AN ABSTRACT OF THE THESIS OF

Jorge F. Martinez-Carballido for the degree of Doctor of Philosophy in Electrical and Computer Engineering presented on November 12, 1982.

Title: PRONTO: A Product Term Reduction Approach

Abstract approved: Redacted for privacy

V. Michael Powers

Regular structures such as PLA's are very important to reduce VLSI design time. Interest in CAD tools such as a practical reducing PLA generator is high. This dissertation presents PRONTO as a practical, near-optimal product term reduction method, whose general heuristic approach consists of the following four steps. First, select a base product term. Second, find for the base product term a set of expandable directions; so that when the base is expanded, it can cover "most" of the uncovered product terms. Third, expand the base product term in those previously found directions to find a "best" expanded product term. Fourth, update the state of the uncovered product terms affected by the inclusion of the expanded base product term in the solution. These four steps are repeated until no product terms are left uncovered.

Results of eight examples (including one with 235 terms, 12 inputs and 25 outputs) show that PRONTO gives up to 21% better solutions (fewer product terms) than a previously published method. PRONTO has three characteristics which simplify the expected calculation. First, PRONTO only expands in those possible directions that cover "most" of the uncovered product terms. Second, PRONTO does not seek prime terms during expansion. Third, PRONTO can reach better solutions faster because its time to solution depends linearly on the number of product terms in the solution. With these three characteristics, a programmed implementation of PRONTO is expected to be faster than previously published product term reducers.

PRONTO: A Product Term Reduction Approach

by

Jorge Francisco Martinez-Carballido

A THESIS

submitted to

Oregon State University

in partial fulfillment of
the requirements for the

degree of

Doctor of Philosophy

Commencement June 1983

APPROVED:

Redacted for privacy

Associate Professor of Electrical and Computer Engineering

in charge of major

Redacted for privacy

Head of Department of Electrical and Computer Engineering

Redacted for privacy

Dean of Graduate School

Date thesis is presented November 12, 1982

Typed by Jorge F. Martinez-Carballido

ACKNOWLEDGEMENT

"The fear of the Lord is the
begining of wisdom;
a good understanding have
all those who practice it.
His praise endures for ever!"
Psalm 111:10

To my beloved wife Cristina and my children Georginna and Jorge Ricardo for their love in Jesus Christ our Lord.

To my parents Jorge and Maria de la Luz for all their effort and love they devoted to me.

To all those professors who unselfishly help me through all my education.

TABLE OF CONTENTS

LIST OF FIGURES

PRONTO: A Product Term Reduction Approach

1. INTRODUCTION

A. Overview

Very Large Scale Integrated technology has been largely restricted to general purpose rather than custom designs. Three recent developments have dramatically reduced costs and design times. First, methodologies such as the one developed by Mead and Conway [MEAD]. Second, computer-aided-design tools that allow implementation of such methodology. Third, a silicon foundry service that provides advanced wafer processing in large or small volumes and with good, previously unheard-of turnaround times [FAIRBAIRN]. Now it has become cost-effective to create high density circuits for specialized short-run applications. One of the techniques championed by people such as Mead and Conway [MEAD] helps the designer through the use of regular structures such as Programmable Logic Arrays (PLA's) and Read Only Memories (ROM's). The fewer individual transistors a designer must work with, the more efficient he/she can be.

The use of a PLA as a regular structure in VLSI design can significantly improve design time for digital logic chips. The chip's real estate is precious in VLSI design. Thus, a switching function reducer is attached to a PLA generator. The use of an optimal reducer would be rather expensive and impractical; thus, near-optimal reducers giving good, fast results are of great importance for a practical reducing PLA generator.

ROM is another useful regular structure often found in VLSI design. Thus, ROM reducers will significantly aid the area usage. Here, too, suboptimal solutions are generated in an attempt to keep the algorithm's complexity modest [MARTINEZ].

B. Objective

The objective of this research is to develop a practical, one-pass heuristically guided direct search algorithm for the reduction of product terms of a given large switching function specification.

The aim of this dissertation is to present results of research on PRONTO; a practical one-pass heuristically guided direct search method which can provide significant improvements in results as compared to a previous method.

Near-optimum reduction methods giving good, fast results are a significant part of reducing PLA generators, which are of great importance for VLSI design.

C. Impact

With the use of PRONTO we can expect fast, good results for practical problems, due to the following three characteristics. First, PRONTO tries to expand only truly expandable directions. Second, PRONTO seeks coverage of "most" uncovered product terms and does not require the expanded product terms to be prime. Third, PRONTO tends to be faster when it reaches better solutions because its complexity linearly depends on the number of product terms in a solution.

PRONTO is a practical near-optimal PLA reduction method which has taken the one-pass heuristically guided search approach. PRONTO consists of the following four steps. First, it selects a convenient uncovered product term from the specification. Second, it finds a set of truly expandable directions biased (guided) to cover "most" of the uncovered product terms. Third, it expands the selected product term to "best" expand it on the previously found directions. Fourth, it updates the state of any fully covered product terms and modifies any partially covered product terms. These four steps are the body of the main loop whose ending condition is that of no product terms are left uncovered. Thus, PRONTO directs its search towards those product terms that when expanded cover "most" of the uncovered product terms. PRONTO does not require that the expanded product terms be prime.

The following chapter presents a new nomenclature for product term reduction methods.

2.BASIC CONCEPTS AND DEFINITIONS

This chapter establishes sufficient mathematical background for a clear and precise understanding of the following chapters' contents. We present a nomenclature which defines the necessary concepts and a notation for representing instances and examples. The new nomenclature simplifies the explanation of reduction algorithms for multiple-output switching functions. For those readers without current familiarity with multiple-output switching function concepts, the new nomenclature and notation in this chapter promotes a full understanding of the concepts. The purpose of the new nomenclature and notation is to ease understanding of concepts. However, the new notation for switching literals and variables (see definitions below) has the drawback of becoming cumbersome to use in large practical problems. Once the concepts have been understood and the purpose of this part of the new notation has been achieved, we can simplify to the less cumbersome traditional switching algebra notation of '0' and '1' for switching literals and '-' or 'd' for switching variables, but still keep the rest of the notation and the nomenclature defined in this chapter.

Definition 2.1 A *switching literal* is a symbol representing one of the two implementation values (asserted low, asserted high) of the switching set P_2.

We will use primed (asserted low) and unprimed (asserted high) lower case letters for switching literals. Thus, $P_2=\{x',x\}=\{y',y\}=\ldots$

Definition 2.2 A <u>switching variable</u> is a symbol representing a variable whose value can be either of the switching literals.

We will use upper case letters for switching variables. Thus, $X = \{x',x\}$, $Y = \{y',y\}$, etc.

Definition 2.3 A <u>switching symbol</u> is a q-tuple of switching literals. For each of q switching variables, exactly one of the two switching literals appears.

Example : say q=4, then w'xy'z and wxyz are switching symbols, but ww'xy and wxx'z are not.

Definition 2.4 A <u>switching alphabet</u>, $P_2{}^q$, is the set of 2^q distinct switching symbols, of some q switching variables.

A switching alphabet is also known as a universe, and can be represented by a q-tuple of switching variables (i.e.,WXYZ).

Definition 2.5 A <u>subalphabet</u> is any of the 3^q elements of B^q=BX...BXB, where B is the set of three elements; a switching variable and both of its literals.

A switching alphabet or subalphabet is said to <u>contain</u> the corresponding switching symbols.

