Logic Synthesis for Field-Programmable Gate Arrays

Ting-Ting Hwang, Robert Michael Owens, Mary Jane Irwin, *Fellow, IEEE,* and Kuo Hua Wang

***Abstract*— In this paper, we consider the problem of configuring Field Programmable Gate Arrays (FPGA's) so that some given function is computed by the device. Obtaining the information necessary to configure a FPGA entails both logic synthesis and logic embedding. Due to the very constrained nature of the embedding process, this problem differs from traditional multilevel logic synthesis in that the structure (or lack thereof) of the synthesized logic is much more important. Furthermore, a metric-like literal count is much less important. We present a communication complexity-based decomposition technique that appears to be more suitable for FPGA synthesis than other multilevel logic synthesis methods. The key is that our logic optimization technique based on reducing communication complexity is good enough to allow a simple technology mapping to work well for FPGA devices.**

I. Introduction

FIELD PROGRAMMABLE Gate Arrays (FPGA's) are a recent technology that provides users programmability in the field. Because of their very short turnaround time and low manufacturing cost, there has been an increasing interest in system prototyping using FPGA's. One important class is the RAM-based FPGA (also called "look-up table FPGA's"). A novel feature of these devices is that each basic block can implement any logic function that satisfies the I/O constraints of the basic block. The interconnections between the basic blocks consist of metal segments joined by program controlled pass transistors. From the above description, we can see that two features of FPGA's that the logic synthesis process has to take into consideration are 1) function units implemented using look-up tables (LUT) and 2) restricted interconnection. To configure a FPGA requires programming both the LUT and the interconnect. In many ways, configuring a FPGA is similar to the traditional custom VLSI synthesis process. Given a logic function to be realized, an equivalent multilevel logic description is first generated, the synthesized description is technology mapped, and finally, the mapped description is embedded in circuitry.

However, even though similar in approach, traditional logic synthesis tools may not be well suited for FPGA configuration for the following two reasons: First, because each LUT can emulate any k input gate, where k is the input limit of a LUT function unit, it is possible for large expressions and/or subexpressions of the synthesized description to be mapped to a single LUT if the expression/subexpression involves k or fewer different variables. For example, consider the case where basic block is a 16-bit, 4-input LUT and let $f_1 = abcde$ and $f_2 = abcd + \bar{b}d + b\bar{c} + \bar{c}d$. Although in terms of literal count f_2 is more complex than f_1, f_2 can be realized using one 4-input LUT while f_1 would need two 4-input LUT's. This implies that traditional logic synthesis that focuses on minimizing the number of literals may not be the best approach for FPGA's. The processes of logic synthesis and technology mapping for FPGA's should not be decoupled. Second, because of the FPGA's structure, embedding the mapped description enjoys less flexibility than in traditional custom VLSI synthesis. This implies that how well a mapped description can be embedded in a FPGA can be very dependent on the structure of the mapped description. To embed a general circuit graph is more difficult than embedding a restricted circuit graph (e.g., a tree). All of this means that how good the final configuration is will be much more dependent on the structure of the synthesized description than in traditional VLSI synthesis.

We present a logic synthesis technique that overcomes these two limitations. Our technique is based on minimizing the communication complexity of the synthesized circuit rather than concentrating on literal count. The remainder of this paper is organized as follows. In Section II, we review the communication-based logic synthesis technique that is used to construct a global, technology-independent structure. In Section III, we show that communication-based logic synthesis is more suitable for configuring FPGA's. We then present an algorithm for mapping technology-independent logic to FPGA's in Section IV. Some benchmarking results and comparisons are given in Section V.

II. Communication Complexity-Based Logic Synthesis

In this section, we will give an overview of communication complexity-based logic synthesis. A detailed description can be found in [11]. Succinctly, communication complexity-based logic synthesis can be stated as follows: For some function

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

where the elements of x are the n binary inputs and the elements of y are the m binary outputs of the function and some partitioning (x_l, x_r) of x, let f_t, f_l and f_r be a set of

Manuscript received May 31, 1992; revised April 26, 1994. This paper was recommended by Associate Editor R. Brayton.

T.-T. Hwang is with the Department of Computer Science, National Tsing Hua University, Taiwan 30043, R.O.C.

R. M. Owens and M. J. Irwin are with the Department of Computer Science, The Pennsylvania State University, University Park, PA 16802 USA.

K. H. Wang is with the Department of Computer Science and Information Engineering, Chiao Tung University, Taiwan 30043, R.O.C.

IEEE Log Number 9402698.

