Synthesis and Optimization of Multilevel Logic under Timing Constraints

KAREN BARTLETT, WILLIAM COHEN, AART DE GEUS, MEMBER, IEEE, AND GARY HACHTEL, FELLOW, IEEE

Abstract—**The automation of the synthesis and optimization of combinational logic can result in savings in design time, significant improvements of the circuitry, and guarantee functional correctness. Synthesis quality is often measured in terms of the area of the circuit on the chip, which fails to take into account the timing constraints that might be imposed on the logic. This paper describes SOCRATES, a synthesis system capable of generating combinational logic in a given technology under user-defined timing constraints. We believe this system is the first to perform optimized, delay-constrained, multilevel synthesis into standard cell libraries. Applied to a large number of examples, the system has successfully traded off area versus delay and performs optimized, delay-constrained, multilevel synthesis into standard cell libraries.**

I. Introduction

THE AUTOMATION OF the synthesis and optimization of combinational logic can result in savings in design time, significant improvements of the circuitry, and guarantee functional correctness. Synthesis quality is often measured in terms of the area of the circuit on the chip, which fails to take into account the timing constraints that might be imposed on the logic.

This paper describes SOCRATES, a synthesis system capable of generating combinational logic in a given technology under user-defined timing constraints. The user-defined timing constraints are used at two places in our system: during the multilevel function synthesis which yields the structure of the circuit and during the gate level optimization of the circuit which yields the technology-dependent implementation of the logic.

The synthesis of combinational logic can be performed either at the transistor level or using a predefined set of logic gates. PLA's fall in the first category and their automatic generation and area optimization have been studied extensively [4], [11]–[13]. The PLA is a two-level structure with limited potential for timing optimization. Recently, methodologies for multilevel implementation at the transistor level have been proposed based on the use of CMOS domino logic. One of these transistor-level implementations is employed by the "YLE" synthesis system [5] and uses a standard cell-like "pluricell" methodology. The other uses a "PLA-like" layout scheme [14], but both of these transistor-level systems require specialized layout generation programs.

Significant previous work has been done on optimized synthesis into gate array libraries, e.g., MACDAS [16] and LSS [9]. The YLE system [6] performs optimal **multilevel** synthesis using "WEAK DIVISION" into domino pluricells, but is not library based. DeGeus *et al.* [8], [10] have reported on an expert systems approach which uses an optimized sequence of local transformations on a multilevel system produced by WEAK DIVISION. However, in all of these previous investigations, little work has been done to automatically generate optimized, multilevel library-based combinational circuitry while simultaneously meeting timing constraints.

The approach presented in this paper builds on all the referenced approaches. It is an extension and elaboration of the work described in [1] and [2]. The synthesis is performed in two main phases: 1) algorithmic creation of the "logic structure" of the circuit, which is then "mapped" into a given logic gate library, and 2) a set of local transformations translating and optimizing the circuit in the target technology, which underlies the given library. In the first phase, a multilevel circuit is built using a variant of WEAK DIVISION. The timing constraints influence the WEAK DIVISION process through the use of approximate delay models and thus shape the "architecture" of the circuit in terms of its delay–area tradeoff. In the second phase, optimization is performed by a rule-based subsystem, called OPTIMIZE, in which the implementation technology is fully known and an exact delay computation is available. A set of timing-specific rules transforms the circuit trading off area versus delay when necessary to meet the timing constraints imposed by the designer.

We make and utilize throughout this paper the following "cooperation assumption":

> WEAK DIVISION and OPTIMIZE are separate and distinct circuit optimization processes, in which the output of the WEAK DIVISION process is the input to OPTIMIZE. We assume that the quality of the OPTIMIZE output improves with the quality of the WEAK DIVISION output. In most cases, with rare exceptions noted in Section V, this assumption appears to be justified. In an important sense, as discussed in Section V below, if either of these two

Manuscript received January 22, 1986; revised May 23, 1986. This work was supported in part by the National Science Foundation under Grant ECS-8121446.

K. Bartlett and G. Hachtel are with the Department of Electrical and Computer Engineering, University of Colorado at Boulder.

W. Cohen and A. De Geus are with the General Electric Microelectronics Center, Research Triangle Park, NC 27709.

IEEE Log Number 8609878.

modules were perfect, there would be no need for the other.

The sequel begins in Section II with an overview which provides a brief description of each of the system components and how they interrelate. Section III describes WEAK DIVISION and MULTILEVEL MINIMIZATION, the techniques used to obtain an efficient multilevel representation of the function. Section III also describes the cost and delay models used in multilevel synthesis. Section IV describes the module performing local transformations, which is a rule-based expert system. Results and conclusions are covered in Section V and VI.

II. Overview

Fig. 1 shows the architecture of the SOCRATES logic synthesis system. The application-specific input to the system is a functional specification which can take one of three different forms:

1) a set of Boolean equations, possibly multilevel;
2) a set of "linked" PLA's (single output) in ESPRESSO format;
3) a "net list" of library gates.

Each of these forms can be represented graphically by a so-called "Boolean network," as illustrated in Fig. 2. Each node in a Boolean network is a two-level function in one of the three forms listed above. The Boolean network is a directed acyclic graph, whose edges represent the logic dependencies implied by any of the three input forms. Note that nodes without "fan-in" are primary inputs (pentagons in Fig. 2).

As shown at the bottom of Fig. 1, the output of the system is a net list of technology and application-specific library gates. This output is dependent on additional input such as the gate library, the rules library, and the specification of timing and fan-in constraints, which are unrelated to the application. Further, design guidelines, such as the type of delay models to use, can also be considered as system inputs.

The problem of synthesis under timing constraints is viewed as a set of translation and optimization problems, where each optimization is carried out at a different level of abstraction. The first level of abstraction is the sum-of-products level. The process of simplifying the two-level Boolean equations at each node of the given Boolean network is called *minimization* and is carried out by the ESPRESSO IIC logic minimizer [4], [15]. Synthesis at the second level of abstraction involves the creation of an optimum multilevel Boolean network, which is the output of the SYNTHESIS module (dashed box in Figure 1). This module is comprised of the WEAK DIVISION, MULTILEVEL MINIMIZATION, and LIBRARY MAPPING submodules.

The technique of *weak division* [3], [6] is used to decompose the given Boolean network, which may or may not have more than two levels, into an alternative optimized multilevel Boolean network. This "structurally optimized" Boolean network begins to reflect the structure of the final circuit; it is apparent at this stage what logic will be re-used, and approximately how many levels of logic will be used to implement the function. Synthesis of this multilevel Boolean network is heavily influenced by timing considerations. The timing delay models used are discussed in Section III.

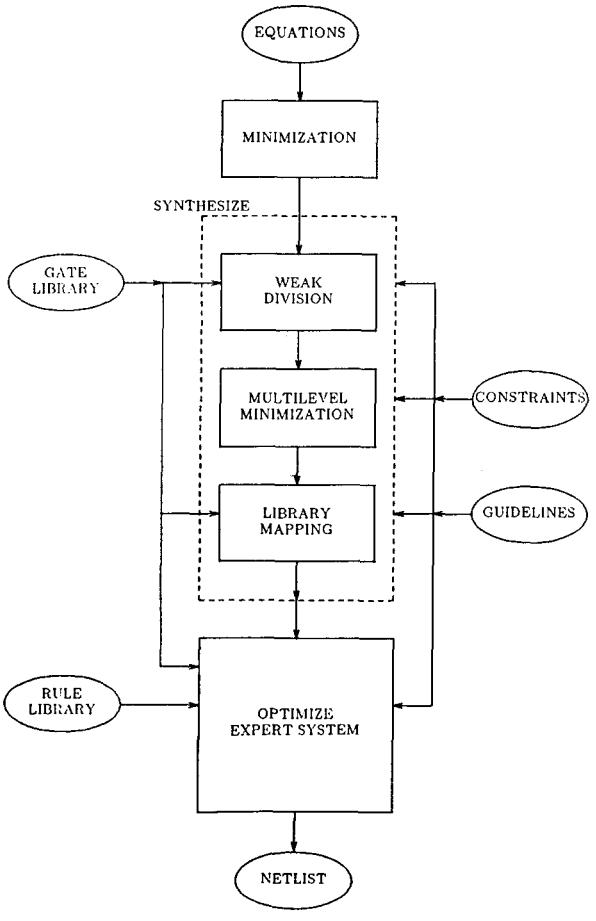


Fig. 1. SOCRATES system overview.

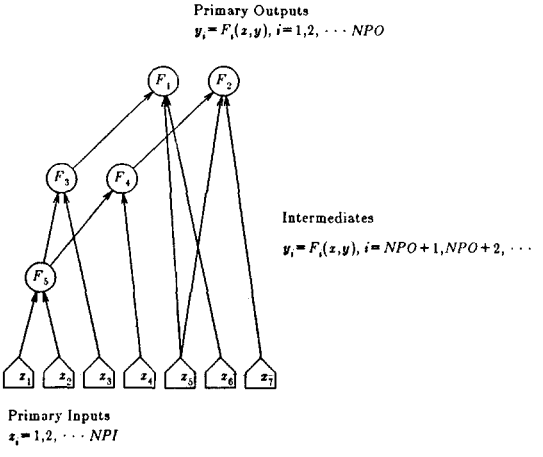


Fig. 2. Multilevel Boolean network.