Example: For the switching alphabet XYZ, we have Xyz, x'YZ, xyZ as sample subalphabets; x'YZ contains {x'y'z', x'y'z, x'yz', x'yz}.

Definition 2.6 A switching function is a function from a switching alphabet of n switching variables, P_2^n, called the input alphabet, to a switching alphabet of m switching variables, P_2^m, called the output alphabet.

We will use boldface upper case letters to represent switching functions. Thus, **F**, **FDC**, etc.

Example: **F** is a switching function with AB as input alphabet and XY as output alphabet. Its specification is as follows:

F={(a'b',xy),(a'b,x'y),(ab',x'y),(ab,xy')}.

For convenience **F** is also represented as

F={a'b' xy,a'b x'y,ab' x'y, ab xy'}.

A symbol from the input (output) alphabet is called an **input (output) symbol.** Thus, a switching function is a mapping that assigns to each input symbol an output symbol.

From now on we will assume the obvious input/output format for function specifications in our examples.

Traditionally, a switching function is defined as one with an output symbol of one literal, and called a single-output switching function. For $m>1$ a system of single output switching functions is defined and called a multiple-output switching function.

Originally, a minimization algorithm was developed for single output switching functions and later extended for multiple output switching functions. The minimization algorithm for multiple output switching functions saw the problem as one of a system of single output switching functions. Thus, multiple output switching functions became naturally expressed as a system of single output switching functions. The representation used here becomes natural when defining the (multiple-output) switching function as a set of ordered pairs.

> Example:The switching function,**F**, from the above example, traditionally would be represented as the following two single output switching functions:
>
> $\mathbf{F}_1 = \Sigma\,(a'b', ab)$ and $\mathbf{F}_2 = \Sigma\,(a'b', a'b, ab')$, or for convenience decimal, octal or hexadecimal equivalents of the input symbols are used. Thus, $\mathbf{F}_1 = \Sigma(0,3)$ and $\mathbf{F}_2 = \Sigma(0,1,2)$.

Definition 2.7 A <u>minterm</u>(minimum term), MT, is a subfunction from an input symbol to an output symbol, where the latter has a single unprimed literal.

Example: in the specification of **F** in the previous example, a'b x'y, ab' x'y and ab xy' are minterms whereas a'b' xy is not.

Definition 2.8 An <u>input term</u> is a set of input symbols forming a subalphabet.

We will use n-tuples of switching variables and/or literals to represent input terms.

Example : with n=5,

vwxyz represents the 2^0 symbol subalphabet,
{vwxyz},

Vw'xyz represents the 2^1 symbol subalphabet,
{v'w'xyz,vw'xyz}, and

VWXyz represents the 2^3 symbol subalphabet,
{v'w'x'yz,v'w'xyz,...,vwx'yz,vwxyz}, all three examples are input terms.

Whenever corresponding inputs of two input terms have different literals (primed, unprimed), it is said to be a <u>disjoint input</u>.

Definition 2.9 A <u>product term</u> is a subfunction from an input term to an output symbol,where the latter has at least one unprimed literal.

Traditionally, product terms are represented only by their literals and whenever a variable is involved either it is replaced by a '-' or it is simply left out.

Example: vWxyz' abcd' is a product term with vWxyz' as input term and abcd' as output symbol.

Definition 2.10 A <u>simple product term</u> is a special product term with a single unprimed literal in the output symbol.

Example: The product term of the above example has the following three simple product terms: vWxyz' ab'c'd', vWxyz' a'bc'd' and vWxyz' a'b'cd'.

Definition 2.11 An <u>output term</u> is a set of output symbols forming a subalphabet.

We will use m-tuples of switching variables and/or literals for output terms. Thus, $Y_1Y_2 \ldots Y_m$, $Y_1Y'_2 \ldots Y_m$, etc.

For two output terms OT_1 and OT_2, it is said that OT_1 is a <u>non-covering output term</u> with respect to output term OT_2, if for any unprimed literal in OT_2 its corresponding output in OT_1 is a primed literal.

Definition 2.12 An <u>incompletely specified product term</u>, ISPT, is an ordered pair of an input and an output term, where the latter has at least two output symbols.

Definition 2.13 A <u>term</u> is either a product term or an incompletely specified product term.

Example: Here we give a sample specification of a four input, two output function with two ISPT's.

a'b'c'D	x'Y
a'Bcd	Xy'
Ab'cd'	x'y'
Abcd'	xy
aBCd	xy
aBc'd'	x'y'
a'bc'd'	xy'
a'bc'd	x'y.

When a term is split into two disjoint terms, each of these two is said to be a *half term*.

A switching function can be specified by its set of minterms or by an equivalent set of product terms. For practical purposes, a specification which lists all the minterms is prohibitively long. In practice, a designer naturally specifies a problem using product terms and/or incompletely specified product terms. We must bear in mind that the presence of output variables in the ISPT's of the function specification gives us freedom of choice among several realizable switching functions. That is, an ISPT violates the switching function definition in that the function is a many-to-one symbol relation and an ISPT is a (many-input)-to-(many-output) symbol relation. The presence of an output variable in a ISPT is known as a "don't care" condition and either a '-' or a 'd' is commonly used in place of the switching variable in the output term. However, the use of upper case letters for switching variables is an attempt to more clearly represent the concept involved, than the use of a '-' or a 'd' or simply the switching variable absence (commonly used in the input term instead of the switching variable).

The presence of ISPTs in a specification leads us to a set of switching functions that can be implemented. The following definitions will help to clarify the range of realizable functions for a particular specification.

Definition 2.14 A minimum (maximum) function , F^{min} (F^{max}), is formed by converting each output term to an output symbol, where each switching variable in the output term is assigned its primed (unprimed) literal.

In other words the minimum (maximum) function is the function with a minimum (maximum) number of minterms among the set of possible functions from a given specification.

Definition 2.15 A solution, F^i, is a switching function with at least the minterms in F^{min}, and at most the minterms in F^{max}.

$$F^{min} \subseteq F^i \subseteq F^{max}$$

A set, G, of product terms is said to **cover** a set, H, of product terms iff each minterm in H, is also in G (i.e., $H \subseteq G$).

Example: for the above specification the minimum function is:

a'b'c'D	x'y'
a'Bcd	x'y'
Ab'cd'	x'y'
Abcd'	xy
aBCd	xy
aBc'd'	x'y'
a'bc'd'	xy'
a'bc'd	x'y.

The maximum function is:

a'b'c'D	x'y
a'Bcd	xy'
Ab'cd'	x'y'
Abcd'	xy
aBCd	xy
aBc'd'	x'y'
a'bc'd'	xy'
a'bc'd	x'y.

Definition 2.16 An <u>expansion procedure</u>, is one that takes a given product term, PT, from a solution, $\mathbf{F}^k$, and finds a product term, PT^*, such that PT^* covers PT, and yet PT^* is covered by the maximum function, $\mathbf{F}^{max}$.

$$PT \subseteq \mathbf{F}^k \ , \ PT^* \subseteq \mathbf{F}^{max} \text{ and } PT \subseteq PT^*$$

Definition 2.17 A prime term or prime implicant, PI, is a product term, such that PI has at least one minterm from F^{min}, and no larger product term, PT, exists, such that PT covers PI, and yet PT is covered by the maximum function, F^{max}.

Definition 2.18 A base term is a term selected as the starting point for an expansion procedure to find an (expanded) product term which will cover it.