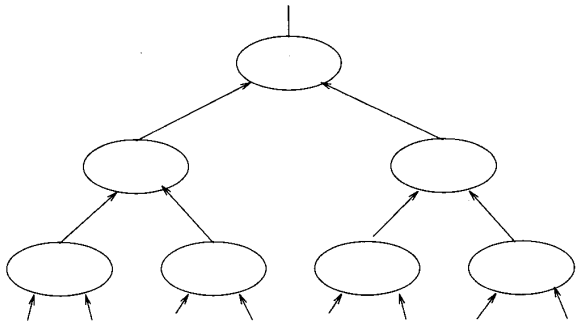


Fig. 1. Balanced decomposition.

functions such that

$$f(x) = f_t(f_l(x_l), f_r(x_r))$$
$$f_l : Z^{|x_l|} \rightarrow Z^{m_l} \qquad f_r : Z^{|x_r|} \rightarrow Z^{m_r}$$
$$f_t : Z^{m_l} \times Z^{m_r} \rightarrow Z^m$$

and m_l and m_r are minimized. The communication complexity of the partitioning (x_l, x_r) is given by m_l and m_r. Thus, the problem of synthesizing a circuit for f has been decomposed into the problem of synthesizing subcircuits for the functions f_t, f_l, and f_r. Communication complexity-based logic synthesis involves recursively finding a partitioning (x_l, x_r) that satisfies certain constraints and has minimum communication complexity and then finding the set of functions f_t, f_l, and f_r.

It is interesting to note that it is possible to find the communication complexity of a given partitioning without actually finding a set of functions f_t, f_l, and f_r that achieve that complexity [17]. Furthermore, for a given partitioning, there are typically many sets of functions f_t, f_l and f_r that achieve the minimum communication complexity. In [10] we presented a method that finds a "good" set of functions f_t, f_l, and f_r for a large set of functions, f, where f is decomposed as

$$f(x) = \sum_{i=1}^{k} f_{l_i}(x_l) \cdot f_{r_i}(x_r)$$

with respect to the algebra $G = \langle xor, and \rangle$. This method is similar to the multiple disjoint decomposition defined in [2], [4], [15]. The subfunctions are synthesized directly using *xor* and *and* gates; that is, there is no encoding problem. The encoding problem was only partially solved using some complicated procedures in [15]. Multiple disjoint decomposition was also used by Murgai *et al.*, in [18], where the first feasible decomposition was proposed. However, solving the encoding problem was not addressed.

The set of subfunctions obtained by our method may not achieve the absolute minimum communication complexity. Even so, in many cases our method can be used to find a better multilevel representation of a function than other multilevel synthesis methods [11]. In order to handle larger size problems, our technique has been extended to include symbolic computation along with efficient heuristics for input partitioning [13].

Two different partitioning schemes are of particular interest. In one partitioning scheme, the inputs are partitioned into two partitions such that both partitions are of approximately equal size. As illustrated in Fig. 1, if these partitions are nonoverlapping, a circuit whose global structure is tree-like results.

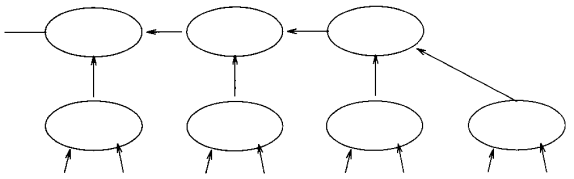


Fig. 2. Unbalanced decomposition.

In the other partitioning scheme, the inputs are partitioned into two partitions such that one of the partitions has a bounded size. As illustrated in Fig. 2, if these partitions are also nonoverlapping, a circuit whose top level structure is a linear structure results. Partitions have been found for both balanced or unbalanced decompositions using a greedy approach and a Kernighan-Lin-type procedure. We have shown in [13] that these partitioning heuristics can obtain satisfactory results. Note that globally, both balanced and unbalanced partitionings have structures of fanout free trees and within each node subfunctions are constructed using *xor-and* operators. A balanced decomposition for the example 9symml is shown in Fig. 9.

We have developed a tool called *factor* that uses this communication complexity approach for multi-level logic synthesis. It can handle functions using either balanced or unbalanced, nonoverlapping partitions. A number of circuits from the MCNC logic synthesis benchmark set have been synthesized using *factor*. Not surprisingly, it competes especially well with other methods on functions that are hierarchically decomposable (e.g. adders, parity generators, comparators, etc). As with any algorithm that runs in polynomial time with respect to the input size, *factor* solves only a "sparse" number of problems. We have shown [11] that the set that *factor* handles well is not handled well by any other alternative method. Furthermore, the set solved efficiently by *factor* is a very important class of circuits. Since we have only considered nonoverlapped partitions for those circuits that have a large set of control variables, *factor* does not perform as well because many interconnections are needed to broadcast control variables. Work is under way to extend the program to handle overlapping partitions [20].

For hierarchically decomposable tree-based circuits, *factor* was able to achieve up to a 30% reduction in gate count compared with *misII*. We also have continued our benchmarking effort to compare layout sizes of the synthesized circuits using in-house gate matrix layout tool [14]. We found that even in the cases where *factor* generated a slightly larger circuit in terms of gate count, *factor's* layout was still smaller than *misII's*. This can be attributed to the synthesis approach used—minimizing communication complexity. Complete details of the benchmarking results are reported in [11] and [13].

