A SURVEY OF LITERATURE ON FUNCTION DECOMPOSITION
VERSION IV
November 20, 1995

Marek A. Perkowski, Stanislaw Grygiel,
and the Functional Decomposition Group,
Department of Electrical Engineering

Portland State University,
P.O. Box 751
Portland, Oregon, 97207-0751
mperkows@ee.pdx.edu

Sponsored by:
Air Force Office of Scientific Research
Bolling Air Force Base, DC

and

Wright Laboratory

Abstract

This report surveys the literature on decomposition of binary, multiple-valued, and fuzzy functions. It gives also references to relevant basic logic synthesis papers that concern topics important for decomposition, such as for instance representation of Boolean functions, or symmetry of Boolean functions.

As a result of the analysis of the most successful decomposition programs for Ashenhurst-Curtis Decomposition, several conclusions are derived that should allow to create a new program that will be able to outperform all the existing approaches to decomposition. Creating such a superior program is necessary to make it practically useful for applications that are of interest to Pattern Theory group at Avionics Labs of Wright Laboratories.

In addition, the program will be also able to solve problems that have been never formulated before. It will be a test-bed to develop and compare several known and new partial ideas related to decomposition. Our emphasis is on the following topics:

1. representation of data and efficient algorithms for data manipulation,
2. variable ordering methods for variable partitioning to create bound and free sets of input variables; heuristic approaches and their comparison,
3. column compatibility problem,
4. subfunction encoding problem,
5. use of partial and total symmetries in data to decrease the decomposition search space,
6. methods of dealing with strongly unspecified functions which are typical for machine learning applications,
7. special types of decomposition, that can be efficiently handled (cascades, trees without variable repetition).

We would like to acknowledge Dr. Tim Ross, Mr. Mark Axtell, and Professors Robert Brayton, Malgorzata Marek-Sadowska, Tsutomu Sasao, Tadeusz Luba, Bernd Steinbach, Maciej Ciesielski, Bernd Becker, Radomir Stankovic, and Randall Bryant for their help in finding some of the surveyed materials.

We would also like to thank the Air Force Office of Scientific Research for providing the opportunity and support to do this research.

The members of the Functional Decomposition Group at PSU are: Ralph Almeria, Paul Burkey, Michael Burns, Karen Dill, Stanislaw Grygiel, Nick Iliev, Feng Yang, Robert Lisanke, Jing Lu, Rahul Malvi, Sanog Mohammad, Dhinesh Manoharan, Robert Price, Roger Shipman, Sridhar Srinivasan, Cynthia Stanley, Zhi Wang, Hongfei Wu, Sida Zhou, and Jin S. Zhang.

[0] †The work presented in this paper was supported by the AFOSR grant and administered by the RDL program.

Contents

List of Figures

1 Introduction

Pattern Theory group at Avionics Laboratory of Wright Laboratories applies the so-called Ashenhurst-Curtis Decomposition of Boolean functions as a new Machine Learning paradigm [537]. Many decomposition ideas have been implemented in the programming system Flash developed by this group [540]. Flash is the system of programs created to implement and experiment with various variants of Functional Decomposition and associated Boolean transformations.

Although much research has been done recently world-wide on the decomposition of Boolean functions for logic circuit design applications, it seems to be a lack of recent monograph books or even survey papers in the area of Boolean and Multiple-Valued Logic Decomposition, despite their important applications in circuit design and Pattern Theory. This survey is an attempt to fill this gap.

The functional decomposition is an extremely wide and multifaceted topic, it is located in the very center of the area of logic synthesis. Therefore, in writing this survey, we had to be restricted to some specific point of view; some kind of special interest in the decomposition. We decided to look at the issues from the point of view of their usefulness to the goals of the Pattern Theory group at Wright Labs, [537].

1.1 Goals of Pattern Theory Group

The goals of the Pattern Theory work, as we understand them, are the following:

- To create practical Machine Learning algorithms that will give useful results on military data. Which means, at the minimum, functions of 30 binary-input, binary-output variables. Better, about 100 variables.

  The program must be robust across various classes of data. It must have better generalization capabilities than the well-known "Decision Tree" constructing program C4.5, as well as other recent decision-tree-based learning or decision-diagram-based learning programs. (The decision-diagram-based approaches were presented in 1994 by Ron Kohavi [338] and Arlindo Oliveira [474].) The new program is also expected to be faster than the current version of Flash.

- To make contribution to the methods of "discovery". Upgrade Flash in such a way that it would be able to "invent" new algorithms and digital structures. For instance: it took about 100 years from the invention of discrete Fourier Transform to the discovery of Fast Fourier Transform by Cooley and Tucker. We would like to be able to find similarity patterns in data in a few seconds, and create a structure of an FFT automatically. Flash can already "invent" simple circuits such as an adder, but is unable to find more complex circuits, such as a combinational sorter, comparator, maximum of k numbers, lattice realizations of symmetric functions, and butterfly architectures. One of the reasons of this inability is that at present, Flash cannot deal with multi-output functions. Second, it cannot detect symmetries or some other global properties of the underlying Boolean functions.

- To advance the state of the art of the **decomposition theory** with respect to many new aspects of decomposition that are specific to Machine Learning applications. It must result not only in a better program, but also, in original and new research results that will be appreciated by the research community, as measured by high quality conference and journal papers. The papers should make *substantial* and *fundamental* contributions to the theory of decomposition especially when applied to Machine Learning.

There are other points to be made about a special nature of the decomposition for Pattern Theory. The first very important observation is the following:

In the past, the work in the area of Ashenhurst-Curtis Decomposition (AC decomposition) was conducted for application in circuit design, [32], [153]. For problems taken from engineering practice in circuit design, when the number of input variables increases, the number of don't cares increases slightly with the number of minterms (the minterms are also called *true minterms* or *positive samples*), and the number of zeros (called also *false minterms* or *negative samples*) increases dramatically. This kind of examples can be found in the well-known benchmark sets for logic synthesis: MCNC Benchmarks, ISCAS Benchmarks, and other. These benchmarks have been created by designers from industrial companies and universities, and deal with Boolean functions from: computer, DSP and telecommunication, such as large controllers, state machines, decoding logic, arithmetic and interfaces.

This is in sharp contrast to practical functions from Machine Learning applications, where, with the increase in the number of input variables, there is only a small increase in the number of both positive and negative samples, but a dramatic increase in the number of don't cares. For instance, it is reasonable to expect that for a function of 100 input variables there will be not more than 10,000 cares.

This distinction makes completely different requirements on the decomposition programs in the both areas; "circuit design", and Machine Learning, a point that has been not yet sufficiently observed and appreciated. *This fact calls for the development of totally new approaches to synthesis, and is a very positive opportunity for the Pattern Theory group*, since if properly utilized, it would allow the group to become the leaders in the field. Instead of adapting the algorithms created for circuit design, the group should create new algorithms *from scratch*, taking into account the specifics of a problem as the main characteristics.

1.2 Variants of the System

We believe, that with respect to what was mentioned in previous subsections, the following separate variants of Flash system should be created.

1. Decomposition to minimize DFC. DFC stands for *Decomposed Function Cardinality.* DFC of a circuit decomposed to blocks is the sum of DFC measures of the component blocks. The DFC of a block is $n2^k$ where k is the number of block inputs and n is the number of block outputs.

2. Decomposition to find good extrapolation of don't cares. This is assumed to be the same as the "DFC minimization", if a really minimum value of DFC is found. However, any decomposition method has some kind of bias, even an obvious bias of having disjoint sets of variables in the bound and free sets. Moreover, unavoidably an additional bias must be introduced for functions with many variables, where all solutions to partial combinational problems will be approximate because of the high-order NP-hardness of the entire problem. It is then not sure that the DFC is the absolutely best measure to be used as a heuristic in the decomposition process.

3. Decomposition to recognize hardware structure. Such decomposition would require, at the very minimum, the introduction of multi-output functions and blocks with many outputs, which currently do not exist in Flash.

Of course, these variants can share most data structures and routines, but we feel that these goals would be easier to satisfy in separate "limited" versions, and not in a single "super-version", that would try to "do everything right."

1.3 Terminology and Sources of Information

In the most general formulation, the problem of decomposition of a single-output Boolean function F is to find a representation of F which consists of a composition of simpler Boolean functions. There are several types of decomposition, so to be precise about the subject of our presentation, we will first give respective definitions. Decomposition is the process of expressing a function of n variables as a function of functions of fewer variables. For instance, function $F(X)$ is decomposed to functions H and G, such that $F(X) = H(A, G(B))$, where A and B are proper subsets of the set of inputs $X = A \cup B$, $A \cap B = \emptyset$. When $A \cap B = \emptyset$, the decomposition is called a "disjoint decomposition", otherwise it is called a "non-disjoint decomposition". G is a single-output function. H and G are called *component* *functions* of F, G is called a *predecessor function* and H is called a *successor function.* H has fewer variables than F. A is called the *free set*, and B is called the *bound set.* In Ashenhurst formulation function G has one output. In Curtis formulation function G can have more than one output, but no more than the number of variables in B.

The main assumption of Flash is that it uses the Curtis type of Boolean Decomposition, in order to minimize some cost measure, the DFC. However, an important point is that there is some confusion in the technical literature, graduate textbooks, and understanding of experts in the field, of what actually constitutes the Ashenhurst/Curtis Decomposition and its many special cases and extensions.

The same type of decomposition is discussed under different names in the literature, when applied to different data representations. This in our opinion, is not correct. On the other hand, the name Ashenhurst/Curtis Decomposition has been used to practically quite different types of decomposition, linked only by their superficial references to the original papers by Ashenhurst and Curtis. An attempt has been made while preparing this survey to find the original papers, give precise and uniform definitions, and attribute the name of the decomposition to its original inventor.

Until now, the main source of information on decomposition in USA is the book by Curtis [153] that presents early American research on what is now called the Ashenhurst-Curtis decomposition. Other monograph books include: Bochmann and Steinbach [83] (in German) that presents mostly German and European research, Bibilo and Yenin [70] (in Russian) that presents only the research from the former Soviet Union, and two monographs by Luba [376, 382] (in Polish) that concentrate mainly on the approach developed at the Warsaw University of Technology.

We will define precisely the Ashenhurst decomposition [32], the Curtis decomposition [153], the Flash decomposition [537], the XBOOLE decomposition [83], the Perkowski et al (PUB) decomposition [486, 488, 678], the Luba decomposition [376], the Roth/Karp decomposition [548, 317], and others.

By defining a decomposition, our main distinguishing criterion will be the number of decompositions possible for a given type of decomposition, and not the data representation or some secondary implementation details of the algorithm.

We found that the following types of decomposition are the most interesting for the above-mentioned requirements of the Pattern Theory:

- Ashenhurst decompositions, defined in [32];
- Curtis decompositions, defined in [153];
- Roth/Karp decomposition [548, 317, 550, 549, 318];
- Sasao multiple-valued PLA decomposition [575, 577, 580];
- the PUB decomposition of Perkowski, Hoang, and Brown, [486, 488];
- Yang/Ciesielski PLA decomposition [721, 722, 724];
- Luba's Partition-Based binary [376, 382], and multiple-valued decomposition [396, 397];
- Bochmann/Posthoff/Dresig/Steinbach algebraic AND/OR/EXOR decomposition [83], so-called XBOOLE Decomposition;
- Zakrevskij/Bibilo decomposition [734, 70];
- Orthogonal decompositions of Perkowski et al. [492, 494, 496]; and
- Lai/Pedram/Vrudhula approach based on BDDs [351].

2 Classes of Functional Decomposition

The questions to be answered in this section are:

1. Does a decomposition which creates blocks having more outputs than inputs belong to the category of Ashenhurst/Curtis Decomposition ?

2. Is Roth/Karp decomposition [548, 550, 317] an example of AC decomposition ?

The answer to the first question is no. According to the criteria mentioned in section 1 and applied in this paper, we will not treat such a decomposition as an Ashenhurst/Curtis (AC) decomposition. Following the original papers, only a decomposition which creates blocks with the number of outputs smaller then the number of inputs will be denoted as being the AC decomposition, otherwise there is no DFC gain. A decomposition that creates blocks which have the number of outputs greater than or equal to the number of inputs is certain generalization of the AC decomposition but is not the AC decomposition. All these types of decompositions will be called the *Functional Decompositions.* Ross et al [537] defined such a general decomposition, but it is not actually realized in Flash.

The answer to the second question is yes, but the Roth/Karp decomposition uses a different data representation. Analogously, the Shen/Kellar [607, 608, 609] and Varma/Trachtenberg decompositions [701] are particular cases of the AC decomposition.

When the blocks have overlapping inputs, then every circuit structure, for instance a SOP, or a multi-level NOR network [640], can be obtained by some sort of decomposition. We will call them switching circuit decompositions, or Boolean Decompositions in case of binary functions. Some of these methods have been called "decompositions" by their authors. Again, only if the number of inputs is larger than the number of outputs of a block, and the concept of the multiplicity index is used, the decomposition will be called the AC Decomposition.

We will base our definitions on books on decomposition that have been published until now: Curtis [153], Bibilo [70], Steinbach [83], Luba [376, 382], as well as on the graduate textbooks in logic synthesis: Zvi Kohavi [336], Prather [520], Bolton [85].

We will then define the following types of decomposition:

1. Ashenhurst decompositions, defined in [32].

2. Curtis decompositions, defined in [153].

3. Roth/Karp decomposition [548, 317].

4. Sasao multiple-valued PLA decomposition [577].

5. Ross decomposition, defined in [537] (called Ashenhurst-Curtis there).

6. Perkowski/Hoang/Brown decomposition (PUB), [486, 488].

7. Yang/Ciesielski PLA decomposition [721, 722, 724].

8. Luba's Partition-Based binary and multiple-valued decomposition [376, 382].

9. Bochmann/Posthoff/Dresig/Steinbach algebraic AN/OR/EXOR decomposition, so-called XBOOLE Decomposition [83].

10. Zakrevskij/Bibilo decomposition [734, 70].
11. Orthogonal decompositions of Perkowski [492, 494, 496].
12. Pedram's decomposition using BDDs [351].
13. Murgai/Brayton's decomposition using BDDs [453, 455].
14. Jozwiak generalized decomposition [312, 313].

What was called Ashenhurst-Curtis Decomposition (ACD) by Ross et al [537] in the past, now will be called a Functional Decomposition (FD). This will be a kind of "mother of all methods", but only for the family of decompositions, which are based on what Tim Ross calls "basic decomposition condition" [537] (counting the column multiplicity). According to this criterion, the decompositions of XBOOLE [83], Yang/Ciesielski [721] or orthogonal [492], will be not included to the class of FD. However, Ashenhurst [32], Curtis [153], Ross [537], Sasao [577], Perkowski et al. [486, 488], and Zakrevskij/Bibilo [734, 70] will be examples of Functional Decompositions. Next, particular decomposition cases attributed to various authors will be derived from the FD model, by imposing various constraints on the general FD model.

Other decompositions will be discussed in much less detail, and only if we feel that discussing them may be useful for improving the decomposition model used in Flash. The question arises:

- *'If the FD is applied without any restrictions, will it generate every possible circuit? Can then every circuit structure be treated as a special case generated by certain restrictions imposed on the FD model?'*

The answer is no. For instance, SOP and ESOP circuits, or Dietmeyer-like decompositions [175] cannot be created as particular cases of this model. It is, however, perhaps possible to further expand the FD model to include these classes, but it will be not attempted in this survey report.

Concluding, the decompositions based on column multiplicity criterion will be called the Functional Decompositions (FD). If they assume in addition less outputs than inputs to every block, they will be called the Ashenhurst/Curtis Decompositions (ACD). In addition, if there is only one binary wire from the predecessor block to the successor block, the decomposition will be called the Ashehurst Decomposition. For more than one wires connecting the predecessor and successor blocks, the AC decomposition will be called the Curtis decomposition.

