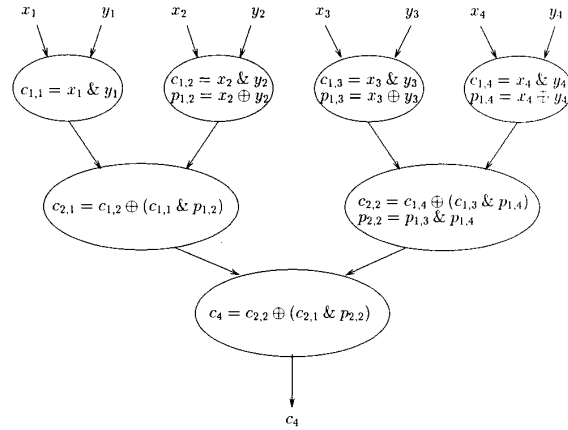


Fig. 10. Balanced decomposition for carry.

cation for a 16 input and 8-output function in less than 45 min of SUN-4 time. The input to and output from *factor* is a circuit description in the language GLUE [12]. For example, when presented with the problem of synthesizing a 4-b adder as described in ripple carry form in Fig. 11(a), using a balanced decomposition *factor* produced the circuit described in Fig. 11(b).

The factored adder (28 gates) has the same gate count as the ripple adder. However, the factored adder (ten gate delays) is about one and half times as fast as the ripple carry adder (14 gate delays). It is interesting to note that the factored adder has a look-ahead structure that does not seem to have appeared in the open literature. A more complete description of this adder can be found in [14]. The factorings given in the examples shown in Figs. 8 and 10 were also produced by *factor*.

V. BENCHMARKING RESULTS

One would expect multilevel logic synthesis tools like *factor*, which are based on approaches that try to minimize communication complexity, to produce fast circuits (when using a balanced decomposition approach) with compact layouts for certain types of functions (e.g., adders and comparators). To confirm this expectation, a series of benchmarking trials were performed on adder-type structures, comparing the performance of *factor* and *misII* [5]. One could also question how well, on a more general class of functions, the communication complexity approach works. To help answer this question, a number of other circuits were also synthesized with both *factor* and *misII*. Several examples of the 1989 MCNC multilevel logic benchmark set with I/O requirements suitable for *factor* (i.e., fewer than 16 inputs and 8 outputs) were selected. In most cases, the function of these circuits was unknown. The 16 benchmark circuits synthesized are listed in Table I. All but the last two circuits are from the MCNC benchmark set. The *blif* file descriptions for the circuits were used unmodified. The version of *misII* and the script used to run it were obtained from *Oct-tools 3.4*. The script file is included in Appendix A.