III. Logic Configuration for FPGA's Using Communication Complexity

Two side effects of the communication-based logic synthesis approach make it especially suitable for the configuration of

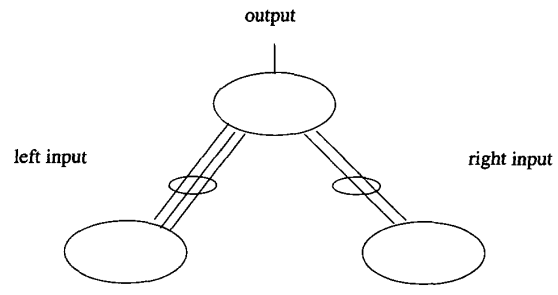


Fig. 3. Communication complexity as a measure of LUT count.

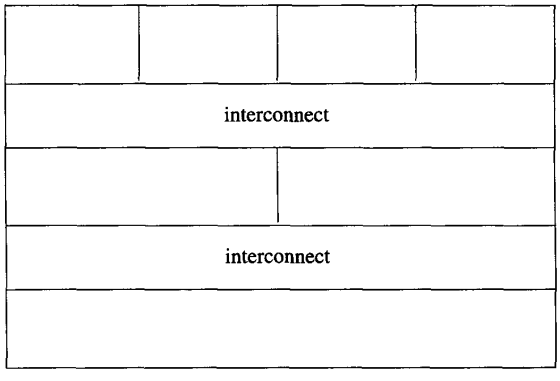


Fig. 4. Tree embedding.

FPGA's: 1) communication complexity is almost a direct measure of LUT count and 2) the resulting tree structure is easy to embed. Recall that the goal of communication-based logic synthesis is to find partitionings that minimize the number of interconnections between subcircuits. In fact, the synthesis process can be set up to minimize the sum of all such interconnections. By minimizing these interconnections, the synthesis process is in turn minimizing the number of subcircuit outputs. In general, each of these outputs can be generated using only a few (if not one) LUT's. The number of LUT's used to implement the subcircuit outputs is thus minimized. Also, note that since the top function is constructed using left and right inputs with the operators *xor-and*, minimizing communication complexity also minimizes the input set of the top function. Hence, a side effect of minimizing communication complexity is that the number of LUT's is also minimized for the top function. Fig. 3 shows a one level decomposition. Each line of *left input* and *right input* is an output of the lower-level subcircuits. Each line is highly independent, and thus each of these lines can be implemented using one or more LUT's. The top functions are functions of left input and right input.

As part of configuring the FPGA, the technology-mapped synthesized circuit must be embedded in the mesh as defined by the FPGA. The top-level tree or linear structures produced by using either balanced or unbalanced decompositions can be embedded in a mesh easier than other more general structures. For example, as illustrated in Fig. 4, a straightforward layering of a tree structure usually works well.

```
highway(/*inputs*/i2, i3, i4, i105, i106, start_timer,
   hl.1, hl.0, fl.1, fl.0, cars, /*outputs*/i5, i6, i7,
   i107, i108, i109, i110, i112, i113, hygreenlight, hyredlight,
   hyyellowlight, farmgreenlight, farmredlight, farmyellowlight)
{
    i5 = !i2*!i3*i4*start_timer + i2*i3*i4*start_timer +
    !i2*!i4*start_timer + !start_timer;
    i6 = !i3*!i4 + i3*i4 + !start_timer;
    i7 = !i4*start_timer;
    i107 = ![307] + [121]*[124];
    i108 = ![90] + i105*i106;
    i109 = ![76] + !i108*i106;
    i110 = !i105*i106;
    i112 = !i106*i105;
    i113 = !i105;
    hygreenlight = !hl.1*!hl.0;
    hyredlight = !hl.1*hl.0;
    hyyellowlight = !hl.0*hl.1;
    farmgreenlight = !fl.1*!fl.0;
    farmredlight = !fl.1*fl.0;
    farmyellowlight = !fl.0*fl.1;
    [124] = i2*i106;
    [121] = !i6*!(i4 + !start_timer);
    [307] = !(!(!i6*i5) + !i7) + !i105;
    [41] = !(i5 + !start_timer)*!(i6 + !(i4 + !start_timer));
    [161] = !cars + !i106;
    [35] = !(!start_timer + !i2)*!(i5 + !cars);
    [90] = !(![41]*![161])*!(!i107*[35]);
    [76] = !([35]*!(!(i106 + !i105) +
    !(i105 + !i106)))*!(!(i105 + !i106)*i107);
}
```

Fig. 5. Baseline highway description.

Since we have observed that in many cases the size of a subcircuit is approximately the same as that of the subcircuits under it taken together, it is not unexpected that the layers would be of approximately the same length. The interconnect advantages of such an embedding are obvious.

In the following, we consider an FPGA configuration for one example. This example (the highway traffic light controller) was initially obtained from the MCNC layout synthesis benchmark set. We will consider here only the combinational part of the circuit. To give us a baseline, we first synthesized a configuration using a straightforward method based on the application of traditional techniques. This method entailed first running the circuit description through misII [3] to produce a good multilevel logic description. We then took the misII-produced decomposition and manually mapped it to the optimal number of 4-input LUT's. The result of this process is given in Fig. 5.