3 The Goals of this Survey.

The goals of writing this survey were the following:

1. Find what work has been already done by other researchers - not to repeat the same effort again.
2. Find what approaches to decomposition did practically work well when implemented in computer programs.
3. Find some partial ideas, heuristics, or representations that can be borrowed, integrated, and enhanced into our future improved programs.
4. Find what are the "open research problems" related to decomposition, or just what are the problems that have been not yet formulated nor solved.
5. Find how people are approaching the problems that we found most important in our research in the past:
   - representation.
   - variable partitioning.
   - coloring/covering for column compatibility.
   - column encoding.
   - subfunction reusing.
   - vacuous variable reduction.

Here, our goal will be not to present systematically all ideas that can be found in the decomposition literature. We feel that there is a lot of repetitions and re-formulations in the published papers and the authors seem to be not familiar with fundamental papers published previously. Some papers are strictly and solely related to K-map representations, that we consider obsolete, and besides, do not introduce any new ideas. Some other are only reformulations of known ideas to the cube calculus representation.

Our survey concentrates on papers and ideas that seem to be the most fruitful for the implementation of decomposing programs. Especially the implementation of the approximate programs for a large number of incompletely specified variables, but also for the exact programs that may be useful to evaluate the quality of the approximate programs.

Therefore, we devote a considerable amount of attention also to some general logic synthesis problems that will be useful to design such decomposers. These general problems include:

- representation issues, such as the use of various types of decision diagrams to represent functions,
- provable search constraints, such as finding first partial or total symmetries in input variables of the functions, and then use these symmetries to limit the search space of the bound sets of variables, without sacrificing the quality of the decomposition.
- applications of heuristic methods to reduce the search, such as variable ordering techniques for BDDs that are strictly related to the problem of variables partitioning into bound and free sets.

We have tried to give the list of literature as complete as possible. It includes some really rare internal reports (some of them, but not all, are available to us). We provide also the literature positions that we have found only indirectly - in other referenced papers (or through personal conversations). We are continuously collecting literature on decomposition, so future editions of this survey will include more references on both new publications, and existing publications that are not yet available to us.

4 Functional Decomposition Versus Algebraic Methods.

4.1 Decomposition is not 'Algebraic'.

Binary Decomposition is not a Boolean concept, it means, that it can be explained without using the concept of Boolean algebra and gates. What is necessary is only to describe a function (as a discrete mapping) in terms of composition of other functions. An analogous property is true for multiple-valued signals. The concept of decomposition can be explained without resorting to Post logic, Galois logic, or any other specific multiple-valued algebra to realize the functions, [530, 236]. Therefore, one can offer a very abstract and comprehensive treatment of decompositions of the binary, multiple-valued and fuzzy logic functions.

However, the above observation does not mean that some kind of 'algebraic' concepts cannot be used in the decomposition process. For instance, in binary decomposition one can use Binary Decision Diagrams [111] for data representation, and the Binary Decision Diagrams are a concept of Boolean algebra. A similar property is true for the multiple-valued data, where multiple-valued logic diagrams can be used instead of binary decision diagrams to represent the functions.

The main advantage of the functional decomposition approach of the current Flash program is that it is not using any algebraic or even Boolean methods. (Although the Flash program is now only for binary logic, the approach applied in it can be very easily expanded to the logic with multiple-valued inputs and outputs, which is exactly one of generalizations that will be presented below.)

In other synthesis methods, one assumes certain sets of blocks to realize the 'gates', for instance the AND, OR, and EXOR set [83], the NOR set [161], the AND/EXOR set [622, 623], or a multiplexer set [488]. Such approach is currently dominating in the industry (the technologies of standard cells, PLDs, sea of gates, etc.). Although the 'universal logic module' approaches, such as the 'lookup-table' approach to design with Xilinx FPGAs, assume, similarly to Flash, arbitrary lookup tables as 'gates', but, contrary to Flash, these approaches assume a limited fan-in to these tables (for instance; 4, 5 or 6 inputs are used in the older version of TRADE, [678]). This is in a clear distinction to the approach of Flash, where no restriction on the number of inputs to a logic block is assumed.

There are also some other kinds of decomposition, for instance the approach in the XBOOLE system, that assume a decomposition to certain types of Boolean gates. In the case of XBOOLE two-input AND, OR, EXOR, and an inverter. Even though we find such decompositions interesting from the Pattern Theory point of view, and these decompositions have relation to the functional decompositions, in this survey we will concentrate only on presenting the approaches that have a similar 'philosophy' to Flash.

The approach used in Flash program has the following consequences:

- It cannot be used directly to hardware realization of circuits, and some another logic minimization programs, such as Espresso, must be used to minimize and realize the blocks which are non-decomposable by the method of Flash. In the future, another 'gate-oriented' decomposers or synthesizers can be used inside Flash on equal terms with Espresso. The realization of non-decomposable blocks, seems however to be of only a secondary importance from the Machine Learning point of view.

- There exists a possibility that the cost evaluation (DFC) of large blocks is not an accurate prediction of complexity. The argument for the current approach in Flash is, however, that such blocks are rarely created by the program.

- The FD approach can be easily expanded in a natural and general way to multiple-valued, fuzzy and continuous logic. Such an extension would be practical when one assumes a software realization of arbitrary complexity blocks. However, one of the very important goals of the Pattern Theory research is to combine the advantages of Flash for discrete data with the advantages of the ABTECH's 'abductive network' approach for continuous data [6]. ABTECH's approach is to compose a network from certain types of blocks rather than decompose. It uses particular kind of continuous blocks: third order polynomials of real-valued input variables. Such blocks are analog circuit (continuous logic) counterparts of the limited-fan-in logic gates in the logic design area. If the current approach of Flash were used for continuous data, then some additional mechanisms should be developed to map the 'non-decomposable blocks' to the polynomial blocks. There may be a concern that such a mapping will not be an optimal one. One then may ask a question: 'why not to try to compose the function from existing types of blocks?' Since, in any case, for continuous functions, ABTECH has to assume certain blocks, such as polynomials, why not to try to uniformly apply the composition methods for both multiple-valued and binary variables? Such an approach, though totally different from the one applied in Flash, definitely appears to be reasonable, and has been used in both digital and analog design [343]. Therefore, some papers on this type of decomposition, with blocks taken from certain library of blocks, will also be briefly reviewed in this survey.

- The current philosophy of Flash is that it is better to assume no knowledge on the types of gates, because assuming any kind of element would be considered a dirty patch-up, that would only add a bias. An argument against this philosophy is that, in any case, assuming some kind of bias is necessary to effectively construct some objects, or to learn some patterns. One possible approach would be to have one bias compete with another bias in order to produce the best learning algorithm. If there were no bias at all, there would be a danger that never-ever anything interesting would be found for larger data sets because the search space would be prohibitively large. The current approach of Flash is that first we want to use FD to investigate the whole space of solutions without a bias. Then we want to study how imposing various kinds of constraints would affect the quality and the structure of solutions. As already noted, other approaches are also possible, including the ones that would assume, or 'guess and check', several specific biases.

4.2 The Controlling of Biases in the Decomposition.

There are many possible ways of dealing with the 'bias' problem.

1. One possible bias is to restrict the search space based on some heuristic principles. For instance, the algorithm created by Craig Files [215] restricts heuristically the search space of all variable partitions to a subset of the original space of all partitions. By using this particular algorithm, and not some other search-restricting algorithm, we make a certain decision. Such decision is a bias in itself.

2. Certain structural properties of the decomposed function can be assumed, which would sometimes allow to prove that searching some subspace of the entire space would be sufficient to find the exact solution in this narrower sense. For instance, this can be achieved by investigating partial and total symmetries of the input variables [175, 332].

3. A certain bias is introduced by assuming blocks with less than 5 inputs, or some other small number of inputs. This bias may be useful when a circuit realization of the decomposition is considered.

4. Any other heuristic way of constraining the search space is a bias.

5. Some other methods of restricting and controlling the decomposition search have been shown by Steinbach [637, 638] (for improving testability, and for improving power consumption - Schneider [597]), Luba [397, 601, 602], and other authors. They select one of few types of available decompositions, depending on the satisfaction of some conditions by the data.

An ideal solution would be to have many methods built into a general decomposition method. This would allow a parameterized user control of the decomposition process. This would be like a user-controlled, dynamical bias, possibly tunable or learnable with each data set.

6. An universal method which would generate specialized methods for particular types of restricted structures. In the past the following structures have been investigated:

   - various kinds of trees [114, 118],
   - Maitra cascades [406],
   - tandem networks of Butler [116],
   - partially symmetric sub-functions [175, 332].

   All these networks are particular cases of the FD decomposition.

   Although we are not aware of using such an approach within the FD context, the OR / AND / EXOR strong-weak decomposition of Steinbach [637, 638] or some special AND, OR and EXOR input decompositions described by Luba [376, 382] and Wnek [689] are good examples of similar decomposition approaches outside the AC framework.

One of the goals of this survey will be to develop an understanding of how the different constraints affect the final DFC value, the generalization abilities of the algorithm, and its learning time. While surveying the literature, we will be paying a special attention to all the issues of the bias problem in multi-level circuit synthesizing programs.

In the following sections we will list other important issues, related to the decomposition, which have guided us in selecting the papers to be covered in this survey.

5 Research Areas Related to Functional Decomposition Applicable to Pattern Theory.

In this section we will list the research topics and options which exist in functional decomposition.

1. **Type of variables.** On what kind of data the algorithm operates. We assume that in all cases the don't cares exist in data representation.
   - (a) **Binary logic** (each variable is from a set 0,1), [32, 153, 376, 537].
   - (b) **Multiple-valued data** (each variable is from a finite set of integers), [211, 396, 397, 677].
   - (c) **Fuzzy** (each variable is from a continuous interval [0,1]), [314, 221].
   - (d) **Analog, or continuous** (each variable is an arbitrary real number), [544].
   - (e) **Integer** (each variable is an arbitrary integer), introduced in [112], but not for AC decomposition.
   - (f) **Mixed**. New research area.
2. **Representation of Functions.**
   - (a) **Lists or Arrays of Samples**, [537, 32, 153, 376].
   - (b) **Binary Cubes and Multiple-Valued Cubes** [83, 637, 638, 70, 678].
   - (c) **Binary Decision Diagram (BDD)** [111]. **Ternary Binary Decision Diagrams** with extension for don't cares - three terminal nodes: $0, 1, X$ [351], the method most successful computationally until 1994.
   - (d) **A pair of BDDs, one for ON-set and one for OFF-set** to specify an incompletely-specified function. A new approach, proposed below.
   - (e) **Kronecker Decision Diagram (KDD) [179] with extension for don't cares,** or **a pair of KDDs, one ON-set and one for OFF-set.** These representations have not been proposed yet for decomposition.
   - (f) **Other binary decision diagrams:** zero-suppressed [435], pseudo-KFDD [579], EVBDDs [351], etc. These are new approaches which have not been applied to decomposition yet (with an exception of EVBDDs in Pedram).
   - (g) **New classes of Decision Diagrams that expand data beyond binary**: Orthogonal [494, 496] multiple-valued KFDD, multiple-valued pseudo KFDD, symmetric diagrams, Bryant's Binary Moment Diagrams [112], etc. These are all new approaches, none used for decomposition yet.
   - (h) **Fuzzy Functions** and **Fuzzy Decision Diagrams**. These are both new approaches.
3. **Input Variable Partitioning to Free and Bound Sets.** How to find all, some or few good bound sets.
   - (a) **State-space Search** (simple genetic search [226], genetic classifiers, expert systems, heuristic search [697, 215], search in planning spaces.)
   - (b) **State-space Search-restricting theorems.** [491].
   - (c) **State-space search-restricting heuristics.** [491].
   - (d) **Applying learning to improve search performance.** [490].
   - (e) **Use strong heuristic to generate just a few candidates.** [678].

(f) **BDD-based.** [351, 697].

(g) **Information theory based** - correlation of variables, order of variables [525, 402].

(h) **Symmetry based.** [175, 332].

4. **Column Minimization Problem,** [488, 678, 537, 99]. How to find the minimum number of compatible columns?

   (a) **Graph-coloring approach** for the Incompatibility Graph [488, 678].

   (b) **Set covering approach** for the Compatibility Graph [376, 70, 550].

   (c) **"Set covering with weights of rows"** for the Compatibility Graph. New approach.

   (d) **Covering/assignment approach of Bibilo et al** for the Compatibility Graph, [70]. (This method solves the encoding problem concurrently with the column minimization problem.)

   (e) **Reduced Covering approach of Luba et al** for the Compatibility Graph, [397].

5. **Encoding of groups of columns.** Columns are also called *symbolic sub-functions,* or *cofactors* with respect to variables from the bound set. Sets of columns are encoded to simplify the logic.

   (a) **Encoding based on the Fast Linear Ordering.** Wei Wan et al, [678].

   (b) **Perkowski/LocNguyen fast rule-based heuristic encoder** [465].

   (c) **Quadratic Assignment algorithms.**

   (d) **Sasao's method** (one hot code and binary code with maximum zeros [575, 577, 580]).

   (e) **Yang and Ciesielski's approach.** [721, 724].

   (f) **Devadas' approach.** [171, 174].

   (g) **Spectral Approach** (Varma and Trachtenberg [701], Karpovsky [323]).

   (h) **The encoding method of Brayton et al encoding method,** [455].

6. **Choice of Bound Partitions and Order of Creating Partitions.** Bound Partitions are selected in such a way that some kind of a special restricted structure is created, such as:

   (a) trees, tandem structures and other limited decomposition structures (Butler [116, 114], Butler and Breeding [118], and Bender [54]).

   (b) totally and partially symmetrical variables are used in predecessor blocks [117, 175].

   (c) trees of non-repeated variables [548, 152, 153].

   (d) cascade structures [406, 667, 114].

   (e) iterative networks (such as a comparator of two numbers of an arbitrary length each).

   (f) combinational systolic structures, pipelines, FTT-like butterflies, or other computer structures.

7. **Realization of Non-decomposable blocks.** Each block evaluated as non-decomposable by ACD is then realized:

   (a) With **SOP Minimizer** such as Espresso, [92].

   (b) With **ESOP Minimizer** such as Exorcism, [622].

   (c) With a **decision diagram,** such as a BDD.

   (d) With a **combination of programs,** which would select the best solution, such as the current variant of Flash which combines Exorcism and Espresso.

(e) A **transformations of a non-decomposable block** is executed to make this block decomposable, such as Wei Wan's EXOR transformation [678], or Steinbach's EXOR transformation.

8. **Sub-function Reusing.**

   (a) **Reuse During Encoding**, this is the approach of TRADE [678] (*TRAnsformation and DEcomposition*).

   (b) **Reuse by Re-entering Sub-functions as Input Variables**, this general logic synthesis approach has not been used yet in the framework of AC decomposition.

9. **Technological constraints** - Block packing (TRADE), XBOOLE [83].

10. **Removal of largest sets of vacuous variables.** Luba [382].

11. **Dealing with multiple-output functions: Luba, TRADE, Pedram et al;** [382, 678, 351].

Below we will briefly present the history of decomposition research and the modern look at the issues which are the most important for Pattern Theory.

6 Historical Overview of the Research on Decomposition.

Below we will list papers published every calendar year, to show the dynamics of the area, as well as the historical influences among the papers. In this section, only a very brief information about each paper is given. The papers that we consider to be the most important, or that have been the most influential ones, will be presented in more detail in the next sections.

The study of the historical literature is nevertheless fruitful, since it helps us to see the most important issues in their development perspective and allows thus to find the most prospective directions of future research.

YEAR 1854. Boole published his fundamental book, [88]. He introduced there the expansion that was later on attributed incorrectly to Shannon. Thus Boole makes the beginning of the decomposition theory, both with the respect to FD, and to the orthogonal (non-singular) expansions, because the "Shannon Expansion" is the fundament of both the decomposition and the decision diagrams used in orthogonal expansions. The so-called "Shannon Expansion" is THE most fundamental formula of Boolean logic. Boole's book is referenced in all American and Russian early research on decomposition.

YEAR 1927. Zhegalkin published his famous, but not known in Western Hemisphere paper [742], where he introduced what was next called "Reed-Muller" transform, and Reed-Muller Canonical Form. Not only was this transform next used in efficient realizations of "Ashenhurst Decompositions", but it was a base to create all orthogonal transforms and decompositions. Moreover, it was also used in new kinds of decision diagrams that can be used as an efficient representation of functions in decomposing programs.