After a decomposition is found that is satisfactory from a performance and area standpoint, *multilevel minimization* [6] is used to simplify this decomposition. This technique is based on the "don't care" conditions associated

with the intermediate variables introduced by the decomposition into a multilevel function. Transformations suggested by this process are rejected if they do not significantly reduce area or if they increase the estimated delay.

The function(s) is then translated into a logic circuit by replacing the two-level logic functions associated with each node of the Boolean network into library specific logic gates. This mapping into the library may use generic "dummy" gates, like AND's and OR's, which are, ultimately, replaced by the OPTIMIZE module with actual gates from a user-supplied library.

Finally, the circuit implementation is optimized in the target technology. A rule-based system, OPTIMIZE, performs local transformations formulated as rules on the circuit. The optimality of the final circuit thus produced depends on the rules in the library and the order in which these rules are applied. Our approach uses a state space look-ahead algorithm to optimize the application order, as discussed in Section V (logic level synthesis and optimization).

The optimization criterion is based on both circuit area and circuit delay. The system will try to meet the timing constraints and subsequently perform area optimization. The timing constraints will guide the specifics of the implementation leading to a locally optimal choice of gates in the given technology.

If the end product does not satisfy the user, or if the user wishes to consider a number of different possible designs, then the timing constraints can be changed and the procedure iterated.

It is appropriate to think of the SYNTHESIS and OPTIMIZE modules of Fig. 1 as interdependent, interacting, heuristic optimization processes.

III. Multilevel Synthesis and Minimization

In this section, we will describe the WEAK DIVISION and MULTILEVEL MINIMIZATION portions of the SYNTHESIS module. Together, these two algorithms can be regarded as a program for finding an optimum architectural restructuring of the given, initial, often two-level, Boolean network. In particular, we will describe how this program acts to minimize the cost function of Figure 3, which, roughly speaking, serves to minimize the gate array (or standard cell) area subject to user-specified delay guidelines.

The plan of the section will be as follows. First, we describe, in Section III-A, the computation of network "cost" as required by WEAK DIVISION and MULTILEVEL MINIMIZATION. Because the cost constraints of Figure 3 depend on critical path delay, we will also discuss the two basic delay models employed in the SYNTHESIS module:

1) the "unit delay" model (which is technology- and fan-out-independent);
2) the "library element gate delay" model which employs the basic delay equation of Figure 4.

p = Primary Inputs
$b(p)_0$ = Given Boolean Function
η = Initial Boolean Network Realizing b(p)
N^* = Set of all Boolean Networks η Realizing b(p)
$\tau(\eta)$ = Critical Path Delay
g = Logic Gate
Area (η) = $\sum_{g \in \eta} A(g)$

Wires (η) = $\sum_{g \in \eta} |\text{FAN\_IN}(g)|$

$$\text{Objective: } \eta^* = \underset{\eta \in N^*}{\text{ARGMIN}}\{\text{AREA}(\eta) + \text{WIRES}(\eta)\}$$

$$\text{Delay Constraint: } \tau(\eta) \leq \text{CHECK\_DELAY}(\eta^0, \text{guidelines})$$

Fig. 3. Delay constrained area optimization via decomposition.

- Levels of Logic Mode (each gate has "unit" delay)
  - "Architectural" Level Approximation
  - Easy Critical Path Problem
- Library-Element Gate Delay Mode
  - Fanout Dependent
  - $C_{gp'}$ = Capacitance of Input Pin p' of Gate g
  - RLOAD_g = Output Load Resistance of Gate g
  - TSET_g = Intrinsic Delay of Gate g

$$\tau_g = \text{TSET}_g + \text{RLOAD}_g * \sum_{p' \in \text{FANOUT}(g)} C_{gp'}$$

Fig. 4. SYNTHESIS delay models.

Once the cost and constraint computations are defined, we can state the problem to be solved by the SYNTHESIS module as follows:

> Create, in terms of library elements, an optimally structured circuit which, when subsequently processed by the OPTIMIZE module, produces the best final implementable Boolean network.

The basic "cooperation assumption" is that the better the SYNTHESIS module does in solving its optimization problem, the better the OPTIMIZE module will do in solving its problem (a similar optimization with a similar cost function). (The validity of this assumption is tested in Section V below.)

In Section III-B, we describe the WEAK DIVISION optimization process and in Section III-C, we discuss overall program control of this process. In Section III-D, we conclude by describing, very briefly, the MULTILEVEL MINIMIZATION process and presenting some limited experimental results.

A. Cost and Delay Estimation

Decisions about what synthesis and optimization steps to take requires the ability to obtain cost and delay estimates for all subfunctions which comprise the multilevel multiple output function.

Cost and delay estimates are obtained by doing a

straightforward mapping on each subfunction in order to generate a gate level representation in terms of the primitives in the user-specified gate library.

The gate library contains two types of elements: implementation primitives and generic elements. The following information is required for each element in the library:

a) the Boolean function defining the cell;
b) the cell area (the number of "transistor pair" units);
c) the terminal count of the cell;
d) delay information for the cell;
e) the equivalent output (load) resistance of the gate;
f) the pin capacitance of each gate input.

Implementation primitives correspond to primitives in the gate library such as NAND2, NAND3, and AOI33. From this library, actual area and delay attributes can be obtained for each element. Generic primitives are ANDS, ORS, and sums of product expressions satisfying fan-in constraints of the implementation primitives. The generic primitives are interim, "dummy" gates which will always be deleted by the OPTIMIZE module discussed in Section IV.

Given a gate level representation, the "AREA" and "WIRES" terms of the cost function in Figure 3 are computed by summing the gate area and terminal count for all gates in the Boolean network. The terminal count serves to estimate the amount of routing that will be needed between gates.

The delay constraint term $\tau(\eta)$ is obtained by performing critical path analysis on the current Boolean network. The delay information needed depends on the level of delay modeling used. When operating in "unit delay" mode, only the set delay through each gate is used. If the "library element gate delay" mode, which incorporates fan-out, is used, it is also necessary to provide the output pin resistances and input pin capacitances.

During straightforward mapping, if the representation of a function or candidate subexpression is in one-to-one correspondence with a primitive in the library, the library-specified area, interconnect, and delay numbers associated with the primitive are used. If there is not a direct mapping, the function is broken down and constructed from a set of generic primitives. Each product or sum is implemented in a few gates as possible without violating fan-in or fan-out constraints. Delay through the configuration is kept down by ordering inputs to each AND or OR representation by their propagation delay. The inputs with the longest delay are assigned to the inputs in the representation with the most "slack" (cf., discussion of (1) below).

The library elements used to implement an intermediate function may implement either the function or its complement. Determining which form of the function is best to implement is known as phase assignment. The best phase for a function depends on the phases available for its inputs and the phases needed by its fan-outs. The overall optimum phase assignment problem is NP-complete. Because of this, neither the mapping nor the cost and delay routines are concerned with determining the best phase for a function. Both phases of all primary inputs and intermediate variables are assumed to be available. We have been investigating optimal phase assignment by simulated annealing, and plan to report our results in a subsequent paper.

The SOCRATES system considers Boolean networks with multiple primary outputs with specified target delays for each output. For simplicity in discussing delays, we shall assume that each Boolean network has a single primary output. The propagation delay through any given function (node of the Boolean network) is the time it takes for a signal to propagate from the primary inputs to the output of the given function. This delay depends on the local propagation time through the gate(s) implementing the function and the propagation delay of the function's inputs. If the implementation of the given function representation requires multiple library gates, their configuration will be based on the arrival time of the functions inputs in order to reduce the total propagation delay. The delay at the output of a given node f is

$$\tau_f = \mathrm{TSET}_f + \max_{i \in \mathrm{FANIN}(f)} (\tau_i - \mathrm{local\_slack}(f, i)). \quad (1)$$

TSET_f refers to the delay through the gates which implement this function. When operating in unit delay mode, this would be one unit for each level of logic implementing the gate. In library element gate delay mode, the gate delay equation shown below is used:

$$\tau_g = \mathrm{TSET}_g + \mathrm{RLOAD}_g^* \sum_{p \in \mathrm{FANOUT}(g)} C_p \quad (2)$$

where C_p is the input capacitance associated with the pin to which g fans out. When the mapping requires multiple gates, the set delay is the sum of the gate delays and load resistance components of the longest path through the configuration. The load resistance is the load resistance of the final gate, i.e., the gate whose output is the value of f. "local_slack"(g, i) refers to the time after start of computation for f that i can arrive before it is considered critical, and is a function of the configuration. It is computed by ordering the inputs by their delays ($O(n \lg n)$, heapsort); those with the most delay are assigned to the input pins with the most local_slack. Also, the total input capacitance of function f fanning out to a function which is realized with multiple gates depends on the number of gates to which f fans out.