Note that this example requires the use of 23 4-input LUT's as it has 23 4-inputs "gates."

To generate a communication-based configuration, we first used our program *factor* to produce a synthesized description. Then, the mapping algorithm was performed within each subcircuit to produce a technology-mapped description. The mapping algorithm will be described in Section IV. The result of this process is given in Fig. 6.

```
highway(/*inputs*/i2, i3, i4, i105, i106, start_timer, hl.1, hl.0, fl.1, fl.0,
   cars, /*outputs*/i5, i6, i7, i107, i108, i109, i110, i112, i113, hygreenlight,
   hyredlight, hyyellowlight, farmgreenlight, farmredlight, farmyellowlight)
{
    highway_left(i2, i3, i4, start_timer, i5, i6, i7, u3, u6);
    highway_right(i105, i106, cars, v1, v2, i110, i112, v7);
    i107 = !u3*i106 + u3*i105;
    i108 = !u6*v1*!v7 + !v1*v7 + u6*!v1;
    t8 = !u6*!v7*!i110 + !u3*v7*i110 + !u3*u6*i110 + u3*!u6*!v7;
    i109 = !v2*t8 + v2*!t8;
    i113 = !i105;
    hygreenlight = !hl.1*!hl.0;
    hyredlight = !hl.1*hl.0;
    hyyellowlight = !hl.0*hl.1;
    farmgreenlight = !fl.1*!fl.0;
    farmredlight = !fl.1*fl.0;
    farmyellowlight = !fl.0*fl.1;
}
highway_left(i2, i3, i4, start_timer, i5, i6, i7, t4, t5)
{
    i5 = !i2*!i3*i4*start_timer + i2*i3*i4*start_timer +
    !i2*!i4*start_timer + !start_timer;
    i6 = !i3*!i4 + i3*i4 + !start_timer;
    i7 = !i4*start_timer;
    t4 = i3*!i4*start_timer + !start_timer + i4 + !i2;
    t5 = !start_timer + !i4 + !i3 + !i2;
}
highway_right(i105, i106, cars, t0, t2, i110, i112, t10)
{
    t0 = i105*i106*!cars + !i106;
    t2 = i105*i106*!cars;
    i110 = !i105*i106;
    i112 = i105*!i106;
    t10 = i105*!i106 + !i105*i106 + !cars;
}
```

Fig. 6. Factor synthesized highway description.

Except for one output, i109, each of the subcircuit outputs can be generated using only one 4-input LUT. This example requires the use of 21 4-input LUTs. Note that 12 of the 15 outputs (i.e., i5, i6, i7, i110, i112, i113, hygreenlight, hyredlight, hyyellowlight, farmgreenlight, farmredlight, farmyellowlight) are functions of four or fewer inputs and, hence, can be implemented by a single 4-input LUT using *any* synthesis approach. Since each different nontrivial primary output requires at least one 4-input LUT, these outputs can not be "optimized." This leaves only three outputs, i107, i108, and i109, for which optimization is possible. Using the first approach, it takes 11 4-input LUT's to generate these outputs while in the communication complexity-based approach it takes nine, a 10% improvement.

The *factor* synthesized highway description can be embedded in a FPGA as illustrated in Fig. 7. The embedding style used is that depicted in Fig. 4.

IV. Technology Mapping

Several specific mappers for RAM-based FPGA's have been developed [1], [5], [6], [7], [8], [16], [18], [19]. It seems that most of them, if not all, work on a decomposed network produced by an algebraic decomposition tool (e.g., misII). In this section, we describe a greedy technology mapper [12] for FPGA's based on the network synthesized using a communication-driven optimization tool. This type of logic optimization allows a simple greedy approach to work well.

The network obtained from *factor* is an *xor-and* representation. Globally, it is a tree structure as shown in Fig. 1. Let a *region* corresponds to a node in the global tree structure. Then, f_t in a region is computed by first anding the left input with right input (left/right input may be the constant 1) and then xoring the product terms of the anding. We define the following sets for ease of description. Let

$F(G) = \{f_1, f_2, \ldots f_m\}$ represents f_t in region G,
$L(G) = \{l_1, l_2, \ldots l_x\}$ represents left inputs, f_l, in region G, and
$R(G) = \{r_1, r_2, \ldots r_y\}$ represents right inputs, f_r, in region G.

Note that f_t, f_l, and f_r are multiple output functions and that the elements in $F(G), L(G)$, and $R(G)$ are single output functions. We also define

$$T_i(f_i) = \{t_i | t_i \text{ are product terms, } l_j \cdot r_k, \text{ of } f_i, \text{ where } l_j \text{ or } r_k \text{ may be constant } 1\}.$$

Our mapping algorithm is conducted in two phases. In the first phase, a region marking is performed on each region of the global tree structure bottom up. Each marked node uses at most k inputs variables, where k is the input constraint of a logic block. In the second phase, we reduce the number of blocks by merging subfunctions. Each phase is described in detail in the following two subsections.