YEAR 1949. Shannon published his paper [605], which started the history of engineering-oriented logic synthesis theory outside of the USSR. This paper influenced also the early research in USSR (Gavrilov, Povarov, Yablonsky).

YEAR 1952. Ashenhurst published early versions of his most influential works [27], that defined what we understand now by the "Ashenhurst Decomposition". This year brought also another early paper on "Ashenhurst Decomposition", published by Semon, [603].

YEAR 1953. The paper by Singer was also written, [617], a member of the same Aiken's group in Harvard, where the whole research on decomposition originated in the USA. (The Bell Laboratories is another place of early research).

Although the report by T. Singer, "The decomposition chart as a theoretical aid", from Harvard Computation Labs is little known now, the introduced by him method became next very influential. This report has been therefore reprinted as an appendix in the book of Curtis, [153].

These papers constitute the classical Ashenhurst Decomposition. In his historical paper [28], Ashenhurst stated the *fundamental disjunctive decomposition theorem.* He used the so-called *"Harvard Decomposition Charts"* in it. The charts, invented at Harvard, perhaps by either Ashenhurst himself, H. Aiken or T. Singer, were used by most of the early researchers, but they are completely equivalent to collections of the more popular Karnaugh Maps, with all possible different selections of variables in their rows and columns. The individual map from the "decomposition chart" was known since the end of the XIX-th Century under the name of the Marquand Chart. In this survey the more familiar Karnaugh maps will be used for illustration. The Ashenhurst method is based on an arrangement of the Karnaugh map with rows corresponding to the variables in the free set and columns corresponding to the variables in the bound set. A simple disjunctive decomposition exists if and only if the map has at most two distinct types of patterns of 0's and 1's in columns. It can be observed that the rows in such case have also at most two types (the function, or the function and its negation) but not counting rows of 0s and rows of 1s. The general observation is that by reducing the column multiplicity one reduces often the row multiplicity as well. All observations and ideas of Ashenhurst were next developed by other researchers.

YEAR 1954. The following papers have been written: [517], [527], [451]. The paper by G.N. Povarov in the Soviet Union originated a controversy who really invented the decomposition. It is quite possible that the Boolean Decomposition has been invented not by Ashenhurst but by Povarov, who in 1953 published (in Russian) a paper "About Functional Separability of Boolean Functions". He formulated there what is popularly known as "fundamental Ashenhurst Theorem", as well as three other theorems useful in simple decompositions. In his paper he gave references to work of Shannon, Gavrilov, and Polya. The paper was recommended by Academician Keldysh on Dec.7, 1953, and published in May of 1954. The Povarov's paper has been cited since then widely in Soviet and European literature, and he was treated in the USSR as the originator of the decomposition idea.

Also this year, Reed and Muller publish separately their fundamental papers [527], [451], being not aware that more or less the same ideas have been published by Zhegalkin in 1927.

YEAR 1956. A more advanced paper of Ashenhurst was published [30], which has been next often referenced.

YEAR 1957. R.L. Ashenhurst published his widely acclaimed paper "The decomposition of switching functions", in Proceedings of International Symposium of Theory of Switching [31]. A paper of Huffman is also published [297], which proved influential in logic design and is often referenced by early decomposition papers.

YEAR 1958. The first in the series of important papers by Curtis is published: [150]. This paper started what is now known as Curtis Decomposition.

The Curtis Decomposition is a generalization of the Ashenhurst method. However, although Curtis deals with a more general type of a decomposition, the general problem representation and the plan of looking for solutions (going through all possible partitions of input variables) are in his method still the same as in the Ashenhurst papers. Therefore, we will call this family of decompositions the Ashenhurst-Curtis, or for short, the AC decomposition.

Curtis [150] extended Ashenhurst's results to multiple decomposition when F is expressed as
$F = H(A, G_1(B), G_2(B), ..., G_k(B))$.

The use of decomposition charts (or K-maps) is of course limited to only small number of variables. Therefore, very early, researchers concentrated on the representation issue in order to allow using computers to decompose larger functions. This was done especially by the IBM group under Karp and Roth.

YEAR 1959. S.B. Akers published a paper [15], "On a theory of Boolean functions", where he introduced the concept of a Boolean Difference, one of the most important concepts in the switching circuit theory, that has been next used much in conjunction with Boolean decomposition. This paper started one approach to decomposition that was next developed mostly in Europe: Belgium (Davio, Deschamps, Thayse), USSR (Gavrilov, Zakrevskij, Bibilo), and Germany (Bochmannn, Posthoff, Dresig, Baitinger, Steinbach).

The following important papers in the mainstream of "Ashenhurst-Curtis Decomposition" have been also published: [32], [151].

YEAR 1960. J.P. Roth from IBM publishes "Minimization over Boolean Trees," [548], a paper that started the sub-area of the Ashenhurst decomposition, called the "Roth-Karp Decomposition". This is the decomposition that is most extensively investigated in the USA - all recent papers from U.C. Berkeley refer to Roth-Karp rather than to other types of decomposition, since it was considered to be the most efficient one.

Instead of looking for compatible columns as in the AC decomposition, Roth and Karp proposed to extract common sub-functions. In their seminal paper [548], they presented an algorithm for identifying a common sub-function between two functions, based on the partitioning to compatible classes. This approach has two disadvantages; first it does not apply to more than two functions; and second, it does not identify more than one shared sub-function.

In a very important step, Roth and Karp proposed a new data representation for the problem, by using the cube covers of sets ON and OFF. This was one of the first ideas and applications of cube calculus. Let us observe that this early representation, or its close derivatives, are still being used in some of the most successful programs for decomposition. Cube represenation is compact, and it is also especially well suited for strongly unspecified functions of many variables [70, 637, 638, 678].

The Roth/Karp algorithm is based on the computation of compatible classes. Let $F(x, y)$ be a Boolean function with a bound set B and a free set A, and $\mathrm{CARD}(B) = m$ and $\mathrm{CARD}(A_1) = n$, where $\mathrm{CARD}(S)$ denotes cardinality of set S. Let $B_b = B^i$ and $B_f = B^{n-i}$ where $B = \{0,1\}$. Two variables x_1 and x_2 from B_b are said to be compatible exactly, if for each y from B_f, $F(x_1, y) = F(x_2, y)$.

Roth and Karp showed that a function has simple disjunctive decomposition with respect to given bound and free sets if and only if B_f can be partitioned into $k \leq 2$ classes consisting of mutually compatible elements.

When a function is completely specified, compatibility is an equivalence relation, and k is simply the number of equivalence classes.

The relation between a *distinct column* of a K-map and a compatible class is *bijective.* The basic difference between the AC and Roth-Karp methods is in the use of different function representations. One uses K-map to represent the function, while the other uses covers of the ON-set and OFF-set of the function. When functions are incompletely specified, the detection of decomposability becomes quite complicated. For example, the compatibility in the Roth-Karp method is no longer an equivalence relation. The determination of k compatible classes requires then solving the minimum clique covering problem for the compatibility graph, which problem is known to be NP-hard.

YEAR 1961. Further papers refine the decomposition approaches of both the Aiken's Lab, [152], and the IBM group [317, 549].

YEAR 1962. H. A. Curtis published a book, [153] - "A New Approach to the Design of Switching Circuits", where he formulated a generalized theory. The book is still one of the basic references to what is called in the Western Hemisphere by the "Ashenhurst-Curtis Theory of Decomposition".

The IBM group continues their line of breakthrough papers: [549], [550].

K.K. Maitra introduces for the first time some kind of a restricted decomposed network - a cascade that was next to become known as the "Maitra Cascade".

YEAR 1963. Book by Maley [408] included early ideas on NOR decomposition, mainly intuitive graphical methods of "inhibiting" functions by other functions. These ideas have been, however, very important, since they laid a groundwork for further research of Darringer, Stoffers, Dietmeyer, and Duley [186] on applying decompositional methods to multi-level circuit design.

Further important work of Karp: [318].

YEAR 1964. The earliest Russian paper other than Povarov that we were able to find is the one by Tumanyan, [672]. We have not yet, however, had a chance to look in full detail to the very early literature from the USSR and Eastern Europe.

YEAR 1965. The following related papers have been published in USA: [16], [17], [73], [519], [679].

The papers published in USSR are: [673], [705], [703], [704].

The paper of Akers [16] is the predecessor of the Decision Diagrams approach that took over the field of logic synthesis in the 90-ties. Interestingly, the *alphabeta* diagrams invented by Akers allow to solve graphically, and for few variables, many problems that remain still of big interest nowadays. The diagrams are represented as successive partitioning of the Karnaugh-like map into smaller and smaller rectangles. Akers points out to the importance of multi-level design; the visual interpretation of such circuit constrains as fan-in, fan-out, levels of logic, decomposition and two-level forms; the reduction to smaller diagrams by Shannon expansion; the dependence on the order of variables; and the merging of diagrams to larger diagrams (compositional synthesis). Another advantage of this approach is the ease with which these constraints may be rephrased in terms of the diagrams. The explanation of Ashenhurst-Curtis decomposition using the *alphabeta* diagrams can be made quite easily. Given the diagram of a function F, select a smaller rectangle in F and remove it as a separate diagram G. Label the resulting empty rectangle in F with G and call this diagram H. Now if G and H are synthesized separately, a realization of F may be obtained by simply connecting the output of G to the input of H. This approach allows to easily find patterns of EXOR, AND, majority or other simple functions in the map. Even now, after 30 years, the Aker's paper can be very useful as a heuristic representational and teaching aid. This is also a one more way of linking Karnaugh maps and decision diagrams to the concepts of decomposition and composition of Boolean functions.

YEAR 1967. P. Deschizeaux in Grenoble, France defended one of the first theses related to decomposition [170], "Synthese de fonction Booleennes generales".

YEAR 1968. R.E. Prather published one of the first, if not the first, general logic synthesis textbooks that included the Ashenhurst decomposition [520]: "Introduction to Switching Theory: A Mathematical Approach".

The following papers have been published: [42], [45], [48], [78], [186], [261], [424], [520], [596], [564], [659].

The paper by Thelliez is perhaps the first paper ever that attempts to generalize the circuit decomposition ideas to logic with more than two values - he discussed the case of ternary logic.

YEAR 1969. The following papers have been published:

[13], [161], [294], [328], [358], [448], [509], [508], [607], [644], [645], [684], [685].

V.Y. Shen defended a Ph.D. thesis in the Department of Electrical Engineering at Princeton, where he used for the first time the idea of Reed-Muller form for the representation of Boolean function in the decomposition process in order to test bound sets more efficiently [607]. He showed thus for the first time, that the very idea of the decomposition is not related to tabular representations of Ashenhurst and Curtis, but can be expressed in any logic representation. This idea has been next carried out very successfully by other researchers.

YEAR 1970. The following papers have been published: [281], [565], [608], [677], [727].

Kellar and Shen published [608]. Their algorithm could provide a quick way to determine if the children decompose, thus restricting the number of attempted decompositions.

The article [677] dealt with functions with more than two possible output values. It was thus the first approach to general-purpose multiple-valued function minimization (logic with an arbitrary number of values).

YEAR 1971. The following papers have been published: [43] [176], [159], [249], [282], [609], [654].

Thayse [654] and Bankowski [43] are the authors of two earliest papers (independently), that attempted at purely algebraic approach to the decomposition, based on the concept of the Boolean derivative.

YEAR 1972. The following papers have been published: [123], [333].

E. Kjelkrud published a paper [333], where he considered for the first time incompletely specified functions in more detail.

YEAR 1973. The following papers have been published: [23], [74], [118], [247], [283], [511], [656], [660].

The paper by A. Thayse, and M. Davio, [656] "Boolean differential calculus and its application to switching theory," proved with time to be a very influential one.

YEAR 1974. The following papers have been published: [130] [196], [260], [368], [369].

G.V. Bochmann and W.W. Armstrong publish a paper "Properties of Boolean Functions with a Tree Decomposition".

YEAR 1975. The following papers have been published: [75], [79], [107], [108], [109], [114], [137], [229], [277].

J.T. Butler publishes first of his papers on the number of functions realized by cascades, disjunctive networks and other special kinds of decompositions.

YEAR 1976. The following papers have been published: [155], [314], [323], [432], [457], [650], [719].

H.A. Curtis published an important paper: "Simplified decomposition of Boolean functions," which influenced the line of research represented recently by Luba et al. [376].

M.G. Karpovsky [323], published the first complete exposition of the spectral approach to logic synthesis: "Finite orthogonal series in the design of digital devices." This book presented also a new approach to decomposition, and introduced the linear (EXOR) pre- and post- processor based decompositions".

A. Kandel, [314], published the first paper that used the idea of functional decomposition to Fuzzy Logic circuits.

YEAR 1977. The following papers have been published: [18], [445], [480], [640], [676].

Stoffers wrote one of the early papers in "Gate Decomposition", [640]. He depended heavily on Karnaugh maps for realizing NOR kind of decomposition.

The method of Walczak [676] decomposes Boolean functions described by Boolean expressions and hence is limited to completely specified unitary functions and is even inferior to Roth-Karp method. We are not familiar, however, with all the references from his literature.

YEAR 1978. The following papers have been published: [19], [116], [117], [162], [192], [263], [339], [356], [661].

Butler [116] used partition matrix to design "tandem networks", as opposed to cascade and disjunctive nets. This is a new special architecture devised for realizing decomposed functions. "Tandem Network" is a one more idea of a restricted decomposed structure that can be further investigated in a practical program.

Kodadapani [339] gave interesting theorems for a, related to decomposition, topic of a design with a limited fan-out constraint.

YEAR 1979. The following papers have been published [25], [127], [349], [458], [513], [512], [531], [710].

Armstrong [25] presented applications of decomposition to pattern recognition, learning, adaptation, extrapolation. This is perhaps the first paper that linked machine learning and functional decomposition, being thus the predecessor of the Pattern Theory research.

The first papers of the German school of functional decomposition arrive: [513], [512].

YEAR 1980. The following papers have been published: [44] [61], [62], [63], [64], [65], [146], [149], [195] [344], [367], [371], [667].

Crist [146] proved that for the uniformly distributed (completely specified) functions, the non-trivial decompositions are very improbable for $n \geq 5$. He gave probabilities that a function will decompose. Much is worked out when the number of variables is small.

The first papers of the Zakrevskij/Bibilo's school in the Soviet Union have been published.

YEAR 1981. The following papers have been published: [54], [80], [133], [268], [571], [664], [566], [616].

D. Bochmann, and Ch. Posthoff publish in Germany "Binaere Dynamische Systeme," one of the first textbooks containing the German school's approach to decomposition.

YEAR 1982. The following papers have been published: [1], [71], [91], [311], [731], [624].

R.K. Brayton, and C. Mc. Mullen publish "The Decomposition and Factorization of Boolean Expressions," which becomes widely referenced by MIS II literature and recent U.S. industrial papers.

A.W. Biermann et al publish "Signature Table Systems and Learning," where several links of learning and decomposition are discussed.

YEAR 1983. The following papers have been published: [67], [177], [221], [242], [298], [342], [164], [376], [479], [631], [690].

The first work of the Polish school by Luba was published, a monograph [376]. This approach uses partitions and Rough Partitions (set systems) to represent functions, and to solve all related decomposition problems.

Francioni and Kandel [221] publish first correct approach to Ashenhurst-like fuzzy logic decomposition.

YEAR 1984. The following papers have been published: [68], [69], [76], [92], [156], [158], [194], [265], [340], [467], [572], [611], [738].

T. Sasao introduces the concept of Input Variable Assignment which was next used to create classes of non-FD decompositions.

YEAR 1985. The following papers have been published: [269], [302], [375], [377], [378], [739].

S.L. Hurst, D.M. Miller, J.C. Muzio publish one of modern textbooks which emphasize the spectral approach to logic synthesis - "Spectral Techniques in Digital Logic."

YEAR 1986. The following papers have been published: [111], [245], [246], [380], [380], [463], [416], [477], [514], [461].