The user may specify propagation delays for primary inputs reflecting their relative arrival times. If delays are not specified, they are assumed to be 0.

Prior to computing the delay through each function, the graph is levelized based on function dependencies. A function which depends only on primary inputs is a level 1 function. A function depending on a level i function is a level $i + 1$ function. The delays for the primary inputs are read in or assigned default values of 0. The remaining delays are then computed by levels, starting with level 1.

When the delay of a function changes, it may affect the delay of those functions in its (transitive) fan-out. Changes

are propagated one level at a time until either a primary output has been reached or the delay of a function does not change. The local slack for each function needs to be recomputed as different delays or inputs may result in a different assignment of inputs to pins.

If the library gate delay mode is used, then a "latent" critical path delay calculation is performed to propagate a local delay change, as calculated by (2) through to the primary output. Here we simplify the calculation with the following assumption:

Assumption: Only the (transitive) fan-out of a gate g in a given Boolean network is affected by a change in the delay of a transformed gate.

This assumption actually constitutes an approximation in the library element gate delay mode because the substitutions made by WEAK DIVISION affect the fan-out of gates not in the (transitive) fan-out of the given gate g. Hence, by the calculation of Fig. 4 the delay through these other gates also could, in principle, change. These changes are ignored when computing the cost of candidate substitutions in order to reduce the overall running time of WEAK DIVISION. Of course, once the best candidate has been selected and substituted, an exact critical path delay calculation is made. This latter calculation is still latent, however, in the sense that it startes only at nodes which fan-in to nodes whose delay has actually changed since the last transformation (i.e., substitution).

B. Library-Based WEAK DIVISION

As discussed above, algebraic decomposition or WEAK DIVISION is a method of recognizing the subexpressions which are either common to two or more different functions or factorize, and thus simplify, one individual function.

Example (Weak Division on Two-Level Function):

Initial representation	Modified representation
F_1: $aef + bef + cef$	F_1: Bef
F_2: $aeg + beg + deg$	F_2: Ceg
	A: $a + b$
	B: $A + c$
	C: $A + d$

If $A = F_5$, $B = F_4$, $C = F_3$, and $(a, b, \cdots, g) = (x_1, x_2, \cdots, x_7)$, the Boolean network which corresponds to the modified representation is that of Fig. 2. In the sequel, we shall refer to product terms such as *aef* as **cubes.** Hence, each Boolean function is regarded as a set of cubes. The true or complement form of a Boolean variable (a, $\bar{a}$, b, $\bar{b}$ $\cdots$) is called a **literal.** In the above example, the primary output functions F_1 and F_2 were reexpressed by detecting and creating intermediate variables for common subexpression A and factors B and C, and substituting these into the original representations. The functions in the initial two-level Boolean network depend only on primary inputs. The functions in the multilevel "modified" Boolean network (four-level in this case) are expressed in terms of both primary inputs and intermediate variables. The SYNTHESIS module permits the initial representation to be either two-level or multilevel.

```
Procedure WEAK_DIVISION
Begin
  /* DECOMPOSITION */
  While (common subexpressions exist)
    Generate candidate subexpressions for
      current functions
    Determine eligible subexpressions
    Select "best" disjoint subexpressions
    Associate new intermediate variables with
      subexpressions and substitute
  Endwhile
  Collapse subexpressions referenced by
    only one function

  For each function (FACTORIZATION)
    repeat above loop for single function
  Endfor
End WEAK_DIVISION
```

Fig. 5. WEAK DIVISION algorithm.

As suggested by the above example, the WEAK DIVISION process regards the two-level functions associated with the nodes of the Boolean network as algebraic expressions. The process consists of a set of decomposition steps, each one involving the following substeps:

a) determination and examination of candidate subexpressions;
b) selection of the "best" candidate subexpressions;
c) substitution of best candidate subexpression (which transforms the current Boolean network into an altered one).

This process serves two purposes. First, it enables sharing of logic between the multiple functions. Second, it is the basis for decomposition of the original representation into primitives which are more readily implementable in the desired target technology.

The algorithm for WEAK DIVISION, outlined in Fig. 5 shows that the WEAK DIVISION process is an iterative one. Each iteration has four phases: the generation of candidate subexpressions, the pruning of subexpressions which do not satisfy user constraints, the selection of the best disjoint subexpressions, and the substitution of these subexpressions into one or more of the functions which they divide. The substitution of a subexpression into one or more functions may create new divisors which are expressed in terms of this new intermediate variable. This process continues until there are no subexpressions of sufficient merit to warrant further substitution. WEAK DIVISION is separated into two steps: *decomposition*, the recognition of subexpressions common to multiple functions, and the *factorization* of individual functions. Any intermediate variables which are only referenced by a single function are collapsed back into the function between these steps. If there are still functions which are not read-

```
Procedure KERNEL_GEN (F,start_lit)
  If CUBE_FREE(F) Then
    RECORD_KERNEL(F,LEVEL(F))
    If (LEVEL(F) == 0) Then
      Return
    Endif
  Endif
  For lit←start_lit,last_lit
    If (lit in F)
      subf←F/lit
      subf←subf/COMMON_CUBE(subf)
      KERNEL_GEN(subf,lit + 1)
    Endif
  Endfor
  Return
End KERNEL_GEN
```

Fig. 6. The Kernel construction algorithm.

ily represented in the primitives of the target technology, these are broken down in a straightforward manner.

Algorithms based on those presented in [3] are used to generate to candidate subexpressions and to substitute the selected subexpressions. The algebraic cofactor h of f with respect to $g(h = f/g)$ is a new function defined to be the largest set of cubes (product terms) such that h and g have no literals in common and every term in the Boolean intersection of h and $g(hg)$ is in f, i.e., $f = (f/g)g + r_g = hg + r_g$, where r_g is the appropriate remainder. In the example above, if $g = A = a + b$, then $F_1/g = ef$, and $r_g = cef$. Note if $f/g \neq \phi$, then both g and f/g are divisors of f.

During the decomposition phase, there are two types of candidate subexpressions, those generated by *distillation* and those generated by *condensation*. There are parameters to guide which type to generate during each pass. Options exist to generate both or to generate distillation terms until the merit drops below a user-specified value and then to generate both.

Candidate distillation expressions are subexpressions of **kernels** which appear in more than one function. The **kernels** K of an expression f are formally defined as a set of cubes $K(f)$ which satisfies

$$K(f) = \{f/c\text{: } c \text{ is a cube and } f/c \text{ is } \textbf{cube free}\}. \quad (3)$$

An expression is **cube free** if there is no literal which appears in every cube. Note that a single product term is not cube free; thus, a kernel is defined to contain two or more product terms.

In the Example, the only kernel of F_1 is $a + b + c$, which is F_1/ef. F_1/e is not cube free and F_1/a, F_1/b, F_1/c do not contain two or more cubes.

The kernel construction algorithm shows in Fig. 6 is a recursive algorithm. Initially, F is the function to obtain kernels for and *start_lit* is set to 0. A level 0 kernel K is defined to be one where $K(K) = K$. A level $n + 1$ kernel is one where $K(K)$ are all of level $\leq n$. In a level 0 kernel, no literal appears in more than one cube. The kernels for each function are generated independently, but all kernels are stored in a common kernel table whose rows correspond to the kernels and whose columns correspond to the functions divided by the kernel.

After generating kernels for all functions, additional

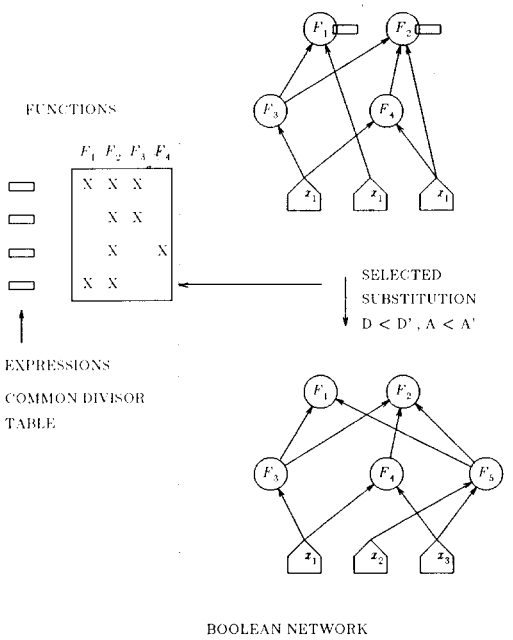


Fig. 7. WEAK DIVISION process.

common divisors are found and added to the kernel table. A common divisor is any $K_i \cap K_j$ such that K_i is a kernel from function i, K_j is a kernel from function j, and there are at least two cubes which appear in both kernels. Associated with each divisor are the cubes comprising the divisor and the functions which it divides.