A. Region Marking

A region is the basic unit for marking. Within a region, we keep track of the input set for each region output node f_i. A node f_i is called an overflow node if its global input support is larger than k, the input constraint of a logic block.

A region, G, is processed as follows. First, we check if the global support of each region output node $f_i \in F(G)$ is more than k. If none of them exceeds k, we proceed to the next higher level region and the inputs to f_i are recorded. If some region output nodes have global support more than k, these nodes are required to be decomposed in order to be implemented in a logic block. Let the set of overflow nodes be denoted as F_o. The immediate fanins, i.e., l_i's and r_i's from $L(G)$ and $R(G)$, to the overflow nodes are the candidates to be decomposed as subfunctions. The reasons for selecting l_i and r_i for marking are as follows. Because the bottom-up mapping is used, all the l_is and r_is are feasible subfunctions to be implemented using a single basic block. However, a more important reason is that the l_i's and the r_i's are the f_t at the lower level and thus are more likely to be used more than once. The selection of the l_is and r_is is based on the following gain function

$$\text{gain } (l_i \text{ or } r_i) = \alpha * (\text{number of fanouts}) + (1 - \alpha) * (\text{number of inputs})$$

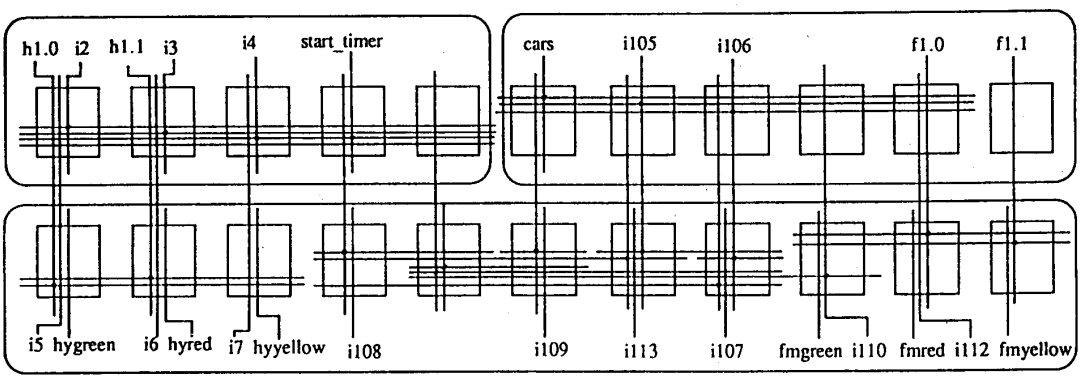


Fig. 7. FPGA highway description.

where α is a real value and $0 \leq \alpha \leq 1$. α can be set with different value for different mapping preferences. If the fanout utilization is the only concern, α is set to 1. If we want to minimize the input used in a basic block, α is set to 0.

When an l_k or r_k is selected, all inputs to it are masked. After each selection, all overflow nodes are checked. If the input constraint is satisfied, we delete the overflow node from F_o. This marking process continues until F_o is empty or all elements in $L(G)$ and $R(G)$ are marked.

If all elements in $L(G)$ and $R(G)$ are marked and F_o is not empty, it means that all l_i's and r_i's are created as basic blocks and there is still some f_i with more than k l_i's and r_i's. We need to group terms of $l_i \cdot r_i$. Remember that a f_i in $F(G)$ is constructed by anding l_i and r_i and then xoring the terms of anding. For each element, f_i, in F_o, we compute $T_i(f_i)$. T_is are processed according to the size of T_i in ascending order. Since f_i is computed by xoring the elements in T_i, elements in T_i can be grouped together in any order as long as the total number of inputs is less than the constraint. Each group corresponds to a basic block. Terms in T_i are grouped according to the utilization of the group. If the group will be used in more f_is, the group has higher priority to be created.

The procedure, *marking*, describing the above algorithm is given in Fig. 8.

An example illustrating the mapping process is shown in Fig. 9 for the case $k = 5$. The circuit, 9symml, from the MCNC benchmark set is synthesized. Fig. 9 shows the global network decomposed by *factor*.

There are eight regions, A, B, C, D, E, F, G, and H, in the network shown in Fig. 9. Our mapper processes each region bottom up. Since there are no overflow nodes in the regions A, B, C, D, E, F, and G, no action is taken for these regions. However, in region H node f contains nine inputs by recursive substitutions of unmarked inputs. Thus, f is an overflow node. Note that since the input sets of unmarked subfunctions are recorded when the procedure moves from the bottom up, no computation time is spent on such substitutions. Through the procedure, the left inputs, $l_{1,0}, l_{1,1}, l_{1,2}, l_{1,3}$ and right inputs, $r_{1,0}, r_{1,1}, r_{1,2}, r_{1,3}$ in D are checked and all are selected to be marked. With all left and right inputs marked, f is still an overflow node. Thus, term grouping is performed. $T(f)$ is computed as $\{l_{1,0} \cdot r_{1,0}, l_{1,1} \cdot r_{1,1}, r_{1,2}, l_{1,2} \cdot \bar{r_{1,0}}, l_{1,3} \cdot r_{1,3}\}$. We

```
procedure marking (F_o)
D = the immediate dependent set of F_o;
sort the nodes in D into list S according to their gains;
while (F_o ≠ ∅ and S ≠ ∅) {
        N_d = the first node in S;
        mark N_d, and all inputs to N_d are masked;
        F_u = the nodes in F_o with required inputs ≤k;
        F_o = F_o - F_u;
        S = S - {N_d };
}
if (F_o ≠ ∅ ){
        for each node f_i in F_o, compute T_i(f_i);
        for each T_i {
                Find a subet T_si with inputs ≤ k and with the highest utilization;
                N_d = a node created to compute T_si;
                Mark N_d, and all inputs to N_d are masked;
                for each T_j of F_o ( T_si ⊂ T_j )
                        T_j = T_j - T_si + {N_d};
        } until the number of inputs to f_i ≤ k;

}
```