Luba published his second monograph book on decomposition in Polish, [382]. R.E. Bryant publishes "Graph-Based Algorithms for Boolean Function Manipulation", one of the most important papers in logic synthesis which invigorated research in decision diagrams, and thus led also to the best recent decomposers.

J.C. Muzio and T.C. Wesselkamper publish "Multiple-Valued Switching Theory", one of very few textbooks on the topic of multiple-valued logic.

YEAR 1987. The following papers have been published: [37], [39], [93], [94], [374], [381], [675], [486], [536], [575], [668], [714].

This is the first year that the increased interest in Boolean decomposition has been reflected in the number of published papers. Several groups that are active in future years published their first papers.

A book by the Belorussian group headed by Bibilo is published, [70].

R.K. Brayton and U.C. Berkeley group publish the first journal paper on MIS II system - "MIS - a Multiple-Level Logic Minimization System."

T. Sasao published "Functional Decomposition of PLAs" where the ideas of PLA decomposition were first discussed.

Perkowski et al [486] is the full version of Perkowski/Brown 1988 paper [488], with several other extensions that were not included in [488]. It included the generalization to multi-output functions, the non-disjoint decompositions, the decompositions with one-hot encodings, the methods to minimize multiplicity index, the extension to multiple-valued input cubes, and the extension to the multiple-valued logic. Generalized don't cares and "multiple-valued relations" have been introduced here for the first time for multiple-valued logic. This paper introduced also the PUB decomposition. Some of this material has been next used in Wei Wan/Perkowski's 1992 paper, [678], but the original version has been never published. A renewed interest in decomposition causes that a modified version of this paper has been included in our recent report [501].

YEAR 1988. The following papers have been published [46], [171], [180], [211], [382], [409], [417], [418], [446], [488], [515], [535], [533], [534], [632].

Paper [211] is a good reference to modular and multiple-valued decompositions.

Perkowski and Brown explained the PUB and Ashenhurst/Curtis decompositions in a simple and unified way, using the concepts of synthesis with multiplexers. A new graphical decomposition method is shown that uses Karnaugh maps instead of Ashenhurst's decomposition maps. Next, a cube-based method for strongly unspecified functions is given. The column minimization problem has been reduced here to a graph coloring problem for the first time in a published paper (this was perhaps also done in one of early Bibilo that we have no access to).

YEAR 1989. The following papers have been published: [174], [181], [189], [218], [276], [303], [357], [359], [523], [577], [633], [634], [701], [682], [721], [741], [743].

S. Devadas et al present an early approach to Boolean decomposition from U.C. Berkeley "Boolean Decomposition in Multilevel Logic Optimization." T.T. Hwang et al present a new EXOR-like decomposition in a paper "Multi-Level Logic Synthesis Using Communication Complexity."

B.G. Kim defends his Ph.D. Dissertation "Multilevel Logic Synthesis Using Extended Array Implementation" at the University of Wisconsin-Madison which includes the ideas of using symmetry in the decomposition.

D. Varma and E.A. Trachtenberg publish a new spectral approach to decomposition in a paper "Design Automation Tools for Efficient Implementation of Logic Functions by Decomposition."

YEAR 1990. The following papers have been published: [3], [4], [96], [110], [85], [129], [182], [230], [235], [243], [266], [304], [321], [335], [407], [167], [452], [465], [560], [578], [582], [583], [635].

The first paper of R. Murgai et al on "Logic Synthesis for Programmable Gate Arrays" has been published, where U.C. Berkeley's approach to lookup-table based FPGAs has been presented.

YEAR 1991. The following papers have been published: [82], [83], [113], [183], [213], [217], [223], [224], [231], [233], [286], [325], [330], [332], [360], [385], [383], [384], [435], [441], [444], [453], [481], [516], [537], [593], [614], [620], [647], [692], [700], [702], [724], [717].

Varma and Trachtenberg [702] discussed the problem of evaluating the complexity of a Boolean function, which is very important in decomposition. As observed already by Yablonsky in 1959, [718], an accurate evaluation of complexity may be as expensive as minimization of the function with the respective method. There exist functional decomposition procedures by Karpovsky 1977 [324] that use abstract complexity criteria based on functional approximations to logic complexity, and that arrive at optimal solutions by analytical methods.

An important paper by Yang and Ciesielski [724] was devoted to a Boolean decomposition of PLAs. Although Ashenhurst/Curtis work was never mentioned here, the paper introduced a powerful method to find variables' partitioning that is not exhaustive. This method has been applied here to PLA decomposition but it can be adapted to general functional decomposition as well. Especially, the presented by them ideas may be useful to find best encodings during the decomposition process.

A paper of Berkeley group by Murgai, Nishizaki, Shenoy, Brayton, and Sangiovanni-Vincentelli, "Logic Synthesis for Programmable Gate Arrays," signified an increased interest of this group in Boolean methods of this group (while the group has been predominantly occupied with algebraic methods earlier). This was the second of their several important papers, that have been mostly devoted to Xilinx's Look-up Table model.

A book with an original approach to decomposition was published in Germany - D. Bochmann, B. Steinbach, "Logikentwurf mit XBOOLE."

YEAR 1992. The following papers have been published: [38], [58], [59], [84], [141], [184], [185], [267], [305], [312], [326], [350], [354], [355], [361], [386], [387], [388], [389], [390], [391], [423], [468], [483], [494], [538], [539], [585] [598], [636], [649], [678], [695].

Y.T. Lai, M. Pedram and S. Vrudhula in their report "BDD-based Logic Decomposition: Theory" presented a new approach to Curtis decomposition of multi-output incompletely specified functions based on Binary Decision Diagrams.

W. Wan and M.A. Perkowski published "A New Approach to the Decomposition of Incompletely Specified Multi-Output Function Based on Graph Coloring and Local Transformations and Its Application to FPGA Mapping" where several new heuristics for variable partitioning, encoding and column compaction were introduced, as well as EXOR decomposition of Curtis-nondecomposable functions.

One of the approaches to improve the general Ashenhurst/Curtis approach to Boolean decomposition is to apply some kind of transformation to non-decomposable blocks. Since many functions are non-decomposable, Wei Wan and Perkowski developed an approach to Ashenhurst/Curtis decomposition, that is particularly useful for new Lookup-table and fine-grain FPGAs [678]. One of the design parameters of their program, TRADE, is the number of inputs to the non-decomposable block (for instance, a CLB of Xilinx). In case of Xilinx, this parameter is 4 or 5. For Algotronix or Atmel, the parameter is set to 2, which produces two-input gates, in practice, AND, OR and EXOR gates. In TRADE, several carefully designed heuristic techniques are used to obtain high quality solutions without actually searching too large a space of candidate partitions, column groupings or column encodings. Firstly, instead of searching all possible bound sets, a cube-based heuristic to generate a limited sequence of "good" bound sets is used. Secondly, for an incompletely specified functions, a fast graph-coloring algorithm to find a minimum partitioning of columns to disjoint sets of compatible columns is applied. (Because of the constraint of 4 or 5 input CLBs, the graph for coloring is small and has not more than 32 nodes). Thirdly, the compatible columns are encoded with a fast encoding algorithm that sorts columns according to their "Hamming distance" similarities. Next, the existing CLB output functions are re-used in encodings of new symbolic functions, which decreases the total number of newly created binary functions. Finally, non-decomposable blocks are recursively transformed to EXORs of smaller functions on which the standard Curtis decomposition attempts are repeated. This approach leads to high quality decompositions on Xilinx and other FPGAs, and can be also used to create multi-level DAG circuits of AND, OR, EXOR and NOT gates (or equivalently, OR, AND, NOT and EQUIVALENCE gates). TRADE is superior on many examples (especially symmetric functions) to other FPGA minimizers.

TRADE tries to construct the network in such a way that:

- the decomposed network is technology feasible (for Xilinx architecture, the input variables of each node is up to five),

- the number of nodes in the network is as small as possible,
- the connections between the nodes are as simple as possible, and
- the path from the input to output, which is measured by the number of CLB layers, is as short as possible.

This approach generates circuits which fit better to Xilinx technology and have less CLBs, less connections and less layers. Thus the circuit is faster and easier to place and route.

The presented methods have the following assets:

- The input data to the program is an incompletely specified Boolean function described by the sets of ON and OFF cubes. It is the property of this method that the more DC cubes exist, the more efficient the method becomes. This makes our approach particularly powerful for strongly unspecified Boolean functions.
- The decomposition methods are specifically adapted to the lookup-table based FPGA architectures.
- A fast variable partitioning method is used to quickly find the good quality partitions for decomposition, avoiding the thorough test of all possible decomposition charts.
- In order to simplify the decomposed blocks, the column multiplicity minimization and the quasi-optimum don't care assignment are performed, they are achieved through a fast graph coloring algorithm. A bound set encoding algorithm is used to further simplify the decomposed blocks.
- A local transformation method is used to make the decomposition possible for all Boolean functions.
- A CLB reusing algorithm is used to decrease the number of CLBs used in the final mapped circuit.

YEAR 1993. The following papers have been published: [7], [21], [35], [36], [55], [98], [226], [285], [351], [392], [393], [394], [395], [396], [469], [482], [496], [532], [541], [542], [543], [558], [559], [563], [568], [579], [580], [601], [590], [637].

As evidenced by the amount of papers, in the last few years the increase of interest in Boolean Decomposition has been very significant.

A paper by Y.T. Lai, M. Pedram and S. Sastry, "BDD-based Decomposition of Logic Functions With Application to FPGA Synthesis," presents what is currently the published binary decomposer that is able to decompose the largest functions.

YEAR 1994. The following papers have been published: [99], [179], [287], [397], [470], [544], [545], [546], [638], [602].

T. Luba, and R. Lasocki in their paper "Decomposition of Multiple-valued Boolean Functions" introduce what is now perhaps the most efficient decomposer for multiple-valued functions. It is especially efficient for strongly incompletely specified functions. It seems that this is the best multiple-valued minimizer available, but not enough sufficient documentation is available.

B. Steinbach and M. Stoeckert in paper "Design of Fully Testable Circuits by Functional Decomposition and Implicit Test Pattern Generation" present the numerical results of Ternary-Vector-List based (a kind of cube calculus) universal decomposer, and its applications to testable circuit design.

Selvaraj in his Ph.D. Thesis defended at the Technical University of Warsaw introduced a program that combines parallel and serial decompositions.

7 Modern Explanation of Ashenhurst-Curtis Decomposition.

In this section we will present the fundamental Ashenhurst-Curtis Approach, and illustrate it with various function representations in order to be able to compare all the existing approaches to decomposition.

For instance, an incompletely specified binary single-output function can be represented as a table with don't cares in some of cells, as an array of cubes, as a ternary BDD (a ternary BDD is a BDD with three types of terminal nodes; 0, 1 and X (representing a don't care)), or as a pair of standard BDDs, one for ON and one for OFF set.

Similarly, a multi-output incompletely specified binary function can be represented as a multi-output table with don't cares, as a multi-output array of cubes, as a shared ternary BDD, or as a shared BDD of ON-OFF pairs of standard BDDs for each component function.

For Machine Learning applications, which have a large number of don't cares (DC), the best representation should deal with function ON and OFF sets represented in some way (cubes, BDDs, etc.).

7.1 Basic Data Formats and Definitions

Suppose that one intends to decompose an incompletely specified function consisting of twenty-five inputs and twenty outputs into several smaller logic blocks. The function is given in Espresso format:

```
.i  25
.o  20
.ilb  i1 i2 i3 i4 i5 i6 i7 i8 i9 i10 i11 i12 i13 i14 i15
i16 i17 i18 i19 i20 i21 i22 i23 i24 i25
.ob  o1 o2 o3 o4 o5 o6 o7 o8 o9 o10 o11 o12 o13 o14 o15
o16 o17 o18 o19 o20
.type  fr
10-01-010101-01-01010-10-   10-10010-1010-01-10-
1-11-111-1--1100000-01-1-   01-01-00101010-----1
00000001-01010101-11-0110   01-0-1-010101-1101-0
..
..
00100101010101010-------1   1-110101001---010101
.end
```

Espresso Format is a two-level description of a Boolean function. It is a character matrix with keywords embedded in the input to specify the size of the matrix and the logical format of the input function. In the above file:

```
.i 25
```

specifies the number of input variables (25).

```
.o 20
```

specifies the number of function outputs (20).

```
.ilb i1 i2 ...... i25
```

specifies the names of input variables.

The left matrix of the above file corresponds to the input cube array, `i1` is the name of the input variable corresponding to the first column of the input cube array, `i2` to the second column, and so forth.

```
.ob o1 o2 ...... o20
```

specifies the names of function outputs.

The right matrix of the above file corresponds to the output cube array, `o1` is the name of the output variable corresponding to the first column of the output cube array, `o2` to the second column, and so forth.

o1 is the name of the function output corresponding to the first column of the output array (right matrix of the above file), o2 to the second column, and so forth.

```
.type  fr
```

sets the logical interpretation of the character matrix of output array. Symbol `fr` specifies that a 1 in the output array means that the corresponding cube in the input cube array belongs to the ON set. A 0 in the output array means that the corresponding cube in the input cube array belongs to the OFF set. The symbol '-' or ' " in the output array means that the corresponding cube in the input cube has no meaning for the value of this function. DC set may be computed as the complement of the union of the ON set and the OFF set.

```
.end
```

marks the end of the input logic.

With respect to the algorithms used in the program, DC cubes are not needed for the output function minimization. This decreases the memory demand and makes algorithm much more efficient (especially when the number of DC cubes is large).

Fundamental Definitions

A *Cube* is a compact expression of a set of minterms. For example, minterms 11010 and 11000 can be expressed as a cube 110-0. '-' means it takes the value of both 0 and 1.
If the output of a cube is 1, it is called the ON cube.
If the output of a cube is 0, it is called the OFF cube.
If the output of a cube is - (don't care. It can be either 0 or 1), it is called the DC cube.

The ON set is the collection of all ON cubes. The OFF set is the collection of all OFF cubes. The DC set is the collection of all DC cubes. Cube, ON cube, OFF cube, DC cube, ON set, OFF set and DC set are showed in Figure 1.

The *Cube Calculus* is a set of operations applied to cubes and arrays (lists) of cubes. In our description, we will use the Intersection operation, which can be illustrated using maps, arrays of cubes, or BDDs.
Figure 2 shows the rules of Intersection operation. The ϵ is the result of the Intersection of 0 with 1 or 1 with 0. x or X have the same meaning as the "-".

Cube
↓

ON	cube →	10110-	1		
		1–000	1	← ON	set
		111010	1		
OFF	cube →	0-1001	0		
		01101-	0	← OFF	set
		0-1001	0		
DC	cube →	01010-	-		
		0100-1	-	← DC	set
		–0101	-		

Figure 1: Cubes and cube sets

From Figure 2, the rules are:
$0 \cap 0 = 0$,
$0 \cap 1 = \epsilon$,
$0 \cap \mathrm{x} = 0$,
$1 \cap 0 = \epsilon$,
$1 \cap 1 = 1$,
$1 \cap \mathrm{x} = 1$,
$\mathrm{x} \cap 0 = 0$,
$\mathrm{x} \cap 1 = 1$,
and $\mathrm{x} \cap \mathrm{x} = \mathrm{x}$.

Figure 3 shows the general decomposition scheme. Boolean decomposition uses Boolean representation.

For example, a bit-by-bit intersection on cubes can be illustrated as follows:
1010xx
100x1x

10ϵ01x

The *Decomposition Chart* [617, 27, 153] is a chart that is similar to the Karnaugh map with the only difference being that the column and row indexes of the decomposition chart are in the straight binary order, while that of the Karnaugh map are in the Gray code order. Figure 4b shows an example of a decomposition chart.

The corresponding Karnaugh map is shown in Figure 4a. The column of the chart is denoted as a vector of its successive minterms. For example, column 1 in Figure 4b is denoted as a vector [1, 1, 0, 1]. Because there is no essential difference between the Karnaugh map and the decomposition chart, Karnaugh maps will be used instead of decomposition charts for illustration.

The *Bound Set* is a set of variables forming the columns of the decomposition chart. In Figure 4b, { c, d, e } is a bound set.

The *Free Set* is a set of variables forming the rows of the decomposition chart. In Figure 4b, { a, b } is a free set.