The complement of each kernel is also generated and each function is checked to see if it can be divided by the complement. Those kernels which do not divide multiple functions will not be considered for substitution. The above process is illustrated in Fig. 7. At the top right is shown the Boolean network prior to the selection and substitution of a common divisor. At the left, a table is shown where the rows correspond to common divisors and the columns to functions (nodes of the Boolean network). The entries X may take one of three values, indicating the "signature" of the expression, i.e., whether the division function occurs in true form, complement form, or in both forms in the function associated with its column. No entry means the divisor does not occur in that function (column). Divisors not common to two or more functions are not present in the table. The cost is computed, as described in Section III-A, for each candidate substitution, and the resulting transformation of least cost is selected.

Note that, generally speaking, divisors which are common to more functions provide greater area savings. However, in the case illustrated in Fig. 7, a divisor common to only two functions (F_1 and F_2) is selected, despite the existence of a divisor (the first) common to three functions (F_1, F_2, and F_3). This is because of the connectivity of the Boolean network. Substitution of the latter divisor would result in an extra level of logic. This subexpression would not be considered if it resulted in violating any delay constraints.

Candidate condensation subexpressions are cubes which can cofactor multiple functions. A condensation table is

constructed containing all such cubes. Associated with each are the literals comprising the cube and the functions which it can cofactor.

A function f can be reexpressed in terms of a subexpression g if f/g is not empty. The complement of the candidate subexpression is also considered for substitution. Thus, as a result of substitution, we have a new representation of the function f in the form $f = (f/g)\ G + (f/\bar{g}) \cdot \bar{G} + r_g$, where g is the expression to be substituted and G is the literal associated with it. As a subexpression may have been substituted into other functions on a previous pass, it is necessary to check if it has already been implemented. If neither the subexpression or its complement exist in the function table, the subexpression is added and associated with a unique literal. All eligible occurrences of the subexpression and its complement are then replaced with the corresponding literal or its complement. Two subexpressions g and h are disjoint if $(f/g)\ g$ and $(f/h)\ h$ do not contain any of the same cubes or if $f/g \supseteq h$ and $f/h \supseteq g$.

As the common subexpressions are typically quite small, the number of kernels to consider can be reduced greatly by considering only level 0 kernels. This also speeds up the kernel intersection process. Often, the more complicated common divisors can be obtained by back substituting (collapsing) any literals referenced by a single intermediate variable.

The algorithms used for factorization are very similar to those used for decomposition. Only one function is considered at a time and the candidate subexpressions are the level 0 kernels of each function.

C. Program Control of WEAK DIVISION

The objective of our WEAK DIVISION process is to select those subexpressions that will yield the best implementation in the specified target technology. After each generate phase, merit and delay increment estimates are obtained for the candidate subexpressions. The best disjoint subexpressions become intermediate functions. Different candidate subexpressions will generate alternate representations of the functions into which they can be substituted. Selecting one intermediate subexpression may eliminate other candidate subexpressions.

The user may influence the final propagation delay by specifying delay constraints. These prevent, regardless of merit, the substitution of any candidate subexpressions which would increase the delay of any critical primary output functions by more than a user-specified amount. There are several different user-specified ways a primary output function can be considered critical:

a) exceeds a certain absolute delay figure;
b) exceeds the maximum initial delay;
c) exceeds the maximum current delay.

Given a criteria for determining if a primary output function is critical, the slack associated with each primary output is calculated as follows. If the initial network is multilevel or if kernels of level >0 are allowed, it is necessary to determine how much a change in the delay of an intermediate function will affect the delay of the primary output functions in its transitive fan-out. This can be done by maintaining an array showing the longest path between each primary output and each intermediate function. The slack σ_j associated with each intermediate function F_j is

$$\sigma_j = \underset{i \in PO}{\text{MIN}} \{\sigma_i + P(i) - (P(j) + LP(i, j))\} \qquad (4)$$

where PO denotes the set of primary output indices in the transitive fan-out of j, $P(j)$ is the length of the longest path from any primary input to the intermediate function F_j, and $LP(i, j)$ is the length of the longest path to primary output function F_i from intermediate function F_j.

If substituting a candidate subexpression into a function causes a delay increase which is greater than the slack calculated for the function, then the subexpression is no longer considered for substitution in this function.

The merit $M(G, F)$ of substituting a subexpression G into a specific function F is the difference in cost between the original (f) and the revised representation $(f' = (f/g)\ G + (f/\bar{g})\ \bar{G} + r_g)$, i.e.,

$$M(G, F) = \text{COST}(f) - \text{COST}(f'). \qquad (5)$$

If the subexpression G is to be substituted into the functions F_j, $j \in J_G$, where J_G is the set of functions G can cofactor without violating delay constraints, then the total merit $MT(G)$ is given by

$$MT(G) = \left(\sum_{j \in J_G} M(G, F_j) \right) - \text{COST}(G) \qquad (6)$$

where COST(G) is as defined in Section III-A.

D. Multilevel Function Minimization

As WEAK DIVISION is a strictly algebraic process, the network may contain intermediate or primary output functions that are functionally equivalent. Because of "don't care" conditions inherent to multilevel Boolean networks, functions which are prime and irredundant in the two-level sense may be nonminimal in the multilevel sense. As a result, the network may also contain functions which can be more efficiently expressed in terms of a different set of intermediate functions. To exploit these possibilities, we now discuss a technique we call "MULTILEVEL MINIMIZATION."

The "two-level" notions of primality and irredundancy are yet to be rigorously defined in the multilevel context, but the don't care methods described below establish a promising multilevel minimization technique for two-level minimization. In fact, if the full don't care set described below is used, then there is a meaningful sense in which the resulting network may be called prime and irredundant. We plan to explore these matters more fully in a later paper [17]. For now we content ourselves by thinking of MULTILEVEL MINIMIZATION as a process which, like WEAK DIVISION, reduces the cost function of Fig. 3 by a series of local transformations. In this case,

the transformations are those made by the ESPRESSO logic minimizer given the multilevel don't care set discussed below.

In two-level minimization, we are given $F: B^n \rightarrow B$ and attempt to minimize $\| F \cup D_x \|$, where $\| \cdot \|$ is some measure of term count (circuits) and literal count (transistors) in the given cubical cover of F. Here, $D^x = \{x \in B^n |$ "x never occurs as input"$\}$ is the so-called don't-care set [4].

In multilevel minimization, we are given a set of primary inputs x_i, $i = 1,2, \ldots, n$, and a set of intermediate variables $y_k = F_k(x,y)$ $k = 1,2, \ldots, m$. Note that here $F_k: B^{m+n} \rightarrow B$. The y_k can be primary outputs or literals representing intermediate variables which are output from node k of the Boolean network example of Fig. 1.

The "intermediate" don't care set associated with function F_j of the Boolean network is given by

$$D_j^I \equiv x_j \bar{F}_j(x,y) + \bar{x}_j(x,y) . \tag{7a}$$

Simply stated, D_j^I gives the set of vertices of B^{n+m} which have inconsistent combinations of the components of x and y vectors, i.e., combinations which cannot occur, and are, therefore, "don't care." The totality of such intermediate don't care conditions

$$D^I = \sum_{j \in 1,2,\ldots,m} D_j^I \tag{7b}$$

is known to be potentially prohibitively large and, when minimizing function F_k of the Boolean network, is usually replaced by a subset D^{I_k}, defined by

$$D^{I_k} = \sum_{j \in j_k} \sum_{i \in I_j} D_i^I \subseteq D^I \tag{7c}$$

where I_j is a set of node indices and J_k is a set of set indices. The purpose of the double summation is to permit sufficiently powerful subsets of D^I to be expressed which, it is hoped, are minimally sufficient for the complete minimization of F_k. We are currently using the transitive fan-out of the transitive fan-in of node k, minus the transitive fan-out of node k for that purpose.

In terms of these definitions, the multilevel don't care set for function F_k can be expressed as follows:

$$D_k = D^z \cup D^{I_k} \cup D^{O_k} \tag{7d}$$

where D^{O_k} stands for the "output" don't care set [7], which we have neglected in the work described below. The effects of D^{O_k} will be discussed in a later paper [17]. If D^{O_k} is neglected, only primary output functions will necessarily be rendered prime and irredundant by our procedure. However, any of the F_k may possibly be simplified, thus reducing the cost function of Figure 3, whether or not they are primary outputs.

Assuming that $D^{I_k} \equiv D^I$ and that node k of the Boolean network is a primary output, for which $D^{O_k} = \Phi$, then it can be shown that (7d) gives the complete don't care set for function F_k. If a complete don't care set is used, for each F_k, then after ESPRESSO is applied to function $F_k \cup D_k$, for all $k = 1,2, \ldots, m$, the resulting Boolean network can be shown to be prime and irredendant in the sense that no cube or product term of any functions can be deleted without altering the Boolean functionality of the Boolean network.