Fig. 8. The algorithm of *marking*.

group $l_{1,0} \cdot r_{1,0}, l_{1,1} \cdot r_{1,1}$ and, $r_{1,2}$, which satisfies the input constraint as a new node N_1. Now since f has inputs less than k, 5, it can be implemented as a basic block.

B. Local Merging

After the marking phase, each marked node requires no more than k inputs. Each marked node can be directly implemented by a single basic block. The Xilinx 3000 series FPGA allows two functions to share a single k-input LUT if each function uses no more than $k - 1$ inputs and the two functions together have no more than k inputs. This type of LUT is referred to as a CLB. If we consider the case where two functions can be implemented in one basic block, the total number of blocks can be reduced.

The merging procedure is also performed bottom up on the global tree structure. Each time, a region G is considered. Due to the feature of disjoint partitioning of inputs, at a given region the marked nodes in $L(G)$ and the marked node in $R(G)$ will never have shared inputs. So we only merge the marked nodes in $F(G)$ and $L(G)$. Similarly, we merge the

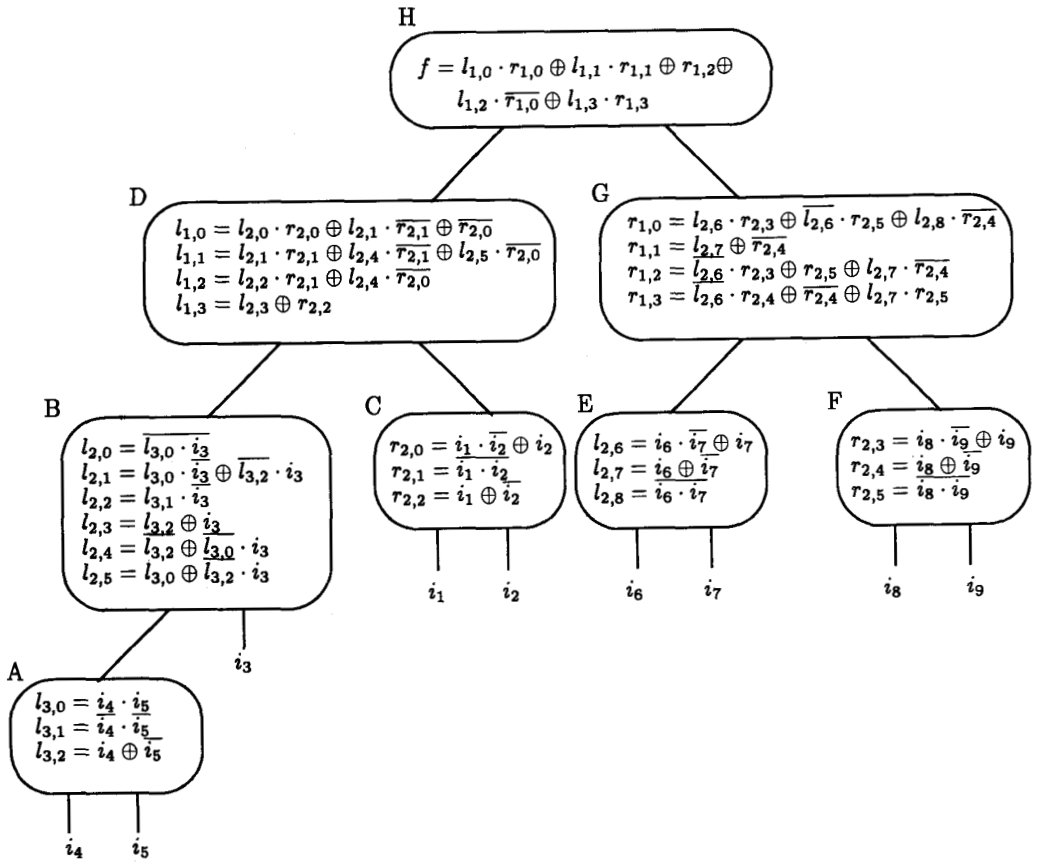


Fig. 9. Circuit 9symml decomposed by *factor*.

marked nodes in $F(G)$ and $R(G)$. The selection of merged nodes is based on a greedy approach. Note that node merging is performed only on adjacent regions. Since only a subset of marked nodes are considered each time, it is very efficient in terms of computation time. For the 9symml example, we consider merging on regions D and H, and G and H.