●	0	1	x
0	0	ε	0
1	ε	1	1
x	0	1	x

Figure 2: Intersection operation

The *Column Multiplicity* denoted by $\mu(A \mid B)$, is the number of different columns in a decomposition chart. In $\mu(A \mid B), A$ stands for the free set, B stands for the bound set. For example, in Figure 4b, $A = \{a, b\}, B = \{c, d, e\}$ and $\mu(A \mid B) = \mu(ab \mid cde) = 3$.

If two horizontally corresponding cells in two columns of the decomposition chart are (0,0), (1,1), (0,x), (1,x), (x,0), (x,1) or (x,x), these two cells are called *compatible*. If all the corresponding cells in two columns are compatible, these two columns are called *compatible*. Otherwise, they are called *incompatible*. In Figure 4b columns 1 ([1, 1, 0, 1]) and 6 ([1, x, 0, x]) are compatible, while columns 5 ([0, 1, 1, x]) and 6 ([1, x, 0, x]) are incompatible. In this explanation, we will use maps, Cube Calculus, or BDDs, to test whether two columns (called also *column functions*, *column cofactors* or *K-map loops*) are compatible or not. The formula [488] to test the compatibility of two columns (columns i and j) is:

[ON(i) $\cap$ OFF(j) = 0] $\wedge$ [ON(j) $\cap$ OFF(i) = 0] $\Rightarrow$ i-th and j-th column are compatible

The 0 stands for a zero function (0 Boolean constant). The formula states that if the Intersection of the ON function of column i and the OFF function of column j is zero, and the Intersection of the ON function of column j and the OFF function of column i is zero as well, these two columns are compatible. Otherwise, they are incompatible. Let us observe, that this condition does not specify

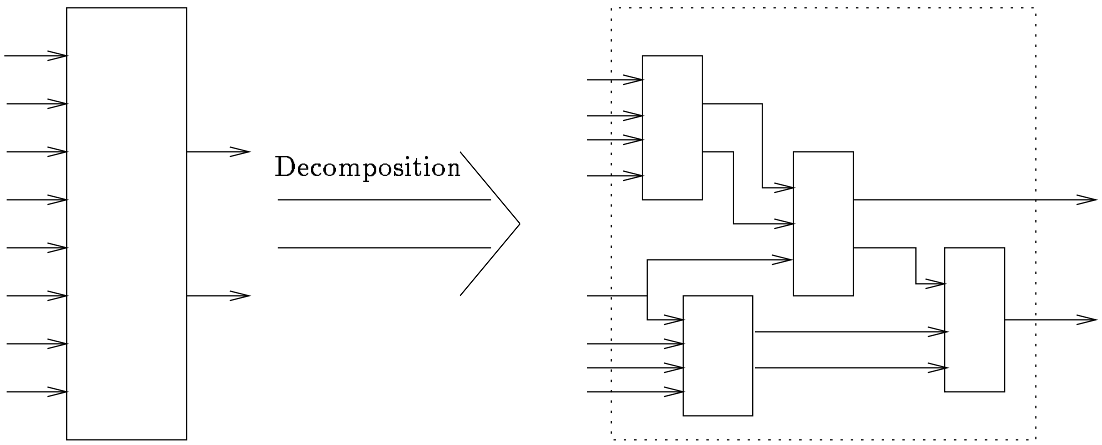


Figure 3: Decomposition

cde / ab

ab \ cde	000	001	011	010	110	111	101	100
00	x	1	1	0	1	0	0	1
01	x	1	x	1	x	x	1	1
11	x	1	x	x	x	x	x	0
10	x	0	0	1	0	1	1	0
Column	0	1	3	2	6	7	5	4

f

(a)

ab \ cde	000	001	010	011	100	101	110	111
00	x	1	0	1	1	0	1	0
01	x	1	1	x	1	1	x	x
10	x	0	1	0	0	1	0	1
11	x	1	x	x	0	x	x	x
Column	0	1	2	3	4	5	6	7

f

(b)

Figure 4: Karnaugh map and decomposition chart

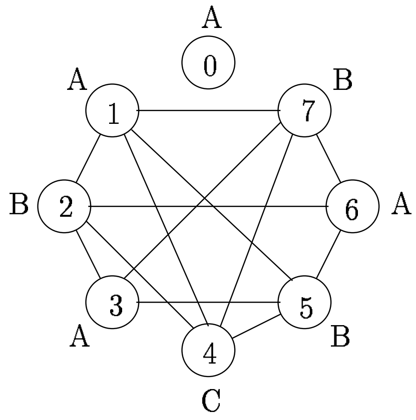


Figure 5: Example of an incompatibility graph

how functions ON and OFF are realized. For instance, ON and OFF are represented as cube arrays in TRADE.

The *Incompatibility Graph* is a graph which illustrates the incompatibility relationship among columns of the decomposition chart. Each node in the incompatibility graph corresponds to a column in the decomposition chart. If two columns are incompatible in the Incompatibility Graph, there is an edge between the corresponding nodes. If they are compatible, there is no edge.
Figure 5 shows an Incompatibility Graph corresponding to the decomposition chart in Figure 4b.

The *Compatibility Graph* is a graph which illustrates the compatibility relationship among columns of the decomposition chart. Because two columns can be either compatible or incompatible, the compatibility graph and the incompatibility graph are mutual complements. It means that the sets of *edges* of these graphs are disjoint and the union of the sets of edges creates a full graph. The column minimization problem has been reduced in the past to one of the following:

- incompatibility graph coloring,
- incompatibility graph maximum independent set partitioning.
- compatibility graph maximum clique partitioning.
- compatibility graph maximum clique covering.

All these problems are mathematically equivalent, but can be solved with heuristic approaches that will perform better or worse on particular categories of graphs (sparse, dense).
In Figure 5, the number in each node (denoted by a circle) is the column number. The letter beside the circle is the color assigned to the node (column) after graph coloring.

7.2 Decomposition of the Incompletely Specified Functions.

In this section, the functional Boolean decomposition of incompletely specified functions will be presented. The basic ideas follow [27, 153, 154, 549] and the general approach based on graph coloring is patterned after [488, 678, 464].
Curtis has described the decomposition of completely specified functions in [154]. He proved the *fundamental theorem*:

$$\mu(A|B) \leq 2^k \Leftrightarrow f(B, A) = F(\phi_1(B), \phi_2(B), \ldots, \phi_k(B), A)$$

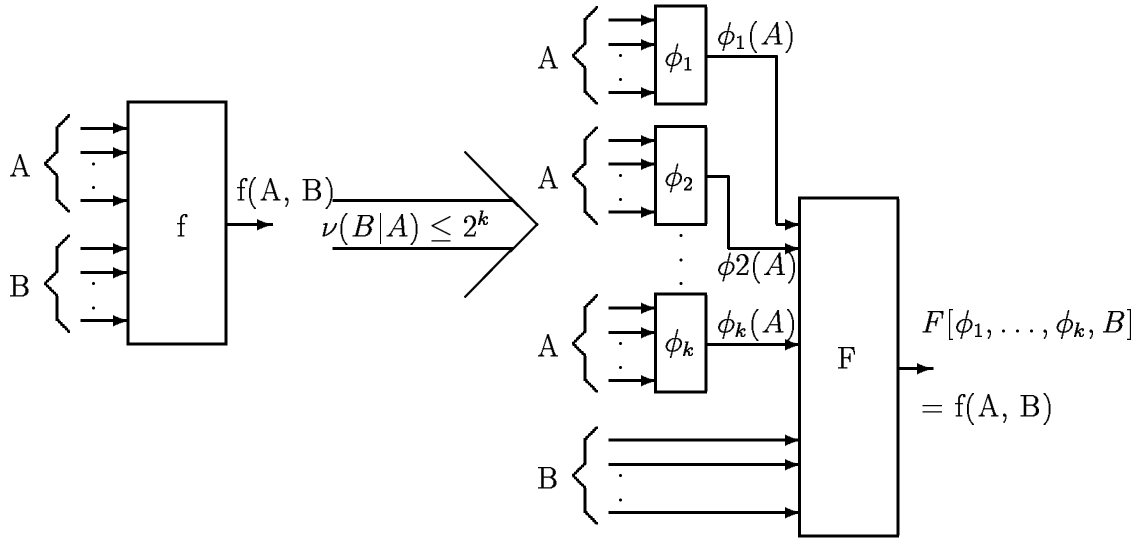


Figure 6: Curtis decomposition

This formula states that if the column multiplicity $\mu(A \mid B)$ (under the partition of the bound set B and free set A) is less than 2^k, then the function $f(B, A)$ can be decomposed into the form:

$$f(B, A) = F(\phi_1(B), \phi_2(B), \ldots, \phi_k(B), A)$$

The graphical representation of this theorem is shown in Figure 6.

From Figure 6 we observe that, after decomposition, the big block f is broken into several smaller sub-blocks ϕ_1, ϕ_2, ..., ϕ_k and F.

The essential problem of the decomposition of incompletely specified function is how to assign DC outputs as 0 or 1 to minimize the column multiplicity. Because the number of colors in a properly colored incompatibility graph is the same as the number of different columns (column multiplicity) in a decomposition chart [488], the problem of finding the smallest column multiplicity can be transformed into that of performing the proper graph coloring to find the smallest number of colors. We use the following criterion:

Assume n to be the expected number of output variables from the blocks with bound set as inputs, and n be less than the number of variables in the bound set. If the column multiplicity is equal to or less than 2^k, and k is less than or equal to n, the decomposition is *successful* (or the function is *decomposable*) for this bound set under the expected value of n. Otherwise, the function is *non-decomposable* for this bound set under the expected value of n.

After a successful decomposition, the number of input variables of each sub-function (decomposed blocks, like ϕ_1, ϕ_2, ..., ϕ_k *and* F in Figure 6) is decreased, and the complexity of each sub-function is decreased as well. This will be illustrated with an example.

Figure 7a is the Karnaugh map of function f with don't care outputs. For instance, one may intend to decompose the function f into several sub-functions (denoted by L, M and N in Figure 7c) with the input variables of each sub-function less than or equal to four.

According to the rules presented above, the incompatibility graph is created as shown in Figure 8.

After graph coloring, three colors, A, B, and C, are obtained. Which means - $\mu = 3$. These colors group nodes as A = {0, 1, 3, 6}, B = {2, 5, 7} and C = {4}. The columns with the same color are combined horizontally by the rules: (0, 0) $\rightarrow$ 0, (0, x) $\rightarrow$ 0, (x, 0) $\rightarrow$ 0, (1, 1) $\rightarrow$ 1, (1, x) $\rightarrow$ 1, (x, 1) $\rightarrow$ 1 and (x, x) $\rightarrow$ x.

ab \ cde	000	001	011	010	110	111	101	100
00	x	1	1	0	1	0	0	1
01	x	1	x	1	x	x	1	1
11	x	1	x	x	x	x	x	0
10	x	0	0	1	0	1	1	0
Column	0	1	3	2	6	7	5	4

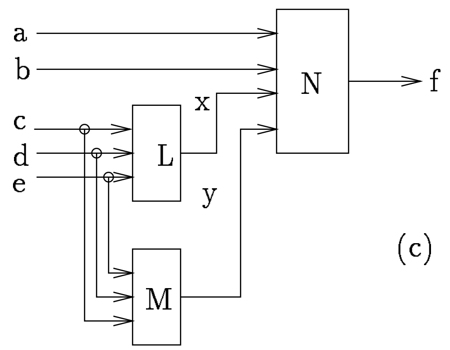
f

(a)

ab \ cde	000	001	010	011	100	101	110	111
00	x	1	0	1	1	0	1	0
01	x	1	1	x	1	1	x	x
10	x	0	1	0	0	1	0	1
11	x	1	x	x	0	x	x	x
Column	0	1	2	3	4	5	6	7

f

(b)



(c)

Figure 7: Karnaugh map, decomposition chart and the expected decomposition

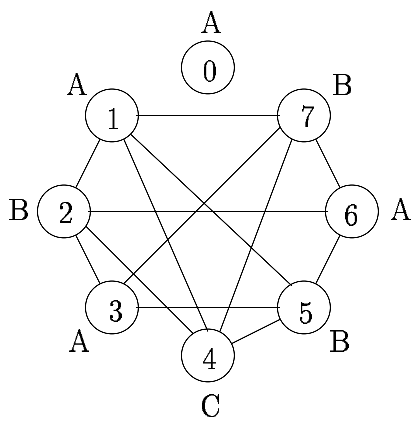


Figure 8: Incompatibility graph

ab \ cde	000	001	011	010	110	111	101	100
00	1	1	1	0	1	0	0	1
01	1	1	1	1	1	1	1	1
11	1	1	1	x	1	x	x	0
10	0	0	0	1	0	1	1	0
Column	0	1	3	2	6	7	5	4
Color	A	A	A	B	A	B	B	C

f

Figure 9: Final don't care assignment

For example, columns 0, 1, 3 and 6 in Figure 7a are combined and replaced by a new vector [1, 1, 1, 0] as shown in the final don't care assignment in Figure 9. In the above example, we have chosen the variables a and b as the free set and variables c, d and e as the bound set. This partition results in a successful decomposition in the sense of the column multiplicity less than or equal to three. In Figure 7c, x and y are the encoded outputs of the bound set, two variables are enough for three different columns ($3 \leq 2^2 = 4$). The encoding of Bound set will be discussed in the next section.

There has been certain criticizm of AC model. It starts from an observation that in the AC decomposition the block F which is the result of decomposition has fewer input variables then the initial function block f. This property, however, is not true for most of the practical circuit realizations, for instance the SOP realization, which has the number of primes in the minimal cover (the second level) that is larger than the number of input variables (the first level). And there are many other practical circuits that have the same property. Because of that, the application of the AC decomposition model is questioned for general circuit design. However, this criticism does not apply to the most general non-disjoint decomposition model, where the number of blocks in intermediate levels can grow, because they use different overlapping subsets of intermediate variables.

7.3 Bound Set Encoding.

There are many methods [722, 724] to implement the decomposed blocks (blocks L, M and N in Figure 7c). Here we introduce an algorithm to encode the bound set that aims at simplifying the block N. The encoding algorithm assigns adjacent codes (Gray code) to the similar columns. This increases the number of large cubes in the block N. The similarity (or difference) between two columns is measured by the so-called *Difference Factor*. The more similar the two columns, the lower the value of the Difference Factor. The *Difference Factor* is the number of minterms in which the two columns are not identical. The Difference Factor between the i-th and j-th columns is:

$$\textit{Difference Factor} = \text{minterm\_size}(\text{ON}(i) \cap \text{OFF}(j)) + \text{minterm\_size}(\text{OFF}(i) \cap \text{ON}(j))$$

In the above formula, "minterm_size()" calculates the number of minterms. "ON(i) ∩ OFF(j)" is the Intersection of ON function of the i-th column with the OFF function of the j-th column. "OFF(i) ∩ ON(j)" is the Intersection of OFF function of the i-th column with the ON function of the j-th column. The more similar the two functions, the smaller the value of the Difference Factor. It becomes a zero for identical functions.