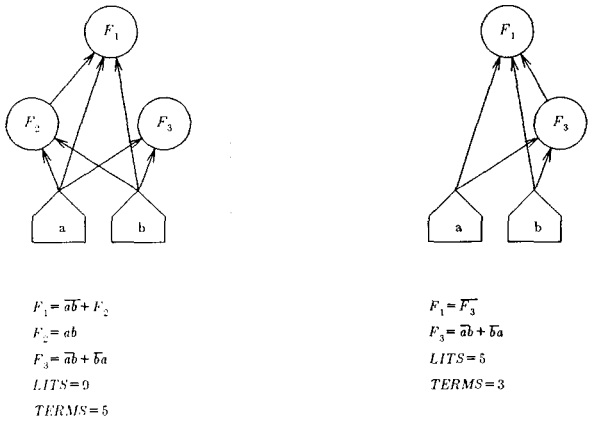


Fig. 8. MULTILEVEL MINIMIZATION example.

However, with any don't care set which is complete in the sense described above, the adjacency relations of the Boolean network may be altered as shown in the example of Fig. 8. It is of interest to observe that in this example the starting representation (on the left) is prime and irredundant. Thus, the first EXPAND and IRREDUNDANT_COVER operations in ESPRESSO will have no effect. However, after the REDUCE operation is performed, the optimal result will be obtained in the second EXPAND step. We observe, in fact, that REDUCE is performing a major part of the role of the minimization process referred to as Boolean substitution in [6]. This effect offers a major avenue for future research.

IV. Logic-Level Synthesis and Optimization

The final step in the synthesis of a circuit is to implement the optimal multilevel function at the logic level. The challenge here is to make the best possible use of the target technology. The approach taken to this problem is to first map the multilevel function to a logic-level circuit, and then to optimize the synthesized circuit. Since this module of the synthesis system has been described in earlier publications [10], we will merely summarize in this paper its workings and concentrate on the timing specific parts.

A. Initial Mapping to Target Technology

The translation of a multilevel function to a circuit is a straightforward operation. In order to make this translation tool technology-independent, the initial circuit is always implemented using the same small subset of library gates. If these gates do not actually exist in the target technology, then a high cost is assigned to them to force their later removal.

B. The SOCRATES Rule-Based System

The optimization of the circuit at the gate level is done by a rule-based system. A knowledge-based approach, rather than an algorithmic approach, was used in order to provide a paradigm for the use of specialized knowledge about the target technology. A circuit is optimized by per-

forming *local transformations* on it: each transformation replaces one small configuration of gates with another configuration that is functionally equivalent. A cost function similar to that described in Section III is used to assess the merit of the replacement. Any given transformation may reduce the cost of the circuit from the standpoint of either area, performance (i.e., delay), or both.

Knowledge acquisition has traditionally been difficult in existing rule-based systems. A *rule entry module* aids the knowledge engineer in adding new local transformations to the system. This module automatically verifies all new rules for functional correctness and orders them in the knowledge base. The ordering is determined by the area and delay benefits associated with the rule in question and is also a function of the relationship of the transformation to the rules already in the knowledge base.

During the optimization, the system performs four sets of tasks.

1) A pattern matcher is used to find which rules apply on the circuit. Generally, more than one rule can be applied at any time during the optimization.

2) For each potential rule application, a cost function is computed to determine the quality of the resulting circuit.

3) A selection mechanism decides which rule is to be applied next.

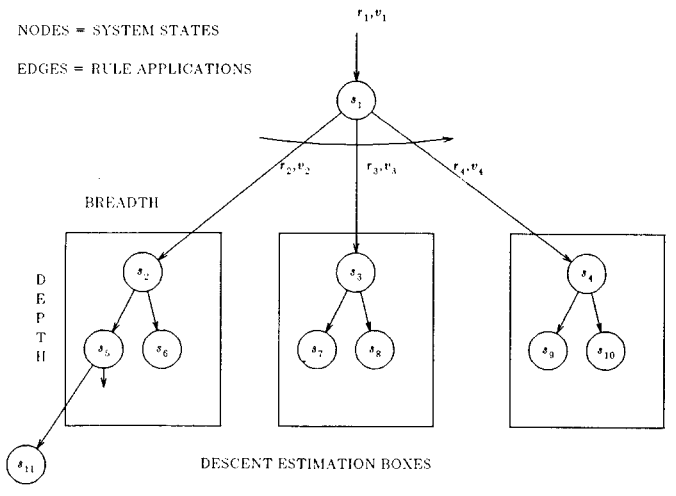
4) The rule is applied by constructing the new gate configuration on top of the old configuration, removing the old one and adjusting the cost function to its new value.

C. The Search Strategy

The actual sequence of transformations that will be used is determined by a control module which "looks ahead" several rule applications in order to select the most useful set of transformations. This is done in order to avoid local minima of the cost function. Although the amount of CPU time required increases rapidly with increased breadth and width of the search space explored for each move, it was found that on the average the resulting circuit size can be reduced by an additional 10 percent using a look-ahead strategy. The search space explored is the space of Boolean networks which can be obtained from the initial library mapping by some sequence of rule applications. If the Boolean network has N_V nodes and there are N_R rules in the rules library, there are, potentially

$$N_V \sum_{k=1}^{N_R} k!$$

such sequences. We explore a meaningful subspace of this potentially large state space by a search algorithm that is illustrated in Fig. 9. This algorithm is controlled by a number of parameters, the most important of which are breadth (the number of rule sequences explored from a given network configuration) and depth (the length of the rule sequence to be explored).



SEARCH METHOD

1. LOOK AT ALL STATES IN EACH SON'S ESTIMATION BOX
2. TAKE BEST RULE (r_2) TO NEW STATE (s_2) AND RECUR
3. IF BOX CONTAINS SUFFICIENTLY MERITORIOUS STATE
4. THEN SEARCH RECURS FROM s_5

Fig. 9. State space search.

D. Meta-Rules

During the development of the optimizer, it became apparent that the search parameters should be dynamically adjusted during optimization to improve results and run time. A second rule-based system was therefore built to control the search strategy of the system. A set of *meta-rules* modifies the control parameters of the search mechanism based on a set of diagnostics from the system. While the transformation rules used reflect the expertise of the circuit designer, the meta-rules capture the experience of the designer of the optimization system. The meta-rule system has proven extremely valuable while experimenting with a variety of search strategies.

Just as in Fig. 7, each "local transformation" may be regarded as changing the "state" of the Boolean network, or, equivalently, transforming the given Boolean network into an alternate Boolean network. This alternate network realizes the same (given) Boolean logic functions, but at lower cost (cf., Fig. 3). On this view, the objective of the expert system is to find the "state" (i.e., Boolean network) with minimum cost.

The rule-based system thus "searches" the space of all possible states, systematically, under meta-rule control. Suppose the system reaches state s_1 of the state space by applying rule r_1 to node v_1 of the previous Boolean network. The META_RULES subsystem then selects an ordered application sequence which explores a set of "descent estimation boxes." The number of these boxes is a breadth parameter determined by the META_RULES subsystem, as are the breadth and depth parameters which characterize each estimation box. Each such box comprises a subspace of the overall state space. The look-

ahead procedure consists of applying a sequence of rules which determines the state of least cost in each descent estimation box. If the box initiated by applying rule r_2 to node v_2 has less estimated cost than that of the boxes associated with the pairs r_3, v_3, r_4, $\cdots$, then state s_2 is selected as the next official state, and the process recurs with s_1 replaced by s_2. In exceptional cases, if the cost of some state, say s_5, is sufficiently low, then the process recurs from s_5 instead of from s_1.

It is significant that the nodes v_2, v_3, v_4 $\cdots$ selected by the META_RULES subsystem are chosen from the immediate neighborhood, in the Boolean network of state s_1, of the original active node v_1. This strategy promotes early competition between rule applications. Such competition is ultimately necessary, since OPTIMIZE terminates only when no further rule applications are possible. Early competition gives the META_RULES subsystem the opportunity to optimize rule tradeoffs.

E. Optimization of Delay

In order to assess the timing benefits of applying a rule, a critical path algorithm is used to compute the typical-case delay times of each signal and the slack time of each gate. Like the critical path finder used in the multilevel synthesis modules, this critical path algorithm is **latent,** i.e., it operates incrementally for greater efficiency. Whenever a transformation is performed, the critical path finder restricts its operation to the gates in the circuit that are descendants or ancestors of an altered gate. This ensures that no time is spent recomputing delay information that didn't change. Since the path finder is working on an actual circuit, the predicted path lengths will now be exact (modulo the accuracy of the library element delay equations) with the exception of the interconnect delays, for which a simple approximation is used.

The user can specify the timing constraints in the form of signal arrival times at the inputs and desired maximum delays at the output. The drive limitations of the logic driving the inputs of the synthesized circuit as well as the loads connected to the output of the synthesized circuit are also specified by the user. The system will synthesize the smallest circuit that meets the timing constraints, or the circuit that comes closest to meeting them. The end product of SOCRATES is a netlist of an automatically generated schematic.