Definition 2.19 The distance between two terms is given by the sum of the number of inputs that differ in the terms and one if any corrresponding outputs of their output terms have different literals (conflict).

Definition 2.20 Two terms are adjacent if they are distance one apart.

Example:

T_1 = Vwx'y Abc'

T_2 = Vwxy abc'

T_3 = vw'x'y abc'

T_4 = Vwx'y' Ab'c'

Input terms Vwx'y and Vwxy differ in one input but their corresponding output terms Abc' and abc' have no conflicting literals so the distance $d(T_1,T_2) = 1$. $d(T_1,T_3) = 2$ and $d(T_1,T_4) = 1+1+2$. So both T_1 and T_2 merge to VwXy abc'.

It is our belief that the use of switching variables in input and output terms, does clearly represent the concepts involved. Thus, we hope the novice reader will easily understand the concepts presented in this chapter.

Our switching literal concept is equivalent to W. Fletcher's assertion levels [FLETCHER]. Some of our other concepts are similar to the ones used by S.J. Hong, *et. al.* [HONG]. Logic operators are of no particular interest in this paper. Therefore, we assume a function specification is in a form equivalent to the one presented here.

The next chapter will describe the PLA area reduction problem.

3. REDUCING THE PLA AREA

One particularly important computer-aided-design tool is a PLA generator. The use of PLA's in VLSI design can dramatically reduce the design time for digital logic chips [RIHERD]. However, chip area is at a premium in VLSI design. Thus, a switching function reduction method is implemented as part of a PLA generator.

The PLA area is basically computed as follows: $A= (2*N_I+N_O)*N_P+r*N_I+s*N_O+t*N_P$, where N_I,N_O and N_P are the number of inputs, outputs and product terms respectively, and r, s, t are overhead factors. The reduction in any of these dimensions reduces the PLA area. Among the above dimensions, the number of product terms gives a larger reduction in the PLA area.

A. PLA Area Reduction Approaches

Often, the original PLA specification results in redundant (inefficient) real estate usage; thus, any real estate redundancy reduction techniques might prove to be useful. The real estate redundancy reduction techniques can be divided into the following: input or output reduction, folding, simple or multiple level partitioning, external logic at the inputs and/or outputs, and product term reduction.

Input (output) reduction techniques aim to reduce the number of redundant inputs (outputs). Martinez and Powers give techniques which can be used to find a reduced set of inputs and/or outputs in an effective way [MARTINEZ]. Kambayashi stressed the importance of reducing the number of PLA inputs [KAMBAYASHI]. His approach assumes that for each distinct output pattern there are several input terms; thus, we need only to distinguish among input terms with distinct outputs.

Folding is a technique which finds a peculiar way of sharing space for columns and/or rows, thus reducing the number of physical columns and/or rows needed to implement the original PLA [PAILLOTIN, HACHTEL]. Seventy-five percent of area reduction is the upper limit for the simple folding technique [HACHTEL].

Partitioning is a technique that breakes a PLA into several smaller ones. This could be in single level parallel [BRAYTON,GRASS,KANG] or multilevel serial [MARTINEZ] partitioning, or any combination thereof.

The addition of external logic is aimed to reduce the number of product terms and yield an overall real estate reduction. One has to consider the extra delay introduced by the external logic, which might in some cases be unacceptable. The use of PLA's for external logic is similar to serial partitioning. However, the possible reduction in area due to partitioning comes from breaking the implementation into several smaller ones, whereas the possible reduction in area due to using external logic comes from the reduction in the number of product terms. J.P. Roth and T. Sasao describe their own techniques to find simple external logic at the inputs [ROTH,SASAO]. No formal techniques for external logic at the outputs has been published. However, one can see that simple logic, like inverters, at the outputs might reduce the number of product terms, or that some redefinition of the outputs will yield a reduced number of product terms; thus, to restore the original outputs some external decoding logic is needed.

An examination of the area formula shows that the most important factor for reduction is N_p (number of product terms). In this dissertation, we consider the PLA product term reduction problem as the most important step to PLA area reduction. Methods for PLA product term reduction is the subject of the next section.

B. PLA Product Term Reduction Methods

Optimum PLA product term reduction is a simplification to the classical sum-of-products minimization of (multiple-output) switching functions. The cost function for the PLA product term reduction is simplified to be the number of product terms; whereas, the cost function for the classical minimization combines the products with their size (number of literals). This change in the cost function allows us to concentrate on the number of product terms without regard to their shapes. There are optimum and near-optimum switching function reduction methods.

Optimum product term reduction methods are based on classical methods from switching theory of: first, generating all prime implicants, and second, finding a minimum cover. Research on optimum reduction methods over three decades has found this to be one of the Non Polynomial (NP) complete class [GIMPEL,GAREY]. This is to say, so far no polynomial time algorithm (its execution time in a deterministic Turing machine can be expressed as a polynomial in the number of inputs and outputs) has been found for finding the optimum, and it is thought very unlikely that one will ever be found. Furthermore, the known algorithm for an optimal solution is equivalent to a large class of similarly difficult problems in the sense that, if a polynomial time algorithm can be found for one of the class; by transformation the entire class can be solved in polynomial time. However, no proof exists stating that no algorithm with polynomial time is possible. It is very unlikely that an optimum reduction method for a PLA generator will be justified due to the prohibitively large requirements in memory and computation time.

Near-optimum product term reduction methods give good, fast solutions while seeking but not guaranteeing an optimal solution. The search for an optimal solution is done with the help of heuristic methods. The heuristic methods aim to use affordable amounts of memory and computational time. Near-optimum reduction methods can be divided into three classes. First, some reduction methods use an internal minterm level description [AREVALO,BRICAUD]. The use of a minterm-level description is limited to small problems due to the prohibitively large exponential increase in memory and computation requirements for practical (large) problems. Second, some other reduction methods use a compact description and the final solution is reached through iterative improvement rather than the classical two step approach of completely generating prime implicants, and then covering the functions. Hong, et. al. in their MINI [HONG] give generally better solutions than Svoboda's PRESTO. However, MINI requires significantly larger amounts of memory and computation than even Brown's modified PRESTO [BROWN]. Third, there are reduction methods using a compact description which heuristically approach a solution throughout a one-pass guided search rather than improving the solution through successive iterations. R.K. Brayton, et. al., in their ESPRESSO [BRAYTON] describe the procedure MINIMAL as a one-pass heuristically guided search approach to product term reduction.

One-pass heuristically guided search procedures can be further divided into direct and indirect. One-pass heuristically guided indirect search procedures are those which generate a prime term for each product term and then find an irredundant solution. One-pass heuristically guided direct search procedures are those that sequentially find the actual product terms of a solution.

The next chapter describes some recent approaches to product term reduction.

4. RECENT APPROACHES

For some recent approaches to near-optimum product term reduction, this chapter describes the expected success of obtaining solutions. Afterwards, three recent approaches to product term reduction are described followed by a proposed method to solve an order dependence problem in some of the recent approaches.

A. Limitations of Near-optimal Approaches

As mentioned in Section 3.B, for practical purposes a near-optimum product term reduction method must accept as input a compact specification of the problem; otherwise, the complexity would be unacceptable. In what follows, we use the nomenclature developed in Chapter 2.

The search for a solution among the set of all solutions, PS, of a switching function could be attempted as a two step process. First, we could find all the prime implicants of the function. Second, we could find a minimal cover of prime implicants.