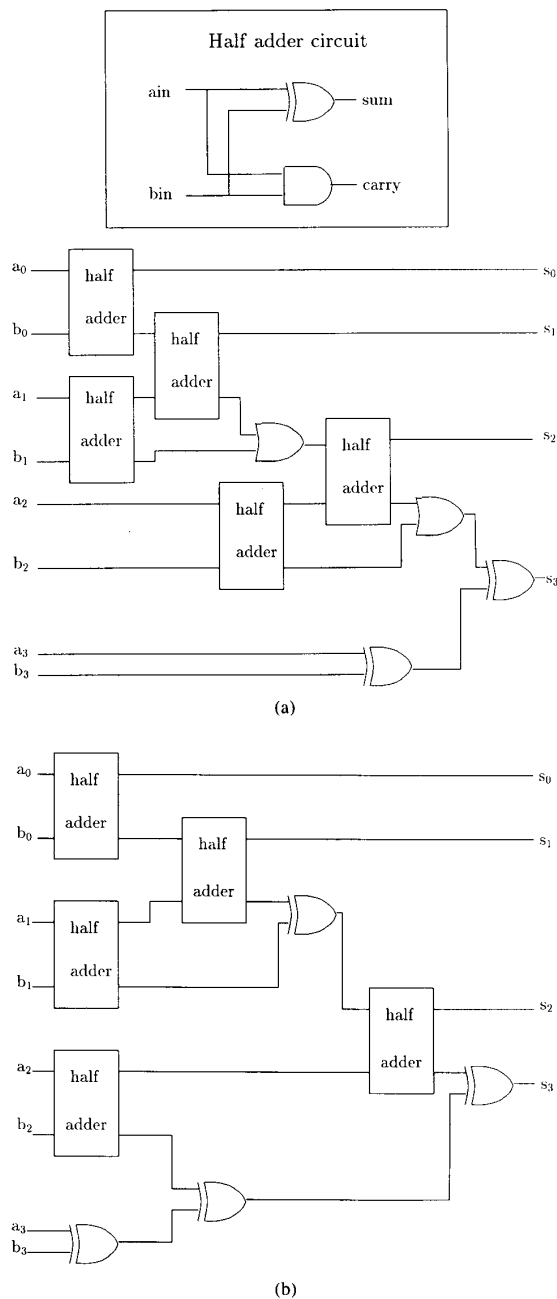


Fig. 11. (a) Ripple carry adder. (b) Balanced factored adder.

Like others, we view the problem of logic synthesis as a two-step process. The first process is one of factoring, or decomposition, and the second is one of technology mapping. For example, *misII* has distinct functions which address each of these two processes. *Factor*, on the other hand, is only a decomposition tool, not a technology mapper. Therefore, an attempt was originally made to compare *factor* to *misII* without technology mapping. However, this biases the results in favor of *factor* to a degree

TABLE I
BENCHMARK SET

Circuit	No. of inputs	No. of outputs
9symml	9	1
C17	5	2
b1	3	4
cm138a	6	8
cm151a	12	2
cm152a	11	1
cm162a	14	5
cm163a	16	5
cm82a	5	3
cm85a	11	3
cmb	16	4
majority	5	1
parity	16	1
z4ml	7	4
adder8	16	8
comp8	16	3

TABLE II
GATE COUNT AND DELAY COMPARISONS (BALANCED DECOMPOSITIONS)

Circuit	No. of Gates			Delay		
	<i>misII</i>	<i>factor</i>	ratio	<i>misII</i>	<i>factor</i>	ratio
9symml	142	93	1.53	30.2	30.9	0.98
C17	6	15	0.40	6.9	10.4	0.66
b1	7	8	0.88	7.8	6.2	1.26
cm138a	23	26	0.88	8.7	7.4	1.18
cm151a	19	101	0.19	11.0	27.3	0.40
cm152a	17	30	0.57	9.4	18.1	0.52
cm162a	34	89	0.38	17.7	23.6	0.75
cm163a	31	49	0.63	14.9	15.7	0.95
cm82a	15	22	0.68	14.6	16.6	0.88
cm85a	43	72	0.60	18.4	19.8	0.93
cmb	47	31	1.52	16.3	7.1	2.30
majority	10	16	0.63	7.4	14.4	0.51
parity	36	30	1.20	18.7	14.7	1.27
z4ml	41	42	0.98	24.0	25.0	0.96
adder8	100	63	1.59	38.3	21.2	1.81
comp8	109	100	1.09	31.2	18.2	1.66

which may not be realistic. Thus the results presented here compare *misII* with technology mapping to *factor* with a simplistic technology mapping. This simplistic technology mapping primarily consists of the migration and subsequent elimination of inverters. This approach, however, biases the results presented here in favor of *misII*.

We ran a number of experiments with various technology mappings for *misII*. The ones presented in Table II used the gate library oi2, ai2, i1, ao121, and oai21. The gates in the library were purposely chosen to be small. The reason for this is twofold. First, we wanted to concentrate on testing our decomposition approach, not technology mapping. Second, in order to reduce "quantitative noise" due to technology mapping, it is important to have a relatively large gate count for even the best circuit implementations. Because of the relatively small size benchmarks, we are forced to include only gates of small size in the technology library. The technology file used is included in Appendix B. The entries in Table II give the

gate count for each benchmark for both *factor* and *misII*. The delay times for each circuit as determined by the delay estimator in the technology mapper of *misII* are also listed. For ease of comparison, the ratios of *misII* to *factor* gate count and delay times are given. A balance decomposition was used for all of the *factor* runs.