1) Optimization Phases: The process of gate-level optimization proceeds in three main phases. In the first phase, only transformations that reduce both the delay and area cost of the circuit are applied. This phase produces a circuit that is appropriate to the target technology and that strikes a balance between area efficiency and time efficiency. In the second phase, the circuit is optimized for delay only by applying delay-saving rules to the gates along the critical path until the timing constraints are met or the optimizer can do no better. In the last phase, area-saving transformations are used on gates off the critical path. These transformations will reduce the area of the circuit without changing the delay of the circuit.

2) Delay Improving Rules: There are several ways in which local transformations can speed up a circuit.

- *Faster, Similar Gates.* Often logic functions can be implemented in a variety of ways resulting in the same area but different timing behavior. For instance, in CMOS, a NAND/NAND implementation tends to be faster than a NOR/NOR implementation; XNOR gates tend to be faster than XOR gates.
- *Break-Up Large Gates.* Performance improvements can be obtained by replacing larger gates with sequences of smaller, faster gates. The timing benefits of such replacements are strongly a function of the configuration of neighboring gates and can only be assessed by actually recomputing the resulting loading delays of the new circuit.
- *Move Critical Signals Closer to Output.* This is especially useful in cases where one or more input signals arrive late. Reducing the number of levels of logic that such a signal has to traverse to arrive at the output can significantly improve the overall timing of the circuit.
- *Buffering.* Finally, buffering heavily loaded gates appropriately tends to improve the performance of a circuit.

V. Computational Results

We will now describe three types of experimental results. First, the results of applying the SOCRATES system to a set of the ESPRESSO book PLA's [4] which can be thought of as a set of "real world" multiple output, two-level, combinational logic functions. We then demonstrate the flexibility and technology independence of our system with some detailed case studies of specific PLA's (RD53 and F2). Finally, we conclude with some experiments designed to illustrate basic compatibility and detailed interaction of the SYNTHESIS and OPTIMIZE modules.

A. Experiments on ESPRESSO Book PLA's

Both WEAK DIVISION and OPTIMIZE can be run in either area or delay mode. (Note that the other SYNTHESIZE modules, e.g., MULTILEVEL MINIMIZATION and LIBRARY MAPPING, also depend on the cost function and, hence, have both area and delay modes also.) In area mode, the objective is to minimize area independent of delay. However, in delay mode, the objective, roughly speaking, is to minimize area subject to constraints on delay. Table I shows the results for 24 examples. Four experiments were performed on each example:

1) WEAK DIVISION in area mode followed by OPTIMIZE in area mode (A,A);
2) WEAK DIVISION in area mode followed by OPTIMIZE in delay mode (A,D);
3) WEAK DIVISION in delay mode followed by OPTIMIZE in area mode (D,A);
4) WEAK DIVISION in delay mode followed by OPTIMIZE in delay mode (D,D).

We will subsequently refer to the four modes of operation corresponding to these four experiments as AA, AD, DD,

TABLE I
EXAMPLES OF CONSTRAINTS ON DELAY

	Area				Timing			
name(cpu min)	(a,a)	(a,d)	(d,a)	(d,d)	(a,a)	(a,d)	(d,a)	(d,d)
mark(0.7)	9	9	21	20	3	3	8	4
f1(1.0)	19	23	20	25	6	5	7	5
clpl(1.0)	20	20	51	48	17	17	12	10
f0(0.4)	26	29	26	29	10	7	10	7
gerf(0.8)	26	25	31	33	12	10	12	10
f2(4.0)	32	32	56	61	6	6	9	4
fadd(1.7)	39	39	46	47	12	12	13	9
adde(1.6)	56	57	56	57	19	18	19	18
dec1(5.1)	73	77	77	84	12	10	10	6
z4(9.3)	76	76	108	117	13	12	22	16
rd53(6.4)	89	95	85	90	22	16	15	11
f5(14.)	97	112	103	124	14	11	14	11
exam(13.)	98	108	112	127	13	10	14	9
f3(8.3)	99	107	99	114	14	13	14	9
f4(9.3)	103	113	108	118	13	10	16	10
8fun(14.)	107	127	125	141	14	13	15	13
plab(42.)	158	176	165	192	18	14	17	14
5xp1(89.)	191	217	220	254	24	19	21	16
dec2(66.)	203	219	215	243	22	17	22	16
f51m(63.)	209	235	256	285	25	19	24	18
root(63.)	234	247	269	287	27	22	43	29
plac(51.)	249	288	303	337	22	17	23	16
bw(192)	286	311	283	307	16	14	17	14
9sym(54.)	364	380	397	426	37	25	50	38
average	119	130	134	148	17	14	18	13

TABLE II
AVERAGES OF AREA AND DELAY VALUES

		WDIV	
	Area/Delay	Area Mode	Speed Mode
OPTIM	Area Mode	119.4/16.6	134.7/18.3
	Speed mode	130.1/13.8	148.6/13.5

TABLE III
EFFECT OF MULTILEVEL MINIMIZATION

	AA		DD	
name	Area	Delay	Area	Delay
fadd	32	9		
adde	59	14		
dec1	69	10		
z4	58	16	61	14
rd53	82	15		
f5	95	14	109	8
exam	94	11	116	10
f4	103	9	124	8
8fun	110	13	128	10
plab	176	15		
dec2	201	20		
plac	256	26	336	15

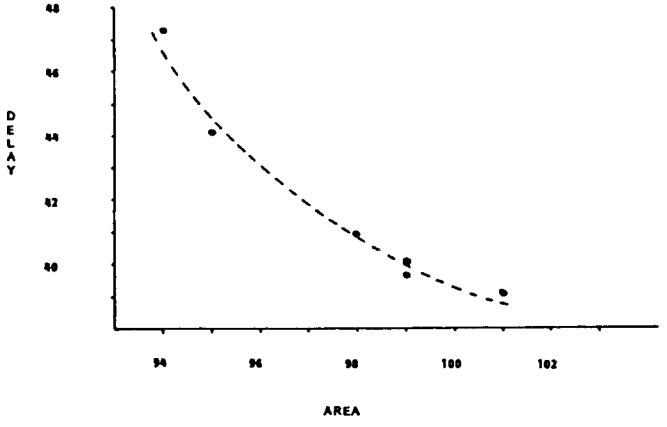


Fig. 10. Area Speed Tradeoffs of RD53.

and DD. In all four experiments, WEAK DIVISION was run with the "unit delay" delay model.

The first half of Table I lists the resulting area values; the second half of the table shows the obtained delay values. Area refers to the number of transistor pairs, delay is in nanoseconds, and (cpu min) refers to the number of VAX 780 cpu minutes to run WEAK DIVISION and OPTIMIZE.

The results are summarized in Table II, which gives the averages of the area and delay values. It is immediately apparent that the best area optimization is obtained by having both modules work in area mode; the best delay optimization is achieved using the two modules in delay mode. Between those two cases, we observe an area increase of 20 percent and a delay decrease of 20 percent. It should be noted that there are some cases (z4, root, 9sym) where a better delay number is obtained for WEAK DIVISION operating in area mode. Section V-C will analyze reasons for these discrepancies and present some partial remedies.

Table III contains a subset of the Table I examples where MULTILEVEL MINIMIZATION was able to further optimize the output. *AA* corresponds to running MULTILEVEL MINIMIZATION on WEAK DIVISION-A output and then running OPTIMIZE-A. *DD* corresponds to running MULTILEVEL MINIMIZATION with delay constraints on WEAK DIVISION-D output and then running OPTIMIZE-D.

Comparing the two tables, it can be observed that, in the 12 *AA* examples, better area numbers (than AA) were obtained in eight cases. Three of the other cases illustrated better area delay tradeoffs. In the 6 *DD* examples, the delay was reduced (relative to DD) in all but one case (exam2). These numbers indicate the MULTILEVEL MINIMIZATION is often a valuable step to take in the optimization process.

B. Flexibility and Technology Independence

To show the system's flexibility, we took one example and ran it with increasingly severe timing constraints. That is, the required arrival times at circuit outputs were progressively reduced. The results in Fig. 10 show the area/delay tradeoff for example RD53, and demonstrate how SOCRATES can be used to explore a design space quickly and easily.

To demonstrate the technology independence of SOCRATES, when we changed the description of our two input NOR from small and fast to large and slow in the library and ran an example again, SOCRATES replaced all of the original NOR gates with NAND gates and inverters. The schematics for both the original (on the left) and the new circuit (at the right) are shown in Fig. 11. The new implementation without the NOR's is clearly less efficient.

C. SYNTHESIS–OPTIMIZE Interactions

The results of Table I and II clearly establish the overall benefits of the partnership between OPTIMIZE and

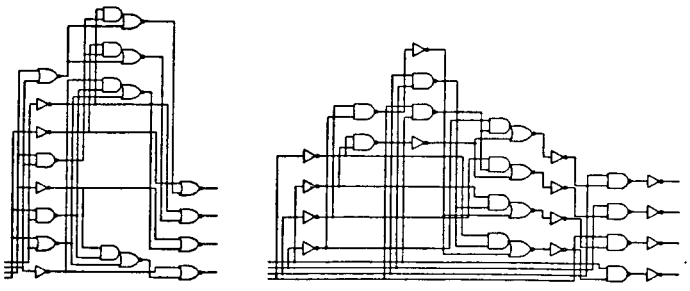


Fig. 11. Example change in technology for F2.