Say for the sake of understanding, that we were to find a solution in two steps. First, for every product term in the specification generate, through expansion, all prime implicants covering it. Second, find a minimum cover. The first step will only generate a subset of all prime implicants of the function because the product terms may not all be minterms. Thus, we are limited to a subset of reachable solutions, RS1, of all possible solutions ($RS1 \subseteq PS$). The generation of all prime implicants for every product term in the specification is in general impractical. Thus, a heuristic.expansion selects a "best" prime implicant for each product term. This further limits us to a subset RS2 of RS1. Now, we are limited to a solution in RS2; that is, we are bound to find at best an optimum in RS2, which might or might not be an optimum in PS. This is to say there are specifications from which a global optimum is not reachable, when a two step method consisting of a heuristic expansion and a minimum cover is used. The use of a minimum cover makes the latter method computationally intensive and the heuristic expansion step can severely limit the likelihood of finding an optimal solution.

In an attempt to diminish the above mentioned problems, the minimum cover step can be replaced with either one of two approaches. If we are content with a reduced irredundant cover, we replace the minimum cover step with an irredundant cover step. In the event that we wish to seek minimization, we replace it with a reduction step. This reduction step is one of eliminating redundant product terms and trimming. Trimming reduces a non redundant term to a smaller term so that the cover remains proper. This allows for further expansion through another iteration. This replacement has changed our two step method to an iterative one. Trimming allows for a selection of different subsets of reachable solutions. Thus, trimming allows a more extensive set of reachable solutions, which in turn increases the probability of finding a better solution. Furthermore, we can improve the method by adding a reshaping step after reduction. The reshaping step allows further expansion through the selection of a different subset of reachable solutions.

The procedures MINI, PRESTO and ESPRESSO are described in the following sections.

B. MINI

The iterative improvement procedure, MINI, was developed by S.J. Hong, et. al. for minimizing "shallow switching functions",or those functions whose minimal solutions contain at most a few hundred product terms regardless of the number of variables [HONG].

MINI accepts as input a switching function expressed as a list of product terms and incompletely specified product terms, thus avoiding a prohibitively long minterm specification. MINI starts by making the minimum function (don't cares set to their primed literals) of the specification one of mutually disjoint product terms. A disjoint specification results in a larger set of smaller product terms. By changing the initial specification to a disjoint one, MINI increases the number of reachable solutions. This in turn makes it more likely to find an optimal solution among the reachable solutions. The disjoint specification introduces further merging freedom through its larger set (but not prohibitively numerous such as a minterm list) of smaller terms.

MINI takes the disjoint specification as the initial solution to the following algorithm:

```
Expand_solution;
Compute_solution_size;
REPEAT
     Reduce_solution;
     Reshape_solution;
     Expand_solution;
     Previous_size:= solution_size;
     Compute_solution_size
UNTIL solution_size not < previous_size;
```

MINI's expansion process starts ordering the terms using a simple heuristic algorithm, which tends to put on the top of the list those terms that are hard to merge with others. Once the terms are ordered, expansion proceeds from top to bottom, each time finding a prime term covering the term under expansion and perhaps many of the other terms in the solution. This is accomplished by the use of heuristics biased towards a prime term that "looks" the best (one that covers more terms of the solution). Any covered terms are deleted and the new term is appended to the bottom of the solution. The expansion process terminates when no more unexpanded terms are left.

MINI's reduction process starts reordering the terms of the solution using a heuristic algorithm which tends to put on top of the list those terms that are large, thus hard to reduce. The reduction process proceeds down the list. For each term, T, it takes those terms in T and not in any other term of the current solution to form the set of unique terms, SUT, and finds the product term, ST, covering SUT. No product term, PT^*, exists such that ST covers PT^*, and PT^* is covered by F^{max}. Once ST is found by the reduction process, ST replaces T, because ST covers all the terms that are unique to T. In the event that the set of unique terms, SUT, is empty, T is simply removed from the list. Thus, the solution size is reduced by one because the redundant term T is removed.

Paraphrasing S.J. Hong, et. al., we say that, after an expansion and reduction step the nature of the product terms in the solution is that there is no product term covered by F^{max} which covers more than one product term in the solution, [HONG].

MINI's reshaping process reorders the terms in the solution with more "splittable" terms at the top of the list. Reshaping then finds each pair of terms that can be rewritten as another pair of disjoint terms with the same covering. This increases the probability of further merging through another expansion step. Actually, the reshaping process allows the possibility to search through otherwise unreachable solutions because of specification dependencies (as explained in Section 4.A).

C. PRESTO

The iterative improvement procedure, PRESTO, was reported by Brown as developed by A. Svoboda, who chose a minimal set of operations which were fast and required little computer memory [BROWN].

In what follows, we will consider **F** to be a switching function of 'n' input variables, $(X_1,\ldots,X_n)$, and 'm' output variables $(Y_1,\ldots,Y_m)$.

PRESTO accepts as input a term specification of the switching function, **F**, thus avoiding any prohibitively long specifications such as a minterm list.

PRESTO starts by forming the two extreme functions, $\mathbf{F}^{min}$ and $\mathbf{F}^{max}$, from its original specification. PRESTO works iteratively on $\mathbf{F}^{min}$ by adding minterms from $\mathbf{F}^{max}$ that reduce the number of terms in the solution. Thus on each iteration, i=1,2,..., we have an $\mathbf{F}^i$ such that $\mathbf{F}^{min} \subseteq \mathbf{F}^i \subseteq \mathbf{F}^{max}$, and the number of product terms in F^i is less than or equal to the number of product terms in $\mathbf{F}^{min}$. PRESTO iteratively executes the expansion and reduction processes until no change in the size of the solution happens.

The expansion process transforms each product term, PT, in $\mathbf{F}^i$ to a prime term, *PI*, such that $\mathbf{F}^{max} \supseteq PI \supseteq PT$. For a given product term, PT, the expansion process finds one of its PI's as follows. First, it selects the input literals in turn , x_j or x_j' given by the arbitrary order (left to right). Second, it tries to expand PT by changing x_j or x_j' to its switching variable X_j and checks if this newly formed product term, TPT, is covered by $\mathbf{F}^{max}$. Third, if TPT is covered by $\mathbf{F}^{max}$ then PT is replaced by TPT; otherwise, no change is made. The above three steps are repeated until no more input literals are left to be considered for expansion.

The reduction process in PRESTO is its only means to reduce the number of terms in a solution. This process tests for elimination of each product term, PT, in the solution. The reduction process selects one simple product term, SPT, from the product term PT. Next, the reduction process tests if SPT is covered by the remainder of the solution $\mathbf{F}^i$. If SPT is covered then SPT is eliminated by replacing its unprimed with its primed output literal in PT, the corresponding unprimed output literal of SPT; otherwise, no change is made. Next, the reduction process selects another SPT and repeats until all SPTs from PT have been considered. If at this point the output symbol of PT has only primed literals, then the reduction process eliminates PT reducing the solution size by one.

For the reduction process, one can easily see that for each product term the processing order of its simple-product-terms is immaterial. However, the number of product terms in the solution depends on the order the product terms are tested for elimination.

To see the product term order dependence, consider the case when a product term, PT_k, after going through the reduction process, is transformed to PTT and not deleted. Later on, the reduction process finds a product term, PT_j, covered by the original PT_k, but because PT_k was transformed to PTT the reduction process is unable to eliminate PT_j. In this case, no product term is eliminated, whereas if the reduction process transforms PT_j before PT_k, PT_j is eliminated.