As expected, *factor* outperformed *misII* in tree-based circuits (parity, adder 8, comp 8). This was true even though *factor* generated circuits were parallel in nature while the *misII* generated circuits were serial in nature. *Factor* was also competitive in several of the other benchmarks (9symml, bl, cm138a, cmb, z4ml). Even in some of the benchmarks where *factor*'s circuits were significantly larger, they were not significantly slower than the *misII* generated circuits (cm163a, cm82a, cm85a).

To see what penalty *factor* was paying by being forced to use a balanced decomposition, we also ran a subset of the benchmarks where unbalanced decompositions were allowed. We chose primarily those examples where *misII* outperformed *factor*. These results are given in Table III. For ease of comparison, the *misII* results are repeated from Table II.

As compared to balanced decompositions, the largest improvements in gate count for *factor* were made in the z4ml, cm138a, 9symml, and majority benchmarks. As expected, these gains were usually accompanied by a decrease in speed, since a serial circuit is synthesized by *factor* with unbalanced decompositions. Counting both balanced and unbalanced synthesis, *factor* outperformed *misII* on 7 of the 16 benchmarks. Recall that these results are biased toward *misII* because of its greatly superior technology mapping.

However, the true advantage of a multilevel logic synthesis approach based on minimizing communication complexity should be apparent not from comparing gate count, but from comparing layout size. After all, the synthesis approach of minimizing communication complexity is motivated by layout considerations. Thus we continued our benchmarking effort to compare layout sizes for several of the circuits minimized by both *misII* and *factor*. Those examples where *misII* and *factor* had relatively close gate count were chosen for comparison. For this layout comparison we used *artist* [17], a two-dimensional gate matrix layout tool. *Artist* is a cluster-based simulated annealing layout tool. The results of running *artist* on the output of both *misII* and *factor* are given in Table IV. The areas given in the table are the product of the number of gate matrix row runs times the number of gate matrix column runs. This can be converted to λ^2 (i.e., half the feature size) simply by multiplying a factor of 80 (based on MOSIS rev. 6 design rules). In both cases, the same clustering program, one based on gate connectivity, was run on the synthesized logic prior to generating a layout with *artist*.

The improvement in layout size as compared to gate count for *factor* is the most dramatic for balanced and unbalanced 9symml, z4ml, and balanced comp8. For these circuits, more than a 50% improvement was made in lay-

TABLE III
GATE COUNT AND DELAY COMPARISONS (UNBALANCED DECOMPOSITIONS)

Circuit	No. of Gates			Delay		
	<i>misII</i>	<i>factor</i>	ratio	<i>misII</i>	<i>factor</i>	ratio
9symml	142	82	1.73	30.2	35.5	0.85
C17	6	11	0.55	6.9	9.2	0.75
bl	7	9	0.78	7.8	10.6	0.74
cm138a	23	19	1.21	8.7	9.2	0.95
cm151a	19	55	0.35	11.0	31.6	0.35
cm152a	17	33	0.52	9.4	23.3	0.40
cm162a	34			17.7		
cm163a	31			14.9		
cm82a	15	20	0.75	14.6	19.1	0.76
cm85a	43	56	0.77	18.4	24.2	0.76
cmb	47			16.3		
majority	10	12	0.83	7.4	13.9	0.53
parity	36			18.7		
z4ml	41	26	1.58	24.0	29.4	0.82
adder8	100			38.3		
comp8	109			31.2		