WEAK DIVISION. However, it is of interest to explore further the detailed interaction between modules. One would expect that the AA case (both modules in area mode) would always have the lowest area and that the DD case (both in delay mode) would always have the lowest delay. This is always true in the area case. However, as can be observed in circuits z4, root, and 9sym of Table I, there are discrepancies in the delay case. Examining the differences between AD and DD indicates that when OPTIMIZE is in the delay mode, whichever mode of WEAK DIVISION is used has significant, but mixed, impact on the delay of the final circuit. Subsequent experiments discussed below have shown that, while WEAK DIVISION was always very useful in reducing the area and was effective in reducing the average delay, it was actually detrimental to delay reduction in a few cases.

The delay and area differences between AA and DA or AD and DD can be viewed as indicating how effective WEAK DIVISION is at influencing the delay/area. Similarly, the differences between AA and AD and DA and DD indicate how OPTIMIZE can influence the delay. A tie between AA and AD or DA and DD indicates that the circuit is such that optimizing the area also optimizes the delay. A tie between AD and DD or AA and DA indicates that OPTIMIZE is so effective that it can obtain the best delay/area independent of what WEAK DIVISION does to the input circuit. There are (at least) four possible reasons which may account for these apparent inconsistencies between the WEAK DIVISION and OPTIMIZE components of SOCRATES.

1) limited rule base;
2) WEAK DIVISION unit delay model does not incorporate fan-out or technology dependence;
3) AND/OR translation in the LIBRARY MAPPING module obscures the WEAK DIVISION architecture;
4) different assumptions about forms available for primary inputs

All of these are directly tied to our initial assumption that it would be possible for WEAK DIVISION to operate in a technology-independent manner, using very crude area estimates. Thus, WEAK DIVISION would have the task of efficiently decomposing the original two-level network which could be optimally mapped by OPTIMIZE. Realization that architectural decisions with global implications during WEAK DIVISION were not readily undone by OPTIMIZE's local transformations inspired the technology-independent unit delay model. Clearly, if there were an infinite rule base, OPTIMIZE could obtain the best final implementation from any representation, but this would require much larger global transformations and may not be feasible in terms of the size of the rule base or in computation time.

As WEAK DIVISION produces a multilevel PLA and OPTIMIZE requires a netlist as input, it is necessary to have a translation program in between them. Based on the above assumptions, the initial LIBRARY MAPPING module used in producing the data of Tables I and II mapped each function in the multilevel cover to a representation consisting of inverters, two input AND gates (AND2), and two input OR gates (OR2). OPTIMIZE views AND2 ad OR2 as dummy gates; they are in the library but have been given very high cost to assure that rules will be applied to replace them. What appears to be happening is that, once larger gates (such as AND4 and XOR) are broken down into two input gates, they are not being recovered by the OPTIMIZE rules so the architecture (or "logic structure") generated by WEAK DIVISION is obscured or lost.

There are (at least) two ways to test and/or remedy this situation. One would be to assure that the rule base contains rules to regenerate gates corresponding to the functions that were decomposed. The second would be to have the input to OPTIMIZE more closely correlate to the output generated by WEAK DIVISION. This requires more technology-dependence in the LIBRARY MAPPING module. After WEAK DIVISION, almost all of the functions satisfy the fan-in constraints of the gates in OPTIMIZE's library. By introducing additional intermediate functions, those that don't satisfy fan-in constraints can be transformed in a straightforward manner to functions that do. A new LIBRARY MAPPING module, Exact_Map, has been written which maps each function into the corresponding gate adding inverters where necessary.

Experimental results with the new LIBRARY MAPPING modules have produced more consistent results. Lower delays have been obtained running all modes, and WEAK DIVISION appears to be having more of an effect on the delay. OPTIMIZE also ran considerably faster with an input consisting of the larger gates. We believe even better area delay tradeoffs could be obtained if there were rules to decompose the large gates in order to investigate alternate configurations.

The original delay model utilized by WEAK DIVISION was the unit delay model. In estimating the delay, functions were decomposed into "gates" satisfying the fan-in constraints of the library and one unit was charged for each gate. (Inverters were not considered.) This model does not take into account load delay, which is often a very important contributor to the total delay through a circuit. Experiments using the library element gate delay model of Fig. 4, which incorporates load delay, have produced more consistent results, in which the WEAK DI-

TABLE IV
EFFECT OF INCREASING MODELING DETAIL IN SYNTHESIS MODULE (12 EXAMPLE AVERAGES)

SYNTHESIS Sequence	τ_{CP}(nS)/ Area Ratio				$\Delta\tau$(WD)	Anomaly
	AA	AD	DA	DD	$\tau_{AD} - \tau_{DD}$	Count*
WD1/MAP1	13.2/1.9	11.1/1.8	15.5/1.7	11.0/1.5	0.1	6(1)
WD1/MAP2	12.1/1.9	9.6/1.8	11.5/1.8	9.2/1.6	0.4	6
WD1/MAP3	11.3/1.9	9.5/1.8	11.0/1.7	8.8/1.6	0.7	2(3)
WD2/MAP1	12.3/1.9	9.8/1.8	12.9/1.6	9.2/1.5	0.6	4
WD2/MAP3	11.2/1.9	9.6/1.8	10.5/1.5	8.2/1.4	1.4	2
WD3/MAP3	11.2/1.9	9.6/1.8	10.9/1.6	8.7/1.4	0.9	4

Area Ratio $= \frac{\text{AREA(phiD)}}{\text{AREA}}$.

*Anomalies: Individual cases in which WEAK DIVISION pays penalty for area reduction

$$\tau_{AD} - \tau_{DD} < 0.$$

Ties (if any) in parenthesis.

TABLE V
OPTIMIZE WITH AND WITHOUT WEAK DIVISION

Sequence	AA	DD	
-/MAP1/OPT	142.28 11.53	152.07 8.59	(area) (τ_{CP}(nS))
WD2/MAP3/OPT	80.71 11.22	100.21 8.24	
%improvement	43 2.6	34 5.1	

VISION mode has more influence on the final delay. To see how much of an effect load delay has, we recomputed the delay of the Boolean networks created by WEAK DIVISION in unit delay mode. This experiment showed that almost all of them had exceeded the original input delay rather than decreasing it.

Another possible inconsistency between the two programs is that, while WEAK DIVISION assumes that all primary inputs are available in both their positive and complemented forms, OPTIMIZE does not automatically make this assumption. Examining a few examples manually has indicated that a large portion of their delay is due to the complement of a primary input being used in many places, thus having a very high load delay component. By adding additional primary inputs corresponding to the complement of each primary input and specifying they are complements, OPTIMIZE will not need to generate the complement. Early experiments indicate this may reduce the amount of inconsistency.

Tables IV and V show the effect of correcting some of the possible SYNTHESIS module deficiencies discussed above. To this end, we present the results produced by two successively more sophisticated versions of both the WEAK DIVISION and LIBRARY MAPPING modules. We shall refer to the (technology-independent) WEAK DIVISION module, running with the "unit delay" delay model, as "WD1." When the effects of fan-out are introduced as in Fig. 4, but all gates are assumed to have the same set delay, load resistances, and input pin capacitances, we shall mark the experiments "WD2." When the full technology dependence of the "Library Element gate delay" delay model is used, we shall mark the experiments "WD3."

A similar sophistication level is attached to the three versions of the LIBRARY MAPPING module. The Basic AND/OR mapping scheme is marked "MAP1," More sophisticated mapping schemes which try successively harder to retain the logic structure supplied by WEAK DIVISION are marked "MAP2" and "MAP3," respectively.

The results of running a subset of the 24 problems of Tables I and II are presented in Tables IV and V. The rows of Table IV are marked with the appropriate labels for the WEAK DIVISION and LIBRARY MAPPING modules employed. The average delay (in nS) and area ratio constitute the first four columns of data. Area ratio is the OPTIMIZE (delay mode) output area (no WEAK DIVISION, AND/OR mapping) divided by the area obtained using the sequence specified in column 1. This is a measurement of how much WEAK DIVISION influences the final area. This data shows that, generally speaking, final average delays are reduced by increased sophistication of the SYNTHESIS module, while area improvement ratios are slightly reduced. Note that in all cases, the WEAK DIVISION module produces at least a 40 percent area improvement.

The last two columns address the presence of "anomalies," i.e., delay counter-productive WEAK DIVISION. It is observed that increased technology dependence is markedly helpful in reducing the anomaly count, as well as significantly enhancing the delay improvement which can be ascribed solely to the WEAK DIVISION module (1.4 nS i the WD2/MAPS case). Interestingly though, the best results are **not** obtained by the most sophisticated SYNTHESIS sequence.