D. Brief Comparison

We have said in the above sections that the results of both MINI and PRESTO depend on the function specification. MINI tries to lessen this difficulty by making a larger set of solutions reachable because it uses a specification of mutually disjoint product terms. Thus, a more extensive search for a solution can be done by making the specification closer to one of minterms "but not prohibitively numerous such as a minterm list"

[HONG]. PRESTO does nothing to change the original specification.

By looking to the processes in PRESTO, one can expect that the computer memory required to implement PRESTO be significantly smaller than MINI's.

MINI uses ordering heuristics for each expansion, reduction, and reshaping step. Thus, a significant amount of reordering is done. PRESTO, on the other hand, uses no ordering.

Both MINI and PRESTO provide trimming in the reduction process to allow further merging through another expansion.

MINI has a reshaping step which makes other solutions reachable, while PRESTO does not.

PRESTO has a procedure to check if a product term is part of a function. This procedure does minterm level checking which is computationally expensive. However, D. Brown modified it to a tree method making PRESTO more efficient [BROWN]. This cover checking procedure does not consider don't cares, so solutions with redundant terms due to don't cares may result. On the other hand, MINI checks for don't cares.

All steps in MINI and PRESTO have some kind of order dependence.

E. ESPRESSO

A divide and conquer (partitioning) approach was taken by R.K. Brayton, et. al. in ESPRESSO [BRAYTON] to significantly reduce the computational complexity of product term reduction. In this, they shared an intellectual outlook with S. Kang in SPAM [KANG]. The use of partitioned complementation in ESPRESSO helped significantly to reduce the computational complexity. The divide and conquer approach was achieved by tree-partitioning the function. In this way, they deal with several partial covers of the function instead of a single cover. After finding a set of "best" disjoint partial covers, they find a minimal cover for each disjoint partial cover. These are merged to form a single cover. ESPRESSO can be summarized as a three-step procedure.

```
Find a "best" partition tree for the function;
Find a minimal cover for each of the leaves;
Merge the reduced leaves;
```

MINIMAL as named in ESPRESSO is the basic procedure applied in the above second step. This is the part most closely related with PRONTO as a product term reduction procedure. MINIMAL can be seen as taking a function specification as input to the following procedure.

```
Do single product term containment;
Do "distance-one" merging;
Do ordering product terms by descending size;
Solution:= empty;
For each product term in order Do
     Begin
       Find a "best" prime term;
       Solution:= Solution + prime term;
     End;
Make Solution Irredundant;
```

In order to find a "best" prime term, MINIMAL heuristically selects an order of variables with which to expand the product term. MINIMAL provides the user with three possible ways to do cover checking. These are intersect complete or partial complement and a tree algorithm. In the reported experiments with ESPRESSO, the partial complement cover checking option showed to be best. In order to make the solution irredundant, MINIMAL has a procedure named IRRCOVER as its last step. This procedure deletes any prime implicant found to be covered by the remainder of the solution. The result of IRRCOVER can be seen to depend on the order in which the prime implicants are tested for covering. MINIMAL can be summarized as a procedure which does some preprocessing to the function, then for each product term finds a "best" prime implicant and gives as solution an irredundant set of prime implicants. MINIMAL is thus a one-pass heuristically guided indirect search approach to product term reduction.

F. Order Independence in Reduction and Irredundant Covering Procedures

This section presents a proposed approach to solve the relative order dependence of product terms for the reduction step in PRESTO and for the irredundant cover (IRRCOVER) step in MINIMAL.

Let us examine why the reduction and irredundant covering procedures are dependent on the order in which each product term is tested for redunction or irredundancy. Each time a product term is found reduceable or irredundant, it has the effect of changing the enviroment (set of product terms) for the next product term; therefore, there is a potentially negative effect on the next product term when testing for reduction or irredundancy.

An approach to guarantee order independence can be done in a two-step solution. First, guarantee the same enviroment for each product term regardless of its relative position. This allows us to identify all possible (regardless of its position) redundant product terms. Second, guarantee an order independent selection of those product terms actually redundant. The first step can be achieved by changing a redundant product term to a don't care. In this way, we guarantee the same enviroment and identify those possibly redundant product terms in a solution. The second step can be done as follows:

a). Identify the set of uncovered product terms when all the possibly redundant product terms are deleted from the solution.

b). If the set of uncovered product terms is nonempty, then find an order independent reduced cover of this set by using the possibly redundant product terms; otherwise, all possibly redundant product terms are actually deleted from the solution.

An order independent reduced cover can be found with a heuristic criteria. This finds a cover of the uncovered product terms with an order independent selection of the possibly redundant product terms. Order independence is ensured by making each criterion one which is based only upon order independent properties.

The above approach can be summarized as one that delays the decision to delete a redundant product term until all product terms have been considered. Then a reduced set of the possibly redundant product terms is selected to keep a proper function cover.

The next chapter presents PRONTO as a one-pass heuristically guided direct search approach to product term reduction.

5. A ONE-PASS HEURISTICALLY GUIDED DIRECT SEARCH METHOD

This chapter presents PRONTO as a practical approach to product term reduction. This approach is a one-pass, heuristically guided direct search procedure with the following three characteristics. First, it only attempts to expand product terms in directions which can possibly cover some other uncovered product terms. Second, it does not seek primality (prime terms) in expansion; however, PRONTO seeks to cover "most" uncovered product terms. Third, its complexity is linearly dependent on the number of product terms in the solution.

A. PRONTO

As it has been previously mentioned, PRONTO is a one-pass, heuristically guided direct search approach procedure that sequentially chooses the elements of a solution. The search for the elements in a solution is done with the help of some heuristic criteria. For the selection of each of the elements in the solution, simplicity and effectiveness of the procedures were kept in mind. With this approach in mind, a four-step selection criteria for the product terms in the solution is followed. These four steps are as follows: First, select an uncovered product term (one of those yet needed to attain a solution) to serve as the base product term for expansion. Second, find a set of uncovered product terms that can possibly merge, in full or in part, with the base product term. Third, among the possible expansions of the base product term, select one that can cover "most" of the uncovered product terms. Fourth, update the state of each product term after the base product term has gone through expansion. PRONTO has the following procedure:

```
REPEAT
  Step 1:   Select a Base Product Term;
  Step 2:   Select an "easiest" to Merge Set;
  Step 3:   Expand the Base Product Term;
  Step 4:   Update the State of Product Terms;
UNTIL       No Uncovered Product Term is Left;
```

A description of each of the above four steps, with selection criteria follow:

B. Selecting a Base Product Term

The aim of the Step 1 procedure is to find among the uncovered product terms, one that is highly unlikely to be covered by expansions of other uncovered product terms. To achieve this, we present a selection criteria which assumes we have computed; first, for each output column j of the specification, the number of unprimed output literals u_j and the number of switching variables v_j and second, for each row i the number of unprimed output literals o_i. The criteria first select an output column and then among the uncovered product terms having an unprimed literal on the selected output column, the selection criteria choose a single uncovered product term. A description of the column selection criteria follows:

Criterion CS1. Of the output columns not yet selected, we choose a column J with a minimum number U_J of unprimed literals but greater than zero.

$$U_J = \min_j u_j : u_j > 0$$

Criterion CS2. Among the candidates from CS1, we choose a column with the minimum number V_K of switching variables in it.

$$V_K = \min_J v_J$$

Criterion CS3. Of the candidates from CS2, we select one arbitrarily.