In this example, bound set $\{c, d, e\}$ forms eight columns as shown in Figure 7a. After graph coloring, three different columns were found. These three columns are: [1, 1, 1, 0] corresponding to color A, [0, 1, x, 1] corresponding to color B and [1, 1, 0, 0] corresponding to color C as shown in Figure 9. We

	B	C
A	A-B 2	A-C 1
	B	B-C 2

Figure 10: Similarity Factor Table

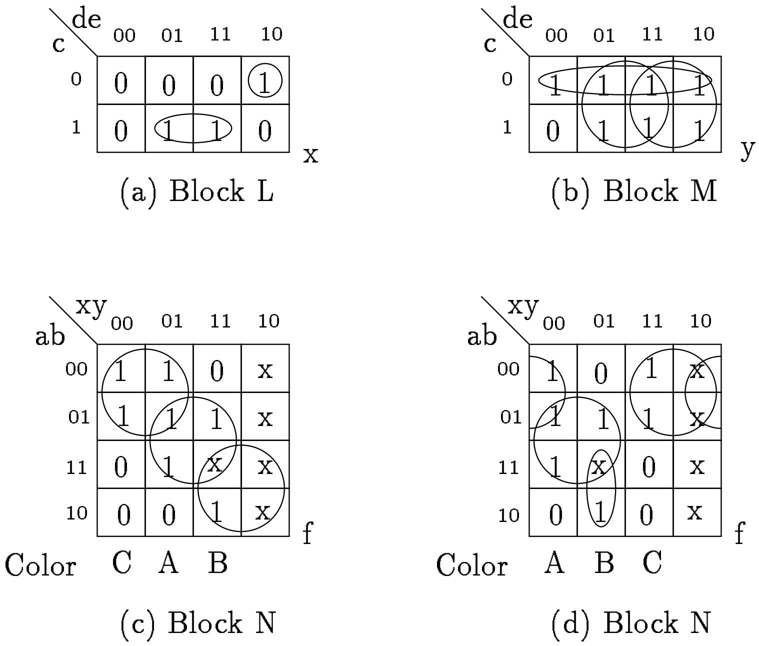


Figure 11: Decomposed Karnaugh maps

introduce two new variables x and y to encode the bound set $\{c,\ d,\ e\}$. First let us calculate the Difference Factors. The Difference Factor between columns corresponding to color A and B has a value of 2. The value of the Difference Factor between columns corresponding to color A and C is 1. And the value of the Difference Factor between columns corresponding to color B and C is 2.

A Table of Difference Factors is created as shown in Figure 10.

Because the Difference Factor between columns corresponding to color A and C is smaller (with a value of 1), these two columns are put in adjacent positions, as shown in Figure 11c. Let us code the column corresponding to color C as 00, the column corresponding to color A as 01 and the column corresponding to color B as 11 as shown in Figure 11c, which is the Karnaugh map of the block N. Color A has the code 01, which means that x is equal to 0 and y is equal to 1 for all columns with the color A. These columns are 000, 001, 011 and 110 in Figure 9, therefore the cells 000, 001, 011 and 110 of the Karnaugh map in Figure 11a, which is the Karnaugh map of the block L, are filled with 0 because x is equal to 0. The same cells in Figure 11b, which is the Karnaugh map of the block M, are filled with 1 because y is equal to 1. The same way, color B has the code 11, which means that both x and y are equal to 1. Columns 010, 111, and 101 correspond to color B, therefore the cells 010, 111 and 101 of the Karnaugh maps in both Figure 11a and (b) are filled with 1. Color C has the code 00, both x and y are equal to 0, column 100 correspond to color C, the cell 100 of the Karnaugh maps in

```
bound_set_encoding( )
{
create Similarity Factor Table;
sort Similarity Factor Table in increasing order;

l_c = one column of the column pair at position 0 of the queue;
mark l_c as used;
r_c = another column of the column pair at position 0 of the queue;
mark r_c as used;
put l_c and r_c in line; *l_c at left, r_c at right *
c_n = 2;

while (c_n < column_multiplicity)
  for (i = 1; i < column_multiplicity*(column_multiplicity - 1)/2; i++)
    if ((c_i = one of the pair at position i) == l_c)
      {
      mark c_i as used;
      put c_i at the left of l_c;
      l_c = c_i;
      c_n++;
      break;
      }
    else if ((c_i = one of the pair at position i) == r_c)
      {
      mark c_i as used;
      put c_i at the right of r_c;
      r_c = c_i;
      c_n++;
      break;
    }
}
```

Figure 12: Pseudo-code of bound set encoding

both Figure 11a and (b) are filled with 0.

Two variables can encode up to four columns ($2^2 = 4$). There are only three columns, corresponding to color A, B and C, that need to be encoded in our example. We fill the remaining column (column 10 in Figure 11c) with don't cares (*DC column*). The existence of this newly introduced DC column will further simplify the block N. This example shows that even if the input function is completely specified, the algorithm may introduce DCs in the middle of the process, which is very useful for the simplification of the later stages.

Figure 11d shows a Karnaugh map of an alternative implementation for the function f, which uses the natural order of the colors. Clearly, the Karnaugh map in Figure 11c is simpler than that in Figure 11d.

The pseudo-code for bound set encoding is shown in Figure 12. The BLIFF format of the result is as follows:

```
.model example
.inputs a b c d e
```

```
.outputs f
.names c d e x
1-1 1
010 1
.names c d e y
0-- 1
--1 1
-1- 1
.names a b x y f
0-0- 1
-1-1 1
1-1- 1
.end
```

BLIF Format is a multi-level description of the Boolean network. Each node in this representation has a single output. Therefore, each net (or signal) has only a single driver, and one can therefore name either the signal or the gate which drives the signal without ambiguity.

```
.model example 16
```

specifies the name of the model (example).

```
.inputs a b c d e
```

gives the name of the input variables (a, b, c, d, e).

```
.outputs f
```

gives the name of the output of the function (f).

```
.names c d e x
```

with the following ON set describes the logic of a node (sub-block L in Figure 7c). The input variables to this node are c, d, e, and the output variable is x.

```
.names c d e x
```

with the following ON set describes the logic of a node (sub-block M in Figure 7c). The input variables to this node are c, d, e, and the output variable is y.

```
.names a b x y f
```

with the following ON set describes the logic of a node (sub-block N in Figure 7c). The input variables to this node are a, b, x, y, and the output variable is f.

```
.end
```

marks the end of the model.

There are four fundamental problems to be solved in a Boolean decomposition of an incompletely specified function:

- How to chose the bound set to minimize the column multiplicity?
- How to minimize the column multiplicity for a given bound set ?
- How to encode the functions?
- How to transform a non-decomposable function into a decomposable one?

These questions will be discussed in more detail in next sections.

Variable Partitioning is the separation of the input variables into two sets, the bound set and the free set. Each partition corresponds to an individual decomposition chart which is going to be used to calculate the column multiplicity. In order to find the decomposition that corresponds to the smallest column multiplicity, one needs to go through all possible decomposition charts. If there are total m input variables and n variables in the bound set, the number of all possible partitions is $\binom{m}{n}$.

For example, if m = 64 and n = 5, then $\binom{64}{5} = 7,624,512$.

If the time required to calculate the column multiplicity of a decomposition chart is 0.01 second, one would need more than 20 hours to complete all calculations. This 20 hours will be repeated thousands of times to get the decomposition done. Therefore, it is impractical to try all possible partitions to find the best one.

There are basically three approaches to the Variable Partitioning problem:

- a fast method - find few good partitions using a powerful heuristic,
- perform a heuristic search for a reasonably good subset of all partitions,
- check all partitions to find the exact solution.

8 Roth-Karp Decomposition

8.1 Introduction

Roth's serial decomposition technique, [549], can be illustrated as follows. Given $F(a,b,c,d,e)$ a selection is made of free and bound variable sets (these sets are not necessarily disjoint) and tests are made for the existance of single-output or multiple-output predecessor functions. Let us assume a single-output predecessor $g_1(a,b)$, as illustrated in Figure 13.

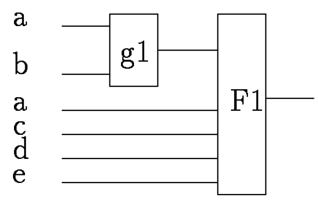


Figure 13: Serial decomposition with single output predecessor, one stage

Function $F_1(g_1(a,b),a,c,d,e)$ is the image of F and its ON-OFF sets can be computed from the ON-OFF sets of F in terms of the outputs $g_1(a,b),a,c,d,$ and e (as shown in a later example). Let us assume the simple disjoint case, i.e. $g_1(a,b)$ is a single-output function. The image of F will exist if the following conditions hold :

Given functions $F(B,A)$ and $g_1(B)$, there exists a function f such that:

$$F(B,A) \leq f(g_1(B),A) \tag{1}$$

(i.e. for each set of inputs B,A where the value of F is defined, f has the same value) if and only if:

1. input cubes b_i and b_j, are incompatible with respect to F ($F(b_i,a_k) \neq F(b_j,a_k)$), where same a_k implies $g_1(b_i) \neq g_1(b_j)$

   or equivalently,

2. $g_1(b_i) = g_1(b_j)$ implies compatible input cubes b_i and b_j, with respect to F, ($F(b_i,a_k) = F(b_j,a_k)$)

Function f is then called the image of $F(B,A)$, such that for all cubes b_i, a_j where the value of F is defined, $f(g_1(b_i),a_j)$ has the same value. Compatible cubes intersect and $\neq$ denotes different values of a scalar Boolean function.

If F_1 is still not realizable with a library component, its image $F_2(g_2(a,c),g_1,e,d)$ is formed if a single-output predecessor $g_2(a,c)$ exists.

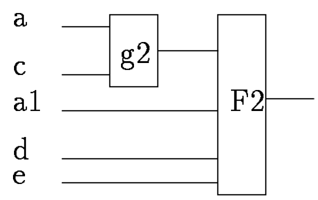


Figure 14: Stage two decomposition F_2

The ON-OFF sets of F_2 can be also derived from the sets for F_1 in terms of $g_2, g_1, e,$ and d; if F_2 is a realizable component then this is the last step of the decomposition process, otherwise the image of F_3 has to be obtained, etc.

	ON		OFF
	a b c d		a b c d
$u_1 =$	1 0 1 x	$v_1 =$	1 1 x 1
$u_2 =$	1 x 1 0	$v_2 =$	1 x 0 x
$u_3 =$	0 1 x x	$v_3 =$	x 0 0 x

Table 1: ON OFF cubes for bound set $B = \{ab\}$ and free set $A = \{cd\}$

The test for existance of a single-output predecessor is as follows: We want to test if $F(a, b, c, d)$ has an image $F_1(g_1(B), A)$ where the bound set $B = \{a, b\}$ and the free set $A = \{c, d\}$. Let us arrange the ON and OFF array of F as in Table 1.

Let's denote the set of ON cubes by $U = \{u_1, u_2, u_3\}$ and the set of OFF cubes by $V = \{v_1, v_2, v_3\}$. Any two cubes u and v, one in the ON set and the other in the OFF set, must be mapped to distinct cubes in the image of F; if u and v have common free-set parts $u - B, v - B$, then the bound-set parts $u - A, v - A$, are the only distinguishing features of u and v and $g_1(u - A) \neq g_1(v - A)$ must hold. Otherwise some cubes of the image F_1 would appear in both the ON and OFF sets of F_1.

In the above example:
$u_1 - A =$ 10, $u_2 - A =$ 1x, $u_3 - A =$ 01;
$u_1 - B =$ 1x, $u_2 - B =$ 10, $u_3 - B =$ xx;
$v_1 - A =$ 11, $v_2 - A =$ 1x, $v_3 - A =$ x0;
$v_1 - B =$ x1, $v_2 - B =$ 0x, $v_3 - B =$ 0x;
The above requirement can be also stated as follows: the bound-parts of u and v are incompatible if their free-parts intersect.

For example $u_1 - B$ and $v_1 - B$ intersect at 11; $F(1011) = 1$, $F(1111) = 0$. Thus we must have:
$F_1(g_1(10), 11) \neq F_1(g_1(11), 11)$ and this requires $g_1(10) \neq g_1(11)$.

All pairs of cubes u, v must be considered to determine if a decomposition with g_1 exists. A systematic way to do this is to list all pairs which intersect and the corresponding incompatible bound-part cubes. Figure 16 shows a graph of compatible values of predecessor $g_1(a, b)$.

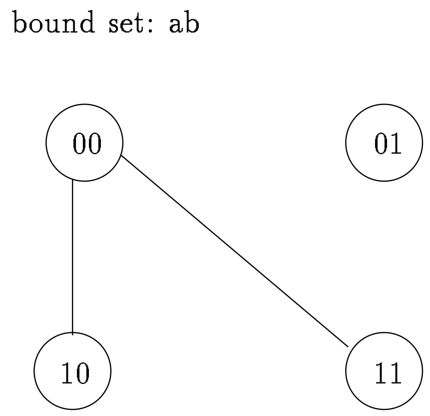


Figure 15: Compatibility Graph of $g_1(a, b)$ values

This graph can be interpreted as the following set of implications :

- $u_1 - B$ intersects $v_1 - B$ implies $g_1(10) \neq g_1(11)$

- $u_3 - B$ intersects $v_1 - B$ implies $g_1(01) \neq g_1(11)$
- $u_3 - B$ intersects $v_2 - B$ implies $g_1(01) \neq g_1(1\mathrm{x})$
- $u_3 - B$ intersects $v_3 - B$ implies $g_1(01) \neq g_1(\mathrm{x}0)$

The above Compatibility Graph from Fig. 16 cannot be partitioned into two cliques of compatible elements, therefore $g_1(a, b)$ has 3 distinct values:

$$g_1(10) \neq g_1(11) \neq g_1(01).$$

This implies that $g_1(a, b)$ can not be a single-output binary function. The Roth-Karp algorithm [549] considers a multiple-output predecessor at this point, such that the number of mutually compatible bound-part sets $\leq 2^t$ where t is the number of outputs of the predecessor: $F_1(g_1(B), g_2(B), \ldots, g_t(B), A)$.

In the above example there were three compatible bound-part sets 11, 10, and 01. Hence a decomposition can be found if a two-output predecessor block is used $3 < 2^2, t = 2$. Two-output predecessor block, as required by Roth's test for single-output/Ashenhurst predecessor above is shown in Figure 16.

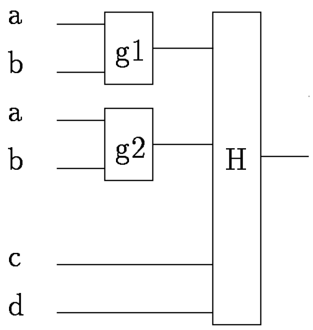


Figure 16: Two-output predecessor block in serial decomposition

Another way to state the above requirement is that the ON set of F must be identical to the ON set of H (image of F). The ON terms of F above can be mapped to the distinct values of $g_1(ab)g_2(ab)cd$ (minterms in the ON set of the image H). The truth table for F is as in Table 2.

a	b	c	d	F
1	0	1	0	1
1	0	1	1	1
1	1	1	0	1
0	1	0	0	1
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1

Table 2: Truth table for F

With a, b as the bound set variables, the decomposition chart shows that one possible mapping is ab = 10 to g_1g_2 = 00, ab = 11 to g_1g_2 = 01, and ab = 01 to g_1g_2 = 10. The truth table for H can then be filled in as in Table 3.

Each row corresponds to a row in the table for F, with variables ab mapping to g_1g_2 as specified above. This represents an arbitrary encoding of the columns in the decomposition chart of F given by Kmap from Table 4.

The relation between the input cubes of the bound set variables ab and g_1g_2 is shown in Table 5. Columns C_1, C_3 form compatible $class_1$, encoded with $g_1g_2 = 01$, C_2 is $class_2$ with $g_1g_2 = 10$, C_4 is $class_3$ with $g_1g_2 = 00$. H depends on the predecessor functions g_1 and g_2 according to:

$$H = g_1\overline{g_2} + \overline{g_1}c\overline{d} + \overline{g_2}c$$

Also $g_1(a,b) = \overline{a}b$ and $g_2(a,b) = \overline{a}\overline{b} + ab$.

The final image:
$H = \overline{a}b + (a+\overline{b})c\overline{d} + (a\overline{b} + b\overline{a})c$
The above observations have been stated formally in two theorems by Roth and Karp [549]:

Theorem 8.1 *Given $F(B,A)$ and $g_1(B)$, there exists $F_1(g_1(B),A) = F$ iff for all cubes b_i and b_j included in B, b_i incompatible (non-intersecting) with b_j implies $g_1(b_i) \neq g_1(b_j)$.*

Theorem 8.2 *If the cubes of bound set B can be partitioned into k mutually compatible sets then a decomposition with a t-output predecessor exists, where $k \leq 2^t$ and*

$$F = F_1(g_1(B), \ldots, g_t(B), A)$$

A circuit synthesis procedure was presented in Karp [318], based on a-priori knowledge of all the predecessor (library) functions $g_1(), g_2(), ..., g_k()$. This requirement can not be maintained in a general decomposition framework. Instead, any form of the predecessors $g_i()$ can be assumed and the DFC minimization strategy should rely on bound and free set evaluations for decomposition forms such as simple (column multiplicity = 2) disjoint, iterative disjoint, and complex disjoint and non-disjoint. The image of F should be constructed at each level as in the example above.