V. Benchmarking

In order to test the performance of *factor*, some examples from the MCNC benchmark set were synthesized for the FPGA model. We restricted our benchmark set to those that have been evaluated by other FPGA mappers and have an input and output size less than 36 (the size circuit that can currently be synthesized by *factor*). First, structured circuits were synthesized by *factor*. Then, subcircuits generated by *factor* were mapped to basic blocks bottom up using the mapping algorithm described in Section IV. We assumed an input limit size of five and that merging of two functions is possible as in the Xilinx FPGA. Both balanced and unbalanced partitioning were tried and the best result from the two was selected. Table I gives the comparisons of our tool and several other FPGA synthesis tools (Hydra [8], edge [21], c-crf [6], and mis-p [18]) for area optimization. The number in each entry is the number of CLB's needed. A blank entry in the table means that the data is not available. Recent results for mis-pga [19] are not quoted since they did not use merging.

TABLE I
Area Comparisons

Circuits	Hydra	edge	c-crf	mis-p	factor-map
9symml	33	52	41	43	8
rd73	13	26			8
rd84	27		53	32	9
z4ml	4	18	3	7	4
5xpl	21	28	20	23	14
9sym	57	63	42	59	8
alu2	94		83	102	68
count	26		27	28	
misex1	8	14	14	10	11
misex2	20	35			29
duke2	79		89	105	171
vg2	20		18	21	63
sao2	36				31

Factor-map has done especially well for adder-type circuits e.g., 9symml, rd73, rd84, etc., but has not performed as well for those circuits with global signals e.g., duke2, and vg2.

TABLE II
DELAY COMPARISONS

Circuits	Chortle-d		factor-map	
	depth	CLBs	depth	CLBs
9symml	5	51	3	8
rd84	4	54	3	9
z4ml	3	18	2	4
5xpl	3	21	3	14
9sym	5	53	3	8
alu2	9	177	4	68
count	4	78	4	27
misex1	2	13	2	12
duke2	4	180	6	171
vg2	4	43	7	63

Table II gives the delay comparisons of our tool and Chortle-d [7]. The maximum depth and the number of CLB's in the mapped circuits are shown.

Factor-map has outperformed Chortle-d in both level of delay and number of CLB's for adder-type circuits e.g., 9symml, rd73, rd84, etc. But for those circuits with global signals, e.g., duke2, and vg2, factor-map has not performed well in both delay and area.

VI. CONCLUSION

We have shown how communication-based logic synthesis can be used to an advantage when configuring programmable logic devices. Configuration of a FPGA involves the processes of logic synthesis and logic embedding. Since the allowable FPGA logic primitives usually include a very large number of gates, the processes of logic synthesis and technology mapping cannot be completely decoupled as they normally are in traditional logic synthesis systems. Our communication-based logic synthesis tool has the advantage of not completely decoupling these two processes. The key is that the logic optimization that reduces communication complexity is good enough to allow a simple technology mapping to work well. This approach is especially suitable for functions that are hierarchically decomposable. Also, the structure of the circuit synthesized with our approach is more amenable to embedding in a FPGA. A simple example was given to illustrate the two advantages of our approach.

REFERENCES

[1] P. Abouzeid, L. Bouchet, and K. Sakouti *et al.*, "Lexicographical Expression of Boolean Functions for Multilevel Synthesis of High Speed Circuits," in *Proc. Synthesis and Simulation Meeting and Int. Interchange,* Kyoto, Oct. 1990, pp. 31–39.

[2] R. L. Ashenhurst, "The Decomposition of Switching Functions," *Ann. Comput. Lab. Harvard Univ.*, 29, 30, 1959.

[3] R. Brayton and R. Rudell *et al.*, MIS: "A Multiple-Level Logic Optimization System," *IEEE Trans. Computer-Aided Design,* vol. CAD-6, no. 6, pp. 1062–1081, Nov. 1987.

[4] H. Curtis, "A Generalized Tree Circuit," *J. ACM,* pp. 484–496, Aug. 1961.

[5] R. J. Francis, J. Rose, and K. Chung, "Chortle, A Technology Mapping Program for Lookup Table-Based Field Programmable Gate Arrays," *DAC-90,* June 1990, pp. 613–619.

[6] R. J. Francis, J. Rose, and Z. Vranesic, "Chortle-crf: Fast Technology Mapping for Lookup Table-Based FPGA's," *DAC-91,* June 1991, pp. 227–233.

[7] R. J. Francis, J. Rose, and Z. Vranesic, "Technology Mapping of Lookup Table-Based FPGA's for Performance," *ICCAD-91,* Nov. 1991, pp. 568–571.

[8] D. Filo, J. Yang, F. Mailhot, and G. Micheli, "Technology Mapping for a Two-output RAM-based Field Programmable Gate Array," in *Proc. EDAC-91,* 1991, pp. 534–539.