TABLE IV
AREA COMPARISONS

Circuit	<i>misII</i>	<i>factor</i>		<i>factor</i>	
		balanced	ratio	unbalanced	ratio
9symml	14181	5626	2.52	3870	3.66
cm138a	660	588	1.12	448	1.47
cmb	1551	924	1.68		
parity	1053	912	1.15		
z4ml	2142	1326	1.62	690	3.10
adder8	3696	2378	1.55		
comp8	7326	4189	1.75		

out size over *misII* as compared to the gate count improvements over *misII*. As one can see from Table IV, even in the case where *factor* generated a larger circuit in terms of gate count (balanced cm138a), the resulting layout was smaller than the *misII* layout. Since layout sizes bear a nonlinear relationship to gate count, it may be important to concentrate on the circuits (e.g., balanced z4ml) with about the same gate count to analyze the real effect of our approach on layout size. The layout size improvement made by using *factor* instead of *misII* for this circuit is significant. The two layouts for the balanced z4ml circuit are shown in Fig. 12.

Unfortunately, even the simple technology mapping done for *factor* destroys the hierarchical circuit description produced by *factor* (see, for example, Fig. 11(b)). Thus we were unable to use clustering derived from this communication-based hierarchical description in our layout trials. We have observed [17] that this type of clustering can give much better layout results than connectivity-based clustering. Thus the true advantage of *factor*'s approach to synthesis for layout is not completely illustrated by these layout comparisons. To do this, a technology mapper which retains the hierarchical circuit description would have to be used. Such a technology mapper does not currently exist.

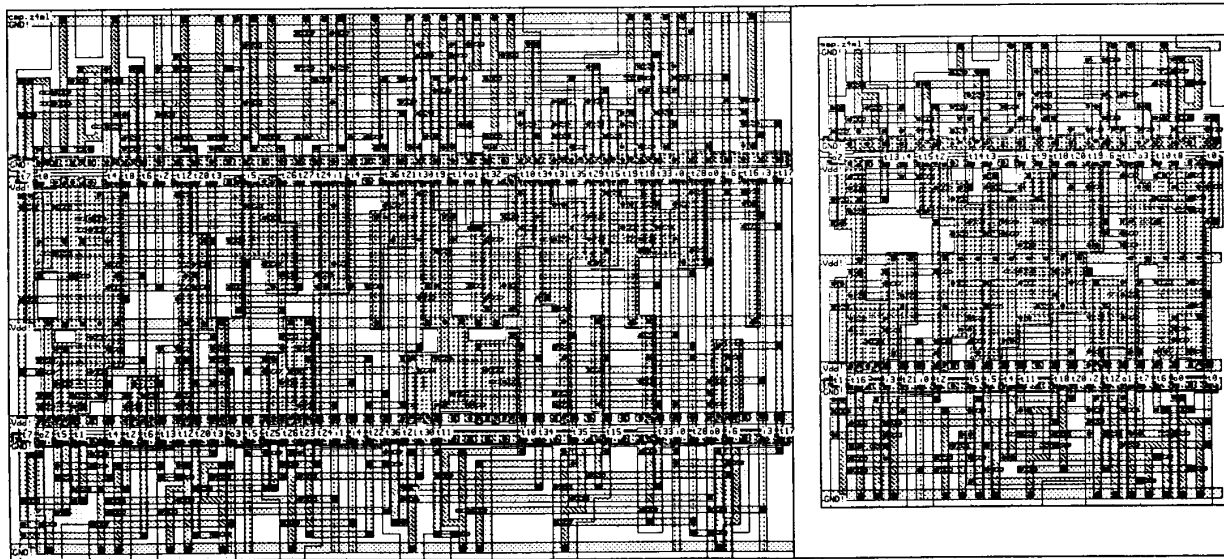
*misII**factor*

Fig. 12. Layouts for balanced z4ml benchmark.

VI. FUTURE EXTENSIONS

In this paper, we have chiefly been concerned with the case where the partitions do not overlap. From Tables II and III, we can see that nonoverlapped partitions are not an appropriate technique for some circuits. If overlapped partitions are allowed, *factor* will be more powerful. Thus an interesting question is how to deal efficiently with overlapped partitions [23].