The iterative disjoint form has been defined in Karp [318], pp.300-303 as shown in Figure 17, with single output predecessor at each stage.

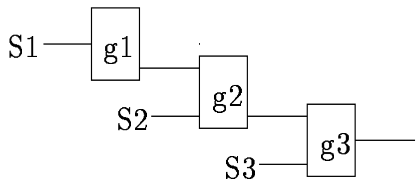


Figure 17: Iterative serial disjoint decomposition

The $S1$, $S2$, and $S3$ are disjoint sets and cover all variables of F, and all $g_i()$ functions are single output.

9 Bibilo-Yenin Decomposition.

Ideas of Ashenhurst and Curtis, and particularly Roth and Karp have been re-implemented by Bibilo and Yenin before 1987 and published in a book [70]. However, the comparison of the quality of their method with the quality of similar approaches in the West is difficult, since they did not use benchmark functions. In the book there is no critical evaluation of the proposed improvements, no references to recent Western literature, and no comparisons to Western programs. They use more efficient algorithms for partitioning and column compaction which is performed together with the encoding. This is perhaps a very good idea worth further study. It is somehow similar to the concurrent state minimization and state assignment, as proposed in [364, 124].

Bibilo and Yenin report to solve functions of 40 input variables, and they tested their programs on randomly generated functions. Since randomly generated functions are very hard, the performance of their approach seems to be good, but it is hard to assess. Some of their algorithms should be implemented and compared.

10 He and Tolkersen Decomposition.

In 1993 He and Tolkersen proposed a new Boolean extraction algorithm based on K-maps. Because of K-map size restrictions, their algorithm is not very practical as presented. However, there are certain new ideas in their paper which can be perhaps adapted to one of the modern representations of Boolean functions, such as BDDs or partitioned representations. This will be discussed in the sequel.

11 Spectral Approach of Shen and McKellar.

The main drawback of using the basic method of Disjoint Decomposition is to check $2^m - m - 2$ "maps" for an m-input function. However, it is possible to devise algorithms that use necessary conditions for the existence of a decomposition in order to prune out certain combinations of input variables which do not belong to bound sets of any decompositions. This effectively speeds up the procedure in all but pathological cases.

The search for a fast algorithm for the disjunctive decompositions of switching binary functions was studied by Shen at el [607, 608, 609]. They presented a fast algorithm based on testing a necessary condition for decomposability of switching functions. During this process, candidate bound (CB) sets were generated. The CB sets are a smaller subset of all possible bound sets. By performing an additional minor test on CB sets, Shen et al were able to determine the decomposability of a switching function in disjunctive form. Shen and McKellar [607, 608, 609] devised an algorithm that detected candidate partitions for disjoint decompositions of logic functions but required further testing of candidates, even though the set of candidates contained far fewer partitions that the original set of all possible partitions. They also showed that Reed-Muller (RM) canonical form was easier to test for decompositions than the disjunctive normal form (DNF). Thus, the idea of a speedup resulting from applying a new function representation has been further reinforced.

g_1	g_2	c	d	H
0	0	1	0	1
0	0	1	1	1
0	1	1	0	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	1

Table 3: Truth table of H, for $g_1(a,b)$, $g_2(a,b)$

cd-ab	00	01	11	10
00	0	1	0	0
01	0	1	0	0
11	-	1	0	1
10	-	1	1	1
	C_1	C_2	C_3	C_4

Table 4: Kmap

a	b	g_1	g_2
0	0	0	1
0	1	1	0
1	0	0	0
1	1	0	1

Table 5: Bound set encoding table

12 The Approach of Steinbach et al.

The approach of Steinbach et al is the result of many years of research of Bochmann, Posthoff and Dresig. This is a 'gate-type' variant of a decomposition. Two-input gates: AND, OR, EXOR, and a single-input NOT gate are assumed. The decompositions for these gates can be either disjoint or non-disjoint.

Although this is is not a Functional Decomposition and the concept of multiplicity index is not used, this decomposition is more similar to AC than any other 'gate type' decomposition from the literature. It can be also shown that their conditions for decomposability can be derived from the basic theorems of Ashenhurst.

The sources of these original ideas seem to start from the independent works of Davio and Zakrevskij. Bochmann himself was perhaps influenced by the work of Zakrevskij's group, one of the three top logic synthesis groups in the former Soviet Union. Zakrevskij was the creator of the original algebraic ideas of this line of research; such as the definitions of differential, maximum and minimum. Some of them may go back perhaps to the early papers of Davio, Thayse and Deschamps. Many of the respective early papers are still unavailable to us.

All recent ideas of this German group are implemented in programming system XBOOLE, therefore we will be referring to the *XBOOLE approach*.

There are five properties that are interesting about the approach of XBOOLE:

- The totally algebraic formulation of the decomposition problem. The definition of Boolean operators that are powerful and universal, and do not depend on the actually used representation of Boolean functions. Being purely Boolean concepts, these operators can be implemented in conjunction with any other representations of binary functions. This can be taken advantage of by reformulating these operators for other types of decompositions: multiple-valued, fuzzy, and other logic. The decomposition method in the most general sense expressed by the operators should remain the same. The symbolic operators used in XBOOLE are: complementation, cofactor, AND, OR, MIN, MAX, and EXOR. The main difficulty in generalizing these ideas to multiple valued logic is in having several complementation operators in multiple-valued logics.

- The efficient use of Ternary Vector Lists, a kind of 'Cube Calculus', in which all the cubes are disjoint. They are being made disjoint ('orthogonalized'), immediately after execution of every operation that may produce non-disjoint cubes.

- The decomposition used by XBOOLE is actually not an AC decomposition, since it decomposes each function to two-input gates: AND, OR and EXOR, and possibly also Inverters. This approach allows to decompose large functions relatively quickly. Contrary to other 'Gate Decomposition' methods, we discuss XBOOLE system here in more detail, because:

  - several of its ideas are still very close to the AC decomposition (for instance the general decomposition models or variable partitioning),
  - we believe that its methods are general and can be adapted for Machine Learning purposes.

  We will call it 'XBOOLE Decomposition'. Such decomposition can be one of 'Special Bias' Decomposition in the future decomposer proposed in this document. It is much closer to AC decomposition than other 'gate-oriented' Boolean decompositions, like those of Zaky [740], Stoffers, Dietmeyer and Duley, and Darringer [157].

- Every step in the decomposition can have a large amount of different results because of different algorithms, different basis elements, and the ambiguity of the decomposition. The program allows then to be controlled by some parameters in such a way that the result optimally meets some kind of user requirements. Thus, it can be tuned to various technological parameters, such as the area, the number of gates, the depth of the circuit, the power consumption, or the circuit's testability. An overview of different kinds of XBOOLE decompositions is given in [82, 184].

- The method can efficiently handle the don't cares.
- The method can be easily converted to either the BDDs with don't cares (ternary BDDs, an extension of a BDD that has three terminal nodes, corresponding to '0', '1', and '-') or to the pairs of standard BDDs that represents ON and OFF sets. XBOOLE uses a pair of cube represented completely specified functions q (we denote it by ON) and r (we denote it by OFF)to represent an incompletely specified function.

XBOOLE decomposition is based on the concept of 'groupability'.

Definition 12.1 *(groupability, Steinbach).*
A Boolean function $f(X_a, X_b, X_c)$ is said to be (disjunctively, conjunctively of exclusive-or) groupable with the operation $\bigcirc \in \{$ AND, OR, EXOR $\}$ and variable sets X_a, X_b if

$$f(X_a, X_b, X_c) \;=\; g(X_a, X_c) \;\bigcirc\; h(X_b, X_c) \tag{2}$$

Although not all Boolean functions are groupable, it has been proven in [359] that all Boolean functions, which are not groupable by the EXOR gate decomposition, are always disjunctively or conjunctively weakly groupable.

Definition 12.2 *(weak groupability) (Steinbach).*
A Boolean function $f(X_a, X_b)$ is said to be (disjunctively, conjunctively, or exclusive-or) weakly groupable with operation $\bigcirc \in \{$ OR, AND, EXOR $\}$ and the variables set X_a if

$$f(X_a, X_c) \;=\; g(X_a, X_c) \;\bigcirc\; h(X_c) \tag{3}$$

A weakly groupable function can be implemented by several weak groupings. For a disjunctively or conjunctively weakly groupable function $f(X_a, X_b)$ the function $h(X_c)$ can always be selected in such a way that $g(X_a, X_c)$ can be implemented more easily than $f(X_a, X_c)$, although the number of variables is not altered.

The question 'is an incompletely specified function with the functions $ON(X)$ and $OFF(X)$ for X_a and X_b groupable?' can be answered using the following existence theorems.

The theoretical basis is explained in works of Steinbach, Le, Dresig, Bochmann, and Posthoff.

Theorem 12.1 *(disjunctively groupable) (Steinbach).*
An incompletely specified function with the functions $ON(X)$ and $OFF(X)$ for X_a and X_b is disjunctively groupable, *if and only if*

$$ON(X) \wedge max_{X_a}{}^{k}\ OFF(X) \wedge max_{X_b}{}^{k}\ OFF(X) = 0 \tag{4}$$

The differential equation 4 implies that no pattern from the ON-set (represented by ON(X)) may be in the projection of the OFF-set (OFF(X)) in the X_a as well as in the X_b direction.

Dual situations exist in conjunctively groupable functions.

Theorem 12.2 *(conjunctively groupable) (Steinbach).*
An incompletely specified function with the functions $ON(X)$ and $OFF(X)$ for X_a and X_b is conjunctively groupable, *if and only if*

$$OFF(X) \wedge max_{X_a}{}^{k}\ ON(X) \wedge max_{X_b}{}^{k}\ ON(X) = 0 \tag{5}$$

The exclusive-or groupability for X_a and X_b is determined step by step. First, the derivation of the incompletely specified function for one variable x_a is calculated using Lemma 12.1.

Lemma 12.1 *(derivation of an incompletely specified function) (Steinbach).*
The derivation of an incompletely specified function with respect to variable x_a forms also an incompletely specified function with the functions

$$ON_a(X) \;=\; max_{x_a}\, ON(X) \;\wedge\; max_{x_a}\, OFF(X) \tag{6}$$

$$OFF_a(X) \;=\; min_{x_a}\, ON(X) \;\vee\; min_{x_a}\, OFF(X) \tag{7}$$

Theorem 12.3 *(exclusively-or groupable) (Steinbach).*
An incompletely specified function with the functions $ON(X)$ and $OFF(X)$ is exclusively-or groupable for one variable x_a and set of variables X_b, if and only if

$$ON_a(X) \;\wedge\; max_{X_b}{}^k\; OFF_a(X) \;=\; 0 \tag{8}$$

The test (8) for (x_{a1}, X_b) and (x_{a2}, X_b) only confirms the exclusive-or groupability for $((x_{a1}, x_{a2}), X_b)$, if there is at least one specified function within the incompletely specified function which meets the requirements for x_{a1} as well as for x_{a2}. The probability of finding such specified functions is increased by using Theorem 12.4.

Theorem 12.4 *(restrictions of an incompletely specified function) (Steinbach).*
The reduced incompletely specified function with the functions

$$ON(X) := ON(X) \vee OFF(X \setminus x_a, \overline{x_a}) max_{X_b}{}^k ON_a(X) \vee ON(X \setminus x_a, \overline{x_a}) max_{X_b}{}^k OFF_a(X) \tag{9}$$

$$OFF(X) := OFF(X) \vee ON(X \setminus x_a, \overline{x_a}) max_{X_b}{}^k ON_a(X) \vee OFF(X \setminus x_a, \overline{x_a}) max_{X_b}{}^k OFF_a(X) \tag{10}$$

contains all Boolean functions of the primary incompletely specified functions that are exclusive-or groupable for (x_a, X_b).

The exclusive-or weak groupability is a property of all Boolean functions. But only functions that satisfy theorems 12.5 or 12.6 enable the disjunctively or conjunctively weak groupings.

Theorem 12.5 *(disjunctively weakly groupable) (Steinbach).*
An incompletely specified function with the functions ON(x) and OFF(x) is disjunctively weakly groupable for X_a if and only if

$$ON(X) \;\wedge\; \overline{max_{X_a}{}^k\; OFF(X)} \;\neq\; 0 \tag{11}$$

The inequality (11) implies that not all patterns of the ON-set ($ON(X)$) may be in the projection of the OFF-set ($OFF(X)$) in the X_a-direction.

The patterns of ON(X) which are not covered by the 'zero-projection' belong to the function ON(X_c). Here, dual situations with regard to conjunctively weakly groupable functions can be also observed.

Theorem 12.6 *(conjunctively weakly groupable) (Steinbach).*
An incompletely specified function with the functions $ON(X)$ and $OFF(X)$ is **conjunctively weakly groupable** *for X_a, if and only if,*

$$OFF(X) \;\wedge\; \overline{max_{X_a}{}^k\; ON(X)} \;\neq\; 0 \tag{12}$$

If only those properties of incompletely specified functions are used which have been mentioned above, a convergent decomposition method can be developed. This is due to the following theorem.

Theorem 12.7 *(groupable) (Steinbach).*
Every Boolean function that is not disjunctively weakly groupable and not conjunctively weakly groupable has the property of exclusive-or groupability.

The algorithm is based on the idea that the more symmetrical and the larger the vectors X_a and X_b are, the better is the timing behavior of the design. The demand for symmetry is based on the fact that Boolean functions with equal number of variables produce circuits with similar numbers of gates.

A maximum size of X_a and X_b results in a smaller vector X_c, to make functions $g(X_a, X_c)$ and $h(X_b, X_c)$ being dependent on less variables.

As a criterion for the best grouping the smaller one of X_a and X_b should be as large as possible. If more than one grouping has the same number of variables in the smaller vector, the grouping with the largest vector is the best. In contrast to the area-optimizing algorithm where one vector is filled first, the numbers of variables in vectors X_a and X_b is increased alternately.

The algorithm uses a heuristic approach with certain rules when to add or subtract a variable, and which one, from sets X_a and X_b. In this sense it is very different from all other variable partitioning methods from the literature.

The largest examples presented in the paper are of a medium size: misex2, has 25 inputs and 18 outputs, duke2 has 22 inputs and 29 outputs. It is then not known, what are the practical limits of the method. The functions are smaller than in Lai/Pedram's approach, but comparable to other recent top decomposers by Bibilo, Luba, and TRADE.

13 Applications of Spectral Methods.

Various spectral methods have been tried to solve the Boolean Decomposition problem. In 1971 Lechner formulated a decomposition method based on orthogonal transforms. Most authors concentrate on various kinds of Walsh transforms (Hadamard, Paley, Karczmarz - these transforms differ only in ordering of their coefficients which is irrelevant from the point of view of logic synthesis), although there are few papers that use Fast Fourier, Haar, or Reed-Muller transform for binary functions and Christenson Transform for multiple-valued logic.

Hurst, Miller and Muzio [302] tried to apply orthogonal transform techniques to decomposition, but this approach could not be practically successful at the time of their publication (1970's), because no fast and memory efficient methods for orthogonal transforms were available at that time. With the recent introduction of fast methods for spectral transforms based on BDDs (Clarke and Zhao 1993 [128]) the method of Hurst, Miller and Muzio may obtain some practical value again.

Tokmen [664], discussed disjoint decomposability of multiple-valued functions by spectral means. Muzio and Hurst [459], found complete and reduced sets of orthogonal spectral coefficients for logic design. Thus, this method could be useful for spectral-based decomposition.

Varma and Trachtenberg [701] created a fast algorithm for the optimal state assignment of large Finite State Machines using spectral methods. Because the algorithm is very fast comparing to other published state assignment algorithms, it can be used for encoding purposes in decomposition.

Recently, the work by Dressig et al. [182, 183, 184] reformulated the spectral approach in the form of "groupability" analysis method, especially tuned for FPGA mapping. It should be further investigated whether the "spectral groupability" of Dressig et al has any advantages over "functional groupability" of Bochmann and Steinbach, that we have already introduced above as one of the leading recent ideas worthy our further investigations.