Once a column K has been selected, choose among the uncovered product terms having an unprimed literal in column K, the first uncovered product term with a minimum number O_I (greater than zero) of unprimed output literals. No new column is selected until all the product terms with unprimed literals in the present column are covered. The next section deals with the selection of a set of uncovered product terms that possibly can fully or partially merge with the base product term.

C. Selecting an "easiest" to Merge Set

The goal for Step 2 is to provide a set of uncovered product terms that can "easily" merge with the base product term and after expansion give a "best" coverage of uncovered product terms. In other words, we are selecting a set of possible directions to expand with a high possibility to cover "most" of the mergeable uncovered product terms. The approach to achieve the goal in this section is through first, classifying each uncovered product term with respect to the base product term and second, selecting an "easiest" to merge set of uncovered product terms. The uncovered product terms can be grouped into those that intersect (share some term) the base product term and those which do not intersect (share no term) the base product term. Each uncovered product term of the intersecting group falls into either a covered, covering or partially covered class, and each uncovered product term of the non intersecting group falls into either a mergeable directly or mergeable indirectly class. These five classes are described as follow:

The uncovered product terms in the covered class are those terms covered by the base product term.

Those uncovered product terms that properly cover the base product term are in the covering class.

The partially covered class of uncovered product terms are those uncovered product terms that intersect the base product term and those that are neither properly covered nor properly covering.

The mergeable directly class of uncovered product terms are those non-intersecting uncovered product terms having a single disjoint input or a non-covering output term with respect to their corresponding output term or input in the base product term.

The mergeable indirectly class of uncovered product terms are those non-intersecting uncovered product terms having more than one disjoint input, or a non-covering output term and a disjoint input with respect to their corresponding output term or input in the base product term.

Let us consider each of the classes and examine them for their usefulness in finding an "easiest" to merge set of uncovered product terms.

For the covered class of uncovered product terms, its members are completely covered and obviously redundant; thus, they should be left out from further consideration (changed from uncovered to a covered state).

A member of the covering class is an uncovered product term which includes the base product term. Thus, if this class is nonempty, the state of the current base product term is changed to the covered state and the procedure starts over by selecting a new base product term.

For the partially covered class of uncovered product terms, its members are partially covered by a part of the base product term. Among the members of this class, select those which can split in two product terms where one is covered by the base product term and the other is mergeable directly. Thus, any uncovered product term meeting the above conditions can be replaced by its mergeable directly half. Any other partial covering is considered as too expensive to carry on.

The members of the mergeable directly class are uncovered product terms that can be either fully or partially merged with either the full base product term or part of it. Thus, there are four possibilities. First, the full uncovered product term can be merged with the full base product term. This kind of uncovered product term certainly is the easiest to merge with the base (adjacent to the base); thus, these uncovered product terms are made members of the "easiest" to merge set. Second, the full uncovered product term can be merged with part of the base product term (properly covered by a product term adjacent to the base). The question here is, what part of the base? a half?, a fourth?...; PRONTO limits the selection for the "easiest" to merge set to those that can be merged with a half of the base product term. This can save a great deal of computational complexity (fewer calls to the cover checking procedure).

Even with these limitations, the results show no reduction in the quality of the solution as presented in Section 6.B. Third, part of the uncovered product term can be merged with the full base product term; again, limiting the part to be a half can save complexity; thus, the uncovered product terms of which a half can be merged with the full base product term belong to the "easiest" to merge set. Fourth, part of the uncovered product term can be merged with part of the base product term; for the same kind of reasons as above, PRONTO limits the parts to both being a half and adds them to the "easiest" to merge set.

A member in the indirectly mergeable class is one which can be merged only if two or more other terms, each adjacent to the base product term, can also be merged. Thus, no member of this class is in the "easiest" to merge set.

The above selection of the <u>easiest to merge set</u> is summarized as follows. The members of the "easiest" to merge set are those uncovered product terms in the mergeable directly class which are adjacent to the base product term or a half of a product term adjacent to the base product term, and those uncovered product terms with a half which is either equal to or a half of a product term adjacent to the base product term.

Example:

This example shows how an easiest to merge set is found. Let us consider the term T1 from Figure 5.1 as the base product term for the specification T1,T2,...,T8. In order to find the easiest to merge set, we first classify each uncovered product term with respect to the base as follows: T2 as mergeable directly because it has a single disjoint input with the base; T3 as partially covered because it shares the term 0101- 010 with the base but half of T3 (0101- 001) is mergeable directly; T4 as mergeable directly because it has a single disjoint input; T6 as mergeable indirectly because it has more than one disjoint input; T7 and T8 as mergeable indirectly because they have a single disjoint input and a non-covering output term. Second, the easiest to merge set is selected from the mergeable directly class. The mergeable directly class is formed by T2, T4, and

T1	0101-	110
T2	11011	110
T3	0101-	011
T4	0111-	110
T5	11010	--0
T6	1111-	111
T7	11-1-	001
T8	0111-	001

Figure 5.1 Selecting an easiest to merge set

0101- 001. All three are easiest to merge because T2 is half of a product term adjacent to the base, and both T4 and 0101- 001 are product terms adjacent to the base.

The next section describes the base product term expansion step of PRONTO.

D. Expanding a Base Product Term

The aim in Step 3 is to provide for the current base product term an expanded base product term that can cover "most" of the uncovered product terms in the "easiest" to merge set. Here, a two-part approach is taken. The first part finds those members of the "easiest" to merge set having a half covered by a product term adjacent to the base product term and each of their covered halves can be expanded to be a product term adjacent to the base product term. At this point, a set of product terms adjacent to the base product term is available. These are those members of the "easiest" to merge set which are adjacent to the base product term and those found by expansion to be adjacent.

The second part of Step 3 expands the base product term by considering only those directions where adjacent product terms were found. Now, in order to find an expanded version of the base product term, we use a tree expansion algorithm like the one used by V.T. Rhyne, et. al. [RHYNE]. As for the cover checking procedure any of the ones in MINIMAL can be used. However, due to the restrictions imposed on selecting the adjacent product terms and the nature of practical problems, we do not expect on the average to have a "large" number of product terms adjacent to the base product term. Thus, the above imposed restrictions limit the number of branches in the expansion tree, which in turn will cause a reduction of the usage of time-consuming cover checking procedures.

For the example in Section 5.C we found the easiest to merge set to be T2, T4 and 0101- 001. Now, let us expand the base product term (T1). First, we check if T2 can be expanded to be a product term adjacent to the base and indeed it can be expanded to 1101- 110 so, we have a total of three product terms adjacent to the base. Next, we expand and find through a tree expansion that T1 can be expanded in all three adjacent directions and yield the term -1-1- 111 as the expanded base product term.

One could fairly easy see that because we are limiting expansion to a selected set of directions, we are bound to end up with product terms which are not prime. Not seeking prime terms (primality) not only can save complexity but also can improve results as our experimental results in Section 6.B show.

The resulting expanded base product term has been guided to cover "most" uncovered product terms. Thus, we only have to record the change in state of those product terms affected by the expansion of the base product term. This is the subject of the next section.

E. Updating the State of the Product Terms

As the title of this section reflects, here the purpose is to update the state of the product terms which are covered fully or in half by the expanded base product term. It is quite simple to find out what uncovered product terms have been covered completely or in half, if when expanding the base product term a record of this data is kept; thus, this data is readily available after the previous step has been completed. Given this data, any product term completely covered by the expanded product term is changed to be in its covered state and any product term covered in half is replaced by its uncovered half.