Our current logic synthesis tool, *factor*, can handle up to 16 inputs and outputs. The principal problem encountered in dealing with larger functions is deciding how to represent them internally. Bit maps can be used. However, a function with 24 inputs and 8 outputs requires $2^{24} = 16$ Mb of storage! Symbolic representation is one way to handle large problems, even though symbolic representation is more difficult to handle [24].

In this paper, we restricted ourselves to discussing completely specified functions. It is also possible to modify the algorithms so as to make use of "don't cares."

VII. CONCLUSION

Multilevel logic synthesis is a difficult problem. We think it is unrealistic to expect a single technique to efficiently solve all logic synthesis problems. Instead, a designer will have to rely on a number of different techniques and will have to know when it is best to apply each technique. By necessity, each technique will have to be fundamentally different from the others. Techniques that are modifications of the same theme will all succeed or fail on the same types of problems.

We have presented in this paper a new multilevel logic-

synthesis technique based on minimizing communication complexity. It performs well for functions that are hierarchically decomposable (e.g., adders, parity generators, comparators, etc.). These are the same class of functions for which other techniques do not perform well. Unlike many other logic synthesis techniques, a lower bound can be efficiently computed to determine how well synthesis was performed.

APPENDIX A

Script File:

```
sweep; eliminate -1
simplify
eliminate -1

sweep; eliminate 5
simplify

resub -a

gkx -abt 30
resub -a; sweep
gkx -bt 30
resub -a; sweep

gkx -abt 10
resub -a; sweep
gkx -bt 10
resub -a; sweep

gkx -ab
resub -a; sweep
gkx -b
resub -a; sweep
```

eliminate 0
decomp -g*
eliminate -1; sweep.

APPENDIX B

Technology File:

GATE inv1	1	O=!a;	PIN * INV 1 999 0.9
			0.3 0.9 0.3
GATE ai2	1	O=!(a*b);PIN	* INV 1 999 1.0 0.2
			1.0 0.2
GATE oi2	1	O=!(a+b);PIN*	INV 1 999 1.4 0.5
			1.4 0.5
GATE aoi21	1	O=!(a*b+c);	PIN * INV 1 999 2.0
			0.4 2.0 0.4
GATE oai21	1	O=!(a+b)*c);PIN	* INV 1 999 1.6
			0.4 1.6 0.4
GATE zero	0	O=CONST0;	
GATE one	0	O=CONST1;	