[9] D. Hill and D. Cassiday, "Preliminary Description of Tabula Rasa, an Electrically Reconfigurable Hardware Engine," *ICCD-90,* Sept. 1990, pp. 391–395.

[10] T. Hwang, R. M. Owens, and M. J. Irwin, "Multi-Level Logic Synthesis Using Communication Complexity," in *Proc. DAC'89,* June 1989, pp. 215–220.

[11] T. Hwang, R. M. Owens, and M. J. Irwin, "Exploiting Communication Complexity for Multi-level Logic Synthesis," *IEEE Trans. Computer-Aided Design,* vol. 9, no. 10, Oct. 1990.

[12] T. Hwang, R. M. Owens, and M. J. Irwin, "Logic Synthesis for Programmable Logic Devices," in *Proc. ICCD'90,* Sept. 1990, pp. 364–367.

[13] T. Hwang, R. M. Owens, and M. J. Irwin, "Efficiently Computing Communication Complexity for Multilevel Logic Synthesis," *IEEE Trans. Computer-Aided Design,* vol. 11, no. 5, pp. 545–554, May 1992.

[14] M. J. Irwin and R. M. Owens, "A Comparison of Four Two-dimensional Gate Matrix Layout Tools," in *Proc. DAC'89,* 1989, pp. 698–701.

[15] R. M. Karp, "Functional Decomposition and Switching Circuit Design," *J. Soc. Indust. Appl. Math.,* vol. 11, no. 2, June 1963.

[16] K. Karplus, "Xmap: A Technology Mapper for Table-lookup Field Programmable Gate Arrays," in *Proc. DAC'91,* June 1991, pp. 240–243.

[17] K. Mehlhorn and E. Schmidt, "Las Vegas is Better than Determinism in VLSI and Distributed Computing," in *Proc. 14th ACM Symp. on Theory Comput.,* 1982, pp. 330–337.

[18] R. Murgai and Y. Nishizaki *et al.*, "Logic Synthesis for Programmable Gate Arrays," *DAC-90,* June 199, pp. 620–625.

[19] R. Murgai *et al.*, "Improved Logic Synthesis Algorithms for Table Look Up Architectures," *ICCAD-91,* 1991, pp. 564–567.

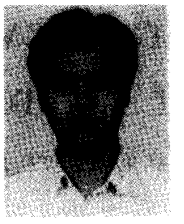
[20] K.-H. Wang, "Overlapped and Multiple Output Decompositions for Communication Complexity Driven Multilevel Logic Synthesis," in preparation.

[21] N.-S. Woo, "A Heuristic Method for FPGA Technology Mapping Based on the Edge Visibility," in *Proc. DAC-91,* June 1991, pp. 248–251.

Ting-Ting Hwang received the M.S. and Ph.D. degrees in computer science from The Pennsylvania State University in 1986 and 1990, respectively. She is currently a faculty member of the Department of Computer Science at the National Tsing Hua University, Taiwan. Her research interests include multi-level logic synthesis and optimization and high-level synthesis.

Robert Michael Owens received the M.S. (1977) degree in computer science from the Virginia Polytechnic Institute and State University and the Ph.D. (1980) degree in computer science from The Pennsylvania State University. He is presently an associate professor of computer science and engineering at Penn State. Before he joined Penn State, he was with IBM and the Naval Surface Weapons Center. Dr. Owens is on the IEEE Signal Processing Technical Committee on VLSI and will be serving as a program cochair of the next ASAP Conference. He is author of UREP, a computer communication package that has been distributed to hundreds of locations world wide. Dr. Owens has authored over 100 scholarly works. His research interests include computer architecture, massively parallel computing, VLSI architectures, and the CAD tools associated with their implementations.

Mary Jane Irwin (S'74–M'77–SM'89–F'94) received the M.S. (1975) and Ph.D. (1977) degrees in computer science from The University of Illinois, Urbana-Champaign. She is a professor of computer science and engineering at The Pennsylvania State University. Dr. Irwin is on the executive committees of the Design Automation Conference and the Supercomputing Conference, is on the editorial boards of *The Journal of VLSI Signal Processing* and the *IEEE Transactions on Computers*, is a member of the Computing Research Board, and is on the IEEE Computer Society Board of Governors. Dr. Irwin is the principal investigator of a Small Scale Institutional Infrastructures grant from NSF. She has authored over 125 scholarly works. Her primary research interests include computer architecture, the design of application specific VLSI processors, high-speed computer arithmetic, and VLSI CAD tools.



Kuo Hua Wang received the B.S. and M.S. degrees, both in computer engineering, from National Chiao Tung University, HsinChu, Taiwan, in 1986 and 1988, respectively. From 1989 to 1991, he was a research assistant in the Information Science Institute, Academia Sinica, Taipei. He is currently working toward the Ph.D. degree in the Department of Computer Science and Information Engineering, National Chiao Tung University. His current research interests include logic synthesis and optimization, logic verification, and high-level synthesis.