At this point, all four steps of PRONTO have been described. Next, an example to help clarify how PRONTO works is presented.

F. Example

We have chosen to present as example a PLA description given by S. Kang, [KANG]. This particular example is well suited for the partitioning developments done by S. Kang in SPAM. The PLA description is given in Figure 5.2 (a) and the solution (developed below) is in Figure 5.2 (b).

Let us follow the steps of PRONTO for this example. In Step 1, CS1 finds the third (from left to right) output column as its only candidate. Thus, among the three uncovered product terms (PT_3, PT_5 and PT_6) having unprimed output literals in Column 3, PT_3 is selected because is the first uncovered product term found with minimum number (2) of unprimed output literals. Therefore, Step 1 selects PT_3 as the first base product term. Step 2 looks for an "easiest" to merge set from the directly mergeable class for PT_3. Uncovered product terms 1,4,5,6 are mergeable indirectly because each is found to be disjoint with the base in at least two inputs; uncovered product terms 2 and 7 have one disjoint input and non-covering output terms, thus they also are mergeable indirectly. For product terms 8 through 18 all the output terms are found to be non-covering and they have no disjoint inputs with respect to the base; thus, they are mergeable directly, but none of them is adjacent nor can it be expressed as two product terms where one of them is covered by a product term adjacent to the base product term, so the "easiest" to merge set is empty. Thus, Step 3 does not expand PT_3 and the first product term, S_1, in the solution is PT_3; Step 4 changes its state to the covered one (marking it, in the right-hand column of Figure 5.2 (*a*), as the base for the first product term selected for the solution, B1). For the following three selections, PRONTO goes through

PT			
1	001000----------	0001000	B9
2	001001----------	0001000	C9
3	001101----------	0011000	B1
4	010001----------	0001000	B10
5	010110----------	0011000	B2
6	011001----------	0011000	B3
7	001111----------	0001000	B11
8	------1011001000	0101000	B5
9	------0011001000	0101000	C5
10	------0101001000	0101000	B6
11	------0111001000	0101000	C6
12	------0001001000	0101000	C5
13	------1101001000	0101000	B7
14	------1001001000	0101000	C5
15	------0001101000	0101000	B8
16	-------111110000	0001000	B12
17	-------001110000	0001000	B13
18	------0110101000	0100000	B4

a). Original specification

S_1= PT_3, S_2 = PT_5,S_3 = PT_6, S_4 = PT_{18}, S_7 = PT_{13},
S_8= PT_{15}, S_{10}= PT_4, S_{11}= PT_7, S_{12}= PT_{16}, S_{13}= PT_{17},

S_5= -------0-1001000 0101000

S_6= ------01-1001000 0101000

S_9= 00100----------- 0001000

b). Solution

Figure 5.2 PRONTO's example

similar choices and finds $S_2=PT_5$, $S_3=PT_6$ and $S_4=PT_{18}$. Next, Step 1 finds PT_8 as the fifth base product term; Step 2 finds that PT_9 and PT_{14} are the "easiest" to merge set; Step 3 finds an expansion that covers PT_8, PT_9, PT_{12} and PT_{14}, thus S_5 is equal to the expanded base product term covering the above four product terms; Step 4 changes their states to the covered one (a B5 for PT_8 and a C5 for the PTs covered by the expansion from B5 to S_5 marking it in the right-hand column of Figure 5.2 (a)). As the base for the sixth product term in the solution, PT_{10} is chosen; Step 2 finds PT_{11} and PT_{13}, but Step 3 finds no expansion covering the base plus both of the "easiest" to merge product terms found in Step 2 so, S_6 is the product term having the same cover as PT_{10} and PT_{11} (note that S_6 is not prime). For the seventh and eighth base product terms, it finds PT_{13} and PT_{15} repectively; these two were not expanded by PRONTO. As the ninth base selects PT_1 is found to be mergeable only with PT_2, which makes S_9 equal to the product term having the same cover as PT_1 and PT_2. For each of the remaining product terms, Step 2 finds no "easiest" to merge product terms. Thus, PRONTO makes $S_{10}=PT_4$, $S_{11}=PT_7$, $S_{12}=PT_{16}$ and $S_{13}=PT_{17}$. The size of the solution found with PRONTO is 13.

S. Kang reported the solution to the above example when applying SPAM. SPAM gave a solution with the same size as the one obtained by PRONTO. However, the expected computational complexity in PRONTO can be better. Furthermore, there is no reason that prevents us from using PRONTO with the partitioning idea for even better results.

The next section discusses the worst and the expected complexity in PRONTO.

G. PRONTO's Complexity

In this section PRONTO's complexity is briefly described. We shall start by individually analysing its four steps.

Step 1 is obviously linearly dependent on the number of uncovered product terms, N_{UPT} since it only goes once through each uncovered product term in order to find a base product term. Therefore, the complexity of Step 1 is $O(N_{UPT})$.

Step 2 classifies each uncovered product term with respect to the current base product term. The classifying operation depends linearly on the number of inputs and outputs, N_{IO}. Thus, the complexity in Step 2 is $O(N_{UPT}N_{IO})$.

Step 3 first finds a set of product terms adjacent to the base product term. This can cause up to N_I+1 calls to the cover checking procedure. Second, the actual expansion takes place and when using a tree expansion the worse case would be to call the cover checking procedure $2^{(N_I+1)}-(N_I+1)$ times. Thus, the complexity for Step 3 is $O(2^{(N_I+1)}*C_c)$, where C_c is the complexity of the cover checking procedure used.

Step 4 is the updating step which depends linearly on the number of uncovered product terms, which have been fully or partially covered by the expanded base product term N_{CPT}. Therefore, the complexity for Stept 4 is $O(N_{CPT})$.

So far we have analysed the worst case complexity which varies exponential with the number of inputs. However, the expected complexity is far better.

To establish expected complexity we make the following assumptions which are satisfied in practice.

A1. The number of uncovered product terms adjacent to the base product term is less than a constant K_{ADJ} and it is independent of N_I.

A2. In order to find a "best" expanded base product term, only a portion of the expansion tree is searched. This portion calls the cover checking procedure no more than $(N_{ADJ})^4$ times, where N_{ADJ} is the number of adjacencies found for the base product term.

Given A1 and A2 the complexity of Step 3 is $O((K_{ADJ})^4*C_c)$ rather than $O(2^{(N_I+1)}*C_c)$. It follows that the expected complexity of all four steps is

$$O(N_{UPT}+N_{UPT}N_{IO}+(K_{ADJ})^4*C_c+N_{CPT}).$$

Hence, the expected complexity of PRONTO is $O(N_S\{N_{UPT}N_{IO} + C_c + N_{CPT}\})$, where N_S is the number of product terms in a solution.

From the expected complexity formula of PRONTO, two characteristics can be seen. First, PRONTO depends linearly on the number of product terms in a solution. Second, the developments in PRONTO <u>minimize</u> the expected number of calls to the cover checking procedure. These two characteristics of PRONTO give an expected speed improvement over previously published product term reducers.

After presenting PRONTO, there is only one more chapter left to present. It contains the summary, results and conclusions.