REFERENCES

- [1] R. L. Ashenurst, "The decomposition of switching functions," *Ann. Comput. Lab. Harvard Univ.*, pp. 29, 30, 1959.
- [2] J. Beekman, R. M. Owens, and M. J. Irwin, "Mesh arrays and LOGICIAN: A tool for their efficient generation," in *Proc. DAC '87*, July 1987, pp. 357-362.
- [3] D. Bostick, *et al.*, "The Boulder optimal logic design system," in *Proc. ICCAD*, Nov. 1987, pp. 62-69.
- [4] R. Brayton, G. Hachtel, C. McMullen, and A. Sangiovanni-Vincentelli, *Logic Minimization Algorithms for VLSI Synthesis*. Hingham, MA: Kluwer Academic, 1984.
- [5] R. Brayton *et al.*, "MIS: A multiple-level logic optimization system," *IEEE Trans. Computer-Aided Design*, vol. CAD-6, pp. 1062-1081, Nov. 1987.
- [6] H. A. Curtis, "A generalized tree circuit," *J. Ass. Comput. Mach.*, pp. 484-496, Aug. 1961.
- [7] J. Darringer, D. Brand, L. Berman, and L. Trevillyan, "Logic synthesis through local transformations," *IBM J. Res. Develop.*, pp. 272-280, July 1981.
- [8] H. A. Curtis, *A New Approach to the Design of Switching Circuits*. New York: Van Nostrand, 1962.
- [9] P. Duris, Z. Galil, and G. Schnitger, "Lower bounds on communication complexity," *Inform. Comput.*, vol. 73, no. 1, Apr. 1987.
- [10] M. R. Garey and D. S. Johnson, *Computers and Intractability*. San Francisco, CA: Freeman, 1979.
- [11] D. Gregory *et al.*, "SOCRATES: A system for automatically synthesizing and optimizing combinational logic," in *Proc. DAC '86*, June 1986, pp. 79-85.
- [12] M. J. Irwin and R. M. Owens, "An overview of the Penn State design system," in *Proc. DAC '87*, July 1987, pp. 516-522.
- [13] R. M. Karp, "Functional decomposition and switching circuit design," *J. Soc. Indust. Appl. Math.*, vol. 11, no. 2, June 1963.
- [14] T. Kelliher, M. J. Irwin, and R. O. Owens, "A fast addition algorithm discovered by a program," to be published in *IEEE Trans. Comput.*
- [15] M. Lightner and W. Wolf, "Experiments in logic optimization," in *Proc. ICCAD '88*, 1988, pp. 286-289.
- [16] K. Mehlhorn and E. Schmidt, "Las Vegas is better than determinism in VLSI and distributed computing," in *Proc. 14th ACM Symp. on Theory of Computing*, 1982, pp. 330-337.
- [17] R. M. Owens and M. J. Irwin, "A comparison of four two-dimensional gate matrix layout tools," in *Proc. DAC '89*, June 1989, pp. 698-701.
- [18] R. H. Risch, "Staggered input networks: An approach to automatic logic decomposition," in *Proc. Int. Symp. on Circuits and Systems*, 1982, pp. 55-57.
- [19] C. D. Thompson, "Area time complexity for VLSI," in *Proc. 11th ACM Symp. on Theory of Computing*, 1979, pp. 81-88.
- [20] T. Sasao, "Application of multiple-valued logic to a serial decomposition of PLA's," in *Proc. 19th Int. Symp. on Multiple-Valued Logic*, 1989, pp. 264-271.
- [21] J. Ullman, *Computational Aspects of VLSI*. New York: Computer Science, 1984.
- [22] A. Yao, "The Entropic Limitations on VLSI Computations," in *Proc. 13th ACM Symp. on Theory of Computing*, 1981, pp. 308-311.
- [23] T. Hwang, R. M. Owens, and M. J. Irwin, "The detection of global variables for multilevel logic synthesis using communication complexity," to be published.
- [24] T. Hwang, R. M. Owens, and M. J. Irwin, "Efficiently computing communication complexity for multilevel logic synthesis," to be published.

*



Ting-Ting Hwang received the M.S. degree in computer science from Pennsylvania State University in 1986, where she is currently working toward the Ph.D. degree at the Department of Computer Science.

Her research interests include multilevel logic synthesis and optimization, logic verification, and VLSI architectures.

*



Robert Michael Owens received the M.S. degree in computer science from Virginia Polytechnic Institute and State University and the Ph.D. degree in computer science from Pennsylvania State University in 1977 and 1980, respectively.

He has worked for IBM and the Naval Surface Weapons Center. He is the author of UREP, a computer communication package. Currently, he is an Associate Professor of Computer Science at Pennsylvania State University. His research interests include computer architecture, parallel algorithms, VLSI architectures, and the CAD tools associated with their implementations.

*



Mary Jane Irwin (S'74-M'77) received the M.S. and Ph.D. degrees in computer science from the University of Illinois, Urbana-Champaign in 1975 and 1977, respectively.

She is currently a Professor of Computer Science with Pennsylvania State University. Her research interests include computer architecture, VLSI design of signal processors, and computer arithmetic.

Dr. Irwin was the Program Chairman for the Eighth Symposium on Computer Architecture. She is currently a member of the editorial board for *The Journal of VLSI Signal Processing* and the *Journal of Parallel and Distributed Computing*. She is also a member of the executive committee of the IEEE Technical Committee on VLSI. She is currently the Secretary/Treasurer of ACM's SIGARCH.