Finally, Table V summarizes the data comparison with and without WEAK DIVISION. The first row of the table shows the result of **no** WEAK DIVISION. In contrast, the second row gives results for the (best) WD2/MAP3 SYNTHESIS sequence. The third row shows the percent improvement for both area and delay. It can be seen that WEAK DIVISION clearly aids in improving both the area and delay when run in either area or delay mode.

VI. CONCLUSIONS

Both the SYNTHESIS (WEAK DIVISION and MULTILEVEL MINIMIZATION) and OPTIMIZE modules are effective at trading off area for delay. The OPTIMIZE module, which operates with a finite rule set and a heuristic state space search, is effective stand alone, but attains better local minima when aided by the optimized decomposition provided by WEAK DIVISION. It is interesting to note that SYNTHESIS on the average, is effective despite operating with relatively crude delay models. Apparently, this is because SYNTHESIS oper-

ates on a higher, more technology-independent level of abstraction. More consistent results, with better delay improvements, can, however, be obtained by adding more technology dependence to the LIBRARY MAPPING modules and incorporating fan-out in the SYNTHESIS delay models (cf., Tables IV and V). These latter results challenge our original paradigm of SYNTHESIS as a purely "architectural" (and, hence, technology-independent) logic optimization process.

In future work, we plan to improve our rule base with more architectural level rules, i.e., ones which are more capable of directly exploiting parallelism to enhance delay. We also plan to investigate still more sophisticated SYNTHESIS modules, including more detailed LIBRARY MAPPING modules, as well as putting WEAK DIVISION in an iteration loop with OPTIMUM PHASE ASSIGNMENT, MULTILEVEL MINIMIZATION, and finally SELECTIVE FLATTENING (which provides WEAK DIVISION with further opportunities for logic optimization).

ACKOWLEDGMENT

We wish to thank D. Gregory, T. Hefner, V. Morgan, and J. Reed for their contributions and helpful comments in regard to the development of the OPTIMIZE rule-based system and the automatic schematics generation. We also wish to acknowledge the technical suggestions and programming assistance of R. Jacoby on the development of WEAK DIVISION and MULTILEVEL MINIMIZATION packages.

REFERENCES

[1] K. A. Bartlett and G. D. Hachtel, "Library specific optimization of multilevel combinational logic," in *Proc. IEEE Int. Conf. on Computer Design*, Oct. 1985.

[2] K. A. Bartlett, W. W. Cohen, A. J. DeGeus, and G. D. Hachtel, "Synthesis and optimization of multilevel logic under timing constraints," in *Proc. IEEE Int. Conf. on CAD*, Nov. 1985, pp. 290–292.

[3] R. K. Brayton and C. T. McMullen, "The decomposition and factorization of Boolean expressions," in *Proc. Int. Symp. on Circuits and Systems*, 1982, pp. 49–54.

[4] R. K. Brayton, G. D. Hachtel, C. T. McMullen, and A. Sangiovanni-Vincentelli, *Logic Minimization Algorithms for VLSI Synthesis*. Hingham, MA: Kluwer Academic Publishers, 1984.

[5] R. K. Brayton, C. L. Chen, C. T. McMullen, R.H.J.M. Otten, and J. Y. Yamour, "Automated implementation of switching functions as dynamic CMOS circuits," in *Custom Integrated Circuits Conf. Proc.*, May 21–23, 1984.

[6] R. K. Brayton and C. T. McMullen, "Synthesis and optimization of multistage logic," in *Proc. IEEE Int. Conf. on Computer Design*, Oct. 1984, pp. 49–54.

[7] R. K. Brayton, private communication, Dec. 1985.

[8] W. W. Cohen, K. A. Bartlett, and A. J. DeGeus, "Impact of meta-rules in a rule based expert system for gate level optimization," in *Proc. IEEE Int. Symp. on Circuits and Systems*, June 1985, pp. 873–876.

[9] J. Darringer, D. Brand, J. Gerbi, W. Joyner, Jr., and L. Trevillyan, "LSS: A system for production logic synthesis," *IBM J. Res. Dev.*, pp. 537–545, Sept. 1984.

[10] A. J. DeGeus and W. Cohen, "A rule based system for optimizing combinational logic," *IEEE Design and Test of Computers*, pp. 22–32, Aug. 1985.

[11] G. DeMicheli and A. Sangiovanni-Vincentelli, "Multiple constrained folding of programmable logic arrays: Theory and applications," *IEEE Trans. Computer-Aided Design*, vol. CAD-2, pp. 151–167, July 1983.

[12] G. DeMicheli, M. Hoffman, A. R. Newton, and A. Sangiovanni-Vincentelli, "A design system for PLA-based digital circuits," in *Advances in Computer Engineering Design*, A. Sangiovanni-Vincentelli, Ed. Greenwich, CT: Jai Press, 1985.

[13] G. D. Hachtel, A. R. Newton, and A. L. Sangiovanni-Vincentelli, "An algorithm for optimal PLA folding," *IEEE Trans. Computer-Aided Design*, vol. 1, pp. 63–77, Apr. 1982.

[14] M. Hoffman and A. R. Newton, "A domino CMOS logic synthesis system," in *Proc. IEEE Int. Symp. on Circuits and Systems*, June 1985.

[15] R. Rudell, master's report, Dept. EECS, Univ. California, Berkeley, May 1986.

[16] T. Sasao, "An application of multiple-valued logic to a design of masterslice gate arrays," in *Proc. ISMVL-82*, May 1982.

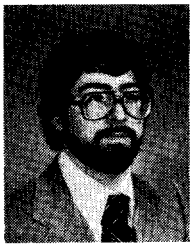
[17] K. Bartlett, R. K. Brayton, G. Hachtel, R. Jacoby, R. Rudell, A. Sangiovanni-Vincentelli, and A. Wang, "Multilevel logic and minimization," in *Proc. IEEE Int. Conf. on CAD*, Oct. 1985.

*

Karen Bartlett received the B.S. degree in computer science from Washington University in 1980 and the M.S. degree in electrical engineering from the University of Colorado, Boulder, in 1986.

From 1980 to 1983, she was employed by GTE Laboratories in Waltham, MA, where her projects included symbolic layout and VLSI database design and evaluation. Since June 1983, she has been active in the area of logic synthesis, first at General Electric's Microelectronic Center and then at the University of Colorado. She is currently employed by Seattle Silicon. Her research interests include logic synthesis, design optimization, and silicon compilation.

*



William Cohen received the B.S. degree in computer science from Duke University in 1984 and is currently pursuing the Ph.D. degree in computer science at Rutgers University.

He is employed at the Space Telescope Science Institute in Baltimore, MD. His research interests include logic synthesis and optimization and natural-language understanding.

*

Aart de Geus (M'80) received the M.S.E.E. degree from the Swiss Federal Institute of Technology in Lausanne, Switzerland, in 1979 and the Ph.D. degree in electrical engineering from Southern Methodist University in Dallas, TX, in 1985.

From 1979 to 1980, he worked on IC process simulation at the Institute of Microelectronics in Lausanne, Switzerland. He received a Swiss National Science Fundation scholarship in 1980 and 1981 for study abroad. In 1982, he joined the General Electric Microelectronics Center as a CAD Research Engineer and later as Manager of the Design Optimization Group. He received the Microelectronics Center Outstanding Technical Contribution Award in 1984 and 1985 for his work on delay calculation and timing verification. He is presently Manager of the Advanced Computer Aided Engineering Group at GE Calma in Research Triangle Park, NC. His research interests are in various aspects of computer-aided design with special emphasis on logic synthesis, design optimization, silicon compilation, and automatic design verification.

Dr. de Geus is a member of ACM and Tau Beta Phi.

Gary Hachtel (S'62–M'65–SM'74–F'80) received the B.S. degree from California Institute of Technology in 1959 and the Ph.D. degree from the University of California, Berkeley, in electrical engineering.

He has taught at U.C. Berkeley, at New York University, at U.C.L.A., where he was Regents Lecturer in 1974, and at the University of Denver. From 1965 to 1981, he was with IBM at the Thomas J. Watson Research Center at Yorktown Heights, NY, where he was Manager of Modeling and Systems Design in the Mathematical Sciences Department. Since 1981, he has been Professor of Electrical & Computer Engineering at the University of Colorado, Boulder. His current research is on algorithms for computer-aided design, simulation, testing, layout, sparse matrices, and optimization.

Dr. Hachtel was Associate Editor for the *International Journal for Numerical Methods in Engineering* and for the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS. He is now Associate Editor for the *International Journal for Mathematics and Computation in Electronics and Electrical Engineering*. He received an IBM Outstanding Contribution Award for integrated circuit modeling in 1968, and an IBM Outstanding Invention Award for the Tableau Approach to Network Design. In 1971, he was co-recipient of the Best Paper Award from the Circuits and Systems Society, and in 1979 he received the W.R.G. Baker Award for the best IEEE Proceedings or Transactions article to appear in calendar year 1978. In 1981, he was a Distinguished Lecturer of the Circuits and Systems Society.