6. SUMMARY/RESULTS/CONCLUSIONS

A. Summary

A new appropriate nomenclature for product term reduction methods was presented. Also, a modification to solve the order dependence of the reduction procedure in PRESTO and the irredundant cover in MINIMAL was given. The major contribution of this research has been the development of PRONTO as a practical product term reduction approach. PRONTO is a one-pass, heuristically guided direct search method using four steps to provide the guided direct search for the selection of the solution's elements. These four steps are as follows.

First, select an uncovered product term as a base term for expansion.

Second, find a set of "easiest" to merge uncovered product terms.

Third, properly expand the base product term so as to cover "most" uncovered product terms which are directly mergeable plus any additional uncovered ones.

Fourth, do the housekeeping to record any covered and modify any half-covered product terms as result of the expanded base product term being part of the solution.

PRONTO can be very useful as part of a reducing PLA generator.

B. Results

The results of applying PRONTO, as described in this dissertation, to eight examples has been provided by D.W. Brown as summarized in Table 1. Each of the rows in Table 1 represents an example. The number in the leftmost column is the example number; the second column's number is the number of inputs in the example; the number in the third column represents the number of outputs in their corresponding example; the fourth column provides the number of terms in the example's specification; the fifth column is the size (number of product terms)of the solution when applying PRESTO [BROWN] and the last column gives the size of the solution obtained when PRONTO was used.

TABLE 1. Summary of Results

Example	Inputs	Outputs	Terms	Reducing with	
				PRESTO	PRONTO
1	6	6	31	14	12
2	10	9	26	15	14
3	9	8	33	18	17
4	14	5	85	47	47
5	9	14	47	23	22
6	16	16	103	39	39
7	15	23	105	42	33
8	12	25	235	69	58

In all examples, PRONTO gave no worse results than PRESTO and in six of them PRONTO improved the results. PRONTO can be expected to be faster due to its complexity.

C. Conclusions

As an approach to the product term reduction problem, PRONTO improves results of a previously published method. PRONTO has three characteristics which make us believe it is faster than some previously published methods. These characteristics are:

1) The expansion of a base product term is limited to a set of possibly mergeable directions.

2) Expansion does not seek prime terms, only "best" coverage of uncovered product terms.

3) The complexity is linearly dependent on the number of product terms in a solution. Thus, we can expect faster results for better solutions.

PRONTO's expected complexity tells us that this approach minimizes the use of the cover checking procedure.

PRONTO can be tuned to the user's needs by relaxing the requirements for the "easiest" to merge set members and allowing a more extensive search for coverage of uncovered product terms, and with the use of more effective approches to cover checking procedures like the ones used in MINIMAL.

To help show that our results are not restricted to a unique class of functions, a function which clearly belongs to the easily partitionable class was presented as an example in Chapter 5. Of course, one could also contemplate the use of PRONTO in a divide and conquer (partitioning) scheme, where PRONTO reduces the product terms for each partition.

PRONTO is simple enough as to be a practical manual method for reasonably large functions.

More work remains to be done. It is highly desirable to have a very cost-effective procedure for cover checking, since it is a most often called procedure in product term reduction. For those functions whose minimal sum-of-products solution require a large number of small product terms, one might be better off by using external logic at the inputs and/or the outputs of the PLA. The purpose of this logic is to reduce the number of product terms in the PLA such that an overall savings in real estate results. Thus, a cost-effective technique to find appropriate external logic is desirable. The extra delay and random logic introduced can pay off when substantial savings in real estate result. Of course, techniques for combining different PLA reduction methods (product term reduction, folding, etc.) to give a better overall area reduction are very much desirable. A specially interesting combination is a method for product term reduction giving highly foldable solutions, which could be combined with a folding procedure and lead to a substantial reduction in real estate.

Our experience makes us believe that PRONTO can prove to be useful when applied to large practical problems.

7. BIBLIOGRAPHY

AREVALO,Z. and BREDESON J.G.,"A Method to Simplify a Boolean Function into a Near Minimal Sum-of-Products for Programmable Logic Arrays," IEEE Trans. Comput., vol. C-27, pp. 1028-1039, Nov. 1978.

BRAYTON,R.K., et. al., "A Comparison of Logic Minimization Strategies Using ESPRESSO: An APL Program Package for Partitioned Logic Minimization," Proceedings of ISCAS82, pp 42-48, May 1982, IEEE Catalog number 82CH1681-6.

BRICAUD,P. and CAMPBELL, J.,"Multiple Output Minimization: EMIN," WESCON 78, paper 33/3.

BROWN,D.W. "A State Machine Synthesizer- SMS," 18th Design Automation Conference, pp. 301-305, Jun. 81.

FAIRBAIRN,D.G. and HAINES,A.L., "New techniques and tools ease design of custom VLSI," Electronic Design, pp. 187-191, Jan. 21, 1982.

FLETCHER,W.I., AN ENGINEERING APPROACH TO DIGITAL DESIGN, 1980 Prentice-Hall, Inc., Englewood, New Jersey 07632.

GAREY,M.R. and JOHNSON,D.S. COMPUTERS AND INTRACTABILITY: A Guide to the Theory of NP-Completness, W.H. Freeman and Company, San Francisco, 1979.

GIMPEL,J.F.,"A Method of Producing a Boolean Function Having an Arbitrarily Prescribed Prime Implicant Table," IEEE Trans. Comput., vol. EC-13, pp. 485-488, Jun. 1965.

GRASS,W., "Implementing a Set of Switching Functions in Terms of Programmable Logic Arrays (PLA)," Digital Processes, vol. 6, no. 1, pp. 75-96, 1980.

HACHTEL,A.L. et. al.,"Some Results in Optimal PLA Folding," Proceedings of the IEEE International Conference on Circuits and Computers ICCC80, Pt. II, pp. 1023-7, Oct. 1980.

HONG,S.J. et al., "MINI: A Heuristic Approach for Logic Minimization," IBM J. Res. Develop., Vol. 18, No. 5, pp. 443-458, Sept. 1974.

KAMBAYASHI,Y., "Logic Design of Programmable Logic Arrays," IEEE Trans. Comput., vol. C-28, pp. 609-617, Sept. 1979.

KANG,S., "Synthesis and Optimization of Programmable Logic Arrays," PhD Dissertation, Stanford University, 1981.

MARTINEZ-CARBALLIDO,J.F. and POWERS,V.M., "General Microprogram Width Reduction Using Generator Sets," *Proceedings of the 14th Annual Microprogramming Workshop*, pp. 144-153, Oct. 1981.

MEAD,C. and CONWAY,L., *Introduction to VLSI Systems*, Addison-Wesley, Reading, PA., 1980.

PAILLOTIN,J.F., "Optimization of the PLA Area," *18th Design Automation Conference*, pp. 406-410, Jun. 1981.

RHYNE,V.T., *et. al.*, "A New Technique for the Fast Minimization of Switching Functions," *IEEE Trans. Comput.*, Vol.C-26, pp 757-764, August 1977.

RIHERD,F.T. *et. al.*, "Mead-Conway designing: No IC experience needed," *Electronic Design*, pp. 115-120, March 4, 1982.

ROTH,J.P., "Decreasing the Size of Associative Logic Arrays," *IBM Technical Disclosure Bulletin*, vol. 17 no. 4, Sept. 1974.

SASAO,T.,"Multiple-Valued Decomposition of Generalized Boolean Functions and the Complexity of Programmable Logic Arrays," *IEEE Trans. Comput.*, vol. C-30, pp. 635-643, Sept. 1981.