In this section we will present in more detail some spectral approaches that may be used in a general-purpose decomposer.

Decomposition of Boolean functions into linear blocks (linear blocks are multi-output combinational functions implemented with EXOR gates only) and nonlinear blocks (which requires a complete base AND, OR, NOT for instance) was discussed in Karpovsky [323], and R. Lechner and A. Moezzi [362].

It was shown that by examining the autocorrelation function of Boolean function F one can easily design a linear preprocessor so that a cascade implementation of this function would have a reduced complexity. No equivalent procedures for such a decomposition in the Boolean domain have been yet proposed. This fact demonstrates that at least in some problem formulations the spectral methods can solve problems that the functional methods can not.

Since the general AC decomposition is time consuming, research has been concentrated on finding ways of decomposing a function into sub-functions of restricted types (a decomposition into sub-functions of certain type will be called a "biased" decomposition. In essence, all practical decompositions are biased (even assuming disjointness of the bound and free sets of variables in a disjoint decomposition is also a "bias"). Special decompositions with EXOR (linear) preprocessor or EXOR postprocessor (EXOR decomposition from TRADE, [678]) can be considered, that assume the usage of EXOR operators of sub-functions or input variables. For instance, these ideas occur also in recent Machine Learning related papers of Wnek [689].

Varma and Trachtenberg found that EXOR decomposition is especially good for functions that belong to the class of arithmetic, error correcting and translating logic, such as code converters, adders, and functions with embedded additions. Symmetric functions, which are usually difficult, also benefit from the linearization process (linearization process consists in creating new variables that are EXORs of some subsets of input variables. It is also called linear decomposition). Randomly defined functions are not considerably affected by linear decompositions.

Functions with more than about 20 variables, which so far have been very difficult to handle with spectral methods, can be now treated efficiently. This is in striking contrast with other spectral methods which ALWAYS required exponential number of operations and storage.

Spectral methods are used in POLO system [586] to detect EXOR part of a Boolean function. Since POLO system uses spectral methods together with programs based on the "unate paradigm" such as those in the SIS package from U.C. Berkeley, POLO can easily handle not only functions that are close to the strongly unate ones, but the strongly non-unate functions as well. The decomposition of Boolean functions with pre- and post-linear parts by spectral means leads to greatly simplified circuits, which may have application in Machine Learning. Moreover, these circuits are highly testable. They can also be efficiently implemented in several existing FPGA technologies. Such decompositions are possible using the spectral approach only. They create a layer of EXOR gates at the inputs, at the outputs, or at the inputs and outputs of the multi-output function. The "EXOR Decompositions" can be applied in tandem with any other logic synthesis method, which in some cases (for instance, for arithmetic circuits) leads to a powerful synergy of the methods.

An interesting possibility investigated in POLO was to relate the two areas of logic synthesis - that have had previously very little in common - the standard "functional" approach (that uses such functional representations as cube calculus and decision diagrams), and the spectral methods. This approach gave new insights to spectral methods and allowed to "demystify" these methods as basically the "pattern matching" methods. It allowed also to create several new families of generalizations of spectral transforms. This approach can be extended to realize the algorithms using the Binary Decision Diagrams (BDDs) and the Shared, Reduced, Ordered Kronecker Decision Diagrams (KDDs) (which are EXOR-based generalizations of BDDs [494, 496]). This representation would make all the algorithms more memory efficient.

In addition, several other families of new orthogonal transforms have been found [492, 494, 496]. These transforms include the Arithmetic Transform, the Adding Transform and the Generalized Walsh Transform [588, 587]. Because of the analogous properties to the Walsh Transform, all these new transforms could find the same applications in logic synthesis, and particularly in the decomposition and characterization of Boolean functions as the Walsh Transform (the last transform is better known). However, in contrast to the classical Walsh Transform, but similarly to the Fixed Polarity Reed-Muller Transform, these new transforms have various "polarities" of input variables - so practically we can consider entire "families" of such transforms. In particular, they allow to create pre- and post- processors with some standard gates, whose types correspond to the selected type and polarity of the transform. These gates include EXOR, AND/EXOR universal gate of Davio Expansion, and other "standard cells" with high EXOR component. Similarly to Walsh, the "remainder" function, created after the removal of the pre- and post- processors, can be synthesized using any other logic synthesis method.

It should be noted here, that all the previously mentioned spectral methods that were historically based on Walsh Transform can be now investigated in light of the new transforms. Moreover, all the transforms can be now represented with efficient Decision Diagrams Methods to create such diagrams for transforms have been recently introduced. Although some of these approaches have advantages for limited fan-in libraries of cells with high EXOR component, it is not yet sure, how these decompositions compare to AC decompositions for DFC minimization. All these are open research problems.

14 PLA Decomposition.

Several authors, including Ciesielski, [721, 722], Devadas [171, 174], and Sasao, [577] proposed to apply Boolean decomposition to decompose a PLA into two or more cascaded PLAs. However, from the point of view of the mathematical apparatus used, these methods have little in common with the classical functional decomposition problem formulation, and can be rather treated as a multiple-valued minimization and input-encoding problems (for multiple-valued minimization see Brayton [92], Muzio et al [461]. and Sasao [572]). Nevertheless, the techniques introduced by them are very efficient, and, because of certain similarities of multiple-valued logic and decomposition, they should be considered by the proponents of the classical FD model as a possible source of enhancing ideas. Although their methods have been presented entirely in the PLA technology framework, they can be perhaps also modified to other types of "macro-blocks" or "universal cells".

14.1 Approach of Tsutomu Sasao to Multiple-Valued PLA Decomposition.

Contrary to Ciesielski and other previous authors, Sasao used classical AC decomposition concepts to PLA decomposition.

In 1989 Tsutomu Sasao, [577], presented an application of multiple-valued logic to serial decomposition of PLAs. His approach resembles the classical Curtis decomposition with all predecessor blocks G combined to a single multi-output block. Sasao divides the problem into two subproblems: *a partition problem* and *an encoding problem.* His first stage is very similar to all multiple-valued Curtis-like decomposition algorithms, with the only difference that the entries in the table are still binary (he discusses a multiple-valued input, binary output functions). Since this is a Curtis-like decomposition and not an Ashenhurst-like decomposition, he allows the column multiplicity index value to be higher than 2.

In order to find a simple circuit he looks for a partition with the smallest multiplicity value. For the n variable function there are 2^n different partitions. When n is small (≤ 16) a brute force method is used. For larger n the following approach is advocated. The number of variables in the bound set is denoted by n_1. Each column of the decomposition chart is represented by a logical expression. The number of expressions in the chart is 2^{n_1}. Therefore when n_1 is small the decomposition chart can be represented with small memory storage. An equivalence of two columns is checked as an equivalence of two logical expressions.

Sasao was one of the first authors to discuss the encoding problem in the framework of functional decomposition. He experimented with the "one-hot" encoding and the "minimum-length" encoding, and he calculated the total sizes of the predecessor and successor's PLAs. A very simple encoding algorithm was used by him - "the more frequent the pattern occurs in the decomposition chart, the more the number of 0's in the code". Sasao found that the one-hot encoding worked better than the minimum-length encoding. The paper was concluded with the desire of finding a good heuristic method to find a good partition more efficiently and to find a better heuristic for the minimum-length encoding.

15 FPGA synthesis.

A renewed interest in AC decomposition in recent years is caused by the introduction of Look-up Table Field Programmable Gate Arrays (FPGAs) by Xilinx in 1986, and other companies in succeeding years (AT&T, Actel, Motorola, Algotronix). When it was found that the adaptation of earlier algebraic methods didn't not work sufficiently well for the Lookup Table model, researchers switched to Boolean decomposition. Currently, in most of the systems, the decomposition is only an auxiliary process (MIS-PGA of Murgai and Brayton [452, 453], HYDRA of Filo, Yang, Mailhot, and DeMicheli, [217]).

However, with the arrival of systems such as TRADE [678], as well as the systems of Pedram [351], Steinbach [83], and Luba [398], that are *primarily* based on the decomposition, the situation is becoming to change drastically. Even the U.C. Berkeley group of Professor Brayton, who invented the algebraic factorization approach and was for a long time its most stubborn adherent, is recently devoting more attention to Boolean decomposition.

15.1 MIS-PGA and MIS-PGA(new).

In [453] R. Murgai et al presented improved algorithms for Table Look Up Architectures which were included in the well-known program MIS-PGA(new). It uses several techniques, and Roth-Karp decomposition is only one of them. They also use other decomposition techniques such as: co-factoring, and AND-OR decomposition.

MIS-PGA(new) [453] starts from a tree-like network. First, it applies a variety of decomposition methods to decompose the input network into a feasible network. The *feasible network* is a network in which the number of inputs of each node is limited. For Xilinx architecture, this input number is up to five. Compared with its predecessor *MIS-PGA*, [452], more decomposition techniques have been incorporated. The decomposition methods that *MIS-PGA(new)* employs include *cube packing* which works well for functions with more or less *mutually orthogonal cubes. Roth-Karp decomposition* is suitable for *symmetric functions* but doesn't work with non-decomposable functions. *AND-OR decomposition* can break an infeasible node into several feasible nodes. The generated feasible node is either an inverter, a two-input AND or a two-input OR gate. *Cofactoring decomposition* uses the concept of Shannon expansion to expand the network into a feasible network, in which all nodes have up to three input variables. *decomp -d* (there is no special name for this decomposition in [453]) partitions the cubes of the input network into a set of cubes having disjoint variable support, and creates a node for each partition of cubes and a node which is the OR of all these partitions. The resulting subnetworks may not be feasible, and neither are those from the *kernel extraction decomposition.* Other decomposition techniques are used to make the network feasible. After decomposition, *MIS-PGA(new)* uses a *maxflow algorithm* to generate all possible *super-nodes* and solves the *binate covering problem* to minimize the cardinality of the super-node set which covers the entire network. Finally, by solving the *maximum matching problem,* it merges all possible nodes into the FG mode CLBs.

An infeasible node is one that cannot be realized by a single Configurable Logic Block (CLB). A feasible function can be realized by a single CLB. The decomposition phase starts from a possibly infeasible network and creates a feasible network.

In *kernel extraction,* the kernels are extracted from an infeasible node. The node and the kernel are then recursively decomposed. In the RK decomposition, a bound set X of cardinality m is chosen from the fan-ins of the infeasible node n. The rest of the inputs of n form the free set, Y.

Then the decomposition of n is of the form:

$$f(X,Y) = g(G_1(X), G_2(X)......G_t(X), Y)$$

.

where $G_1, ..., G_t$ are feasible functions.

If $t + |Y| > m$, g needs to be decomposed further ($|Y|$ denotes the cardinality of Y). To get the best possible disjoint decomposition, all choices of bound sets have to be tried.

Murgai's et al implementation does not follow the original Roth and Karp papers. They do not search all possible choices of bound sets, but select the first one. This is the form in which RK decomposition has become most popular. The results may be then highly non-optimal, but the method is fast. They just observe that if the function is symmetric then all partitions are equally good, so their scheme produced excellent results on some of the benchmarks like 9symml, 9sym, rd84, and rd73. The recent improvements to this approach are in [455].

15.2 Other Lookup-Table FPGA Mappers.

Chortle-crf [223] by Francis et al starts from a *Directed Acyclic Graph* (DAG). It first divides the DAG into a *forest of trees.* Then, by using the *dynamic programming approach*, it carries out technology mapping on each tree to find the *minimum cost circuit.* Several techniques are used, such as *two-level decomposition* which uses a *bin packing algorithm*, *multi-level decomposition*, *exploiting reconvergent paths* and *replication of logic at fanout nodes.* These make the *Chortle-crf* get a significant improvement over its predecessor *Chortle* [222] in both the quality of solutions and the running time.

X-map [322] by Karplus converts BLIFF format into an *if-then-else* DAG, which is a network with the number of inputs of each node less than or equal to three. Then it goes through a *marking* and *reduction processes* to minimize the network. Finally, a simple merging algorithm is applied to merge all possible nodes into the FG mode CLBs.

VISMAP [693] by Woo introduces the concept of invisible edges. The *invisible edge* is an edge which doesn't appear in the resulting network after mapping. It starts from a feasible network (in DAG format), partitions this network into several subgraphs of a reasonable size and goes through a *pre-processing* and a *main processing* step to determine the invisible edges to reduce each subgraph. A merge algorithm is used to merge all possible nodes into the FG mode CLBs.

Hydra [217] by Filo et al is similar to MIS-PGA: it does disjoint decomposition followed by a node minimization phase. The main difference is that both phases are driven by the fact that Xilinx CLB may realize two outputs also. Hydra puts more attention on the FG mode CLBs.

The main objective of the above FPGA technology mapping approaches is to minimize the area. There are several other approaches: *MIS-PGA(d)* by Murgai et al, [454], *Chortle-d* [225] by Francis, [224], and *DAG-MAP* [145] by Cong, which aim at the delay optimization.

The FPGA mapping approaches mentioned above consist, in general, of four major steps:

- graph construction,
- decomposition,
- reduction,
- packing.

In the *graph construction* step, the very first step, a special kind of network-like graph or a set of subgraphs is created. The graph (or network) can be feasible or infeasible. Several specific FPGA mapping techniques are applied to it. In the *decomposition* step, the most important step in the mapping process, a variety of decomposition methods are applied to transform an infeasible network into a feasible one. During the process, the decomposition algorithms try to minimize the number of nodes in the decomposed network as well as the number of input variables per node. In the *Reduction* step, generally more computationally expensive, some covering algorithms are applied in order to find a set of minimum number of CLBs which can cover the entire network. In the *Packing* step, according to the specific FPGA architecture, some algorithms are used to merge the possible nodes to further decrease the area. Most of the operations used in the above four steps are local operations. The dynamic programming algorithm ensures the local operation to traverse across the network. The program is recursively invoked until a satisfactory result is reached.

It is not clear which, if any, of the above methods have some advantages to DFC minimization for very strongly unspecified functions typical for machine learning.

16 Special Restricted Classes of Decomposition.

In this section we will present various special kinds of decomposition that assume some special structures of circuits. All these decompositions, however, are close to FD and can be relatively easily added to a general-purpose FD decomposer.

16.1 The Research of Butler and Bender.

For more than twenty years, there has been an interest in fanout-free networks (trees), where each gate has a fanout of one. This interest has been motivated by the relative ease with which tests can be generated for such networks, and also by some technological restrictions.

In particular, Hayes [277] considered fanout-free networks of AND, OR and NOT gates; Chakrabarti and Kolp [130], Butler and Breeding [118] and Maruoka and Honda [414, 415] have considered networks of AND, OR, EXOR, and NOT gates. Kodandapani and Seth [339] have considered networks which also include the majority function. A special case of a fanout-free network, a cascade, has been discussed by Maitra [406] and Mukhopadhyay [448]. Cascades are fanout-free networks in which each gate connects to at least one net input.

One way to measure the relative merits of various classes of circuits is to compare the number of n-variable functions realized by a class. However, as shown by Butler and Bender, this can be deceptive, since the relative number of functions realized by two networks for small n may be quite different than that for a large n. Let ${F_T}^{CM}(n)$ be the number of n-variable functions realized by networks whose topology T is a cascade (C) or unrestricted fanout free (FF), and whose component modules (CM) are two-input EXORs (E), AND gates (A), OR gates (O), three-input majority gates (M) and inverters (N).

The particular networks of interest discussed by Butler and Bender are:

- unate cascades of Mukhopadhyay [448]: ${F_C}^{AON}(n)$.
- Maitra cascades [406]: ${F_C}^{AOEN}(n)$.
- fanout free #1 [277], [278] : $F_{FF}^{AON}(n)$.
- fanout free #2 [339] : $F_{FF}^{AOMN}(n)$.
- fanout free #3 [130, 118, 414, 415] : $F_{FF}^{AOEN}(n)$.
- fanout free #4 [339] : $F_{FF}^{AOEMN}(n)$.

For all but one of the networks listed (not published for ${F_C}^{AON}(n)$) recursive relations have been derived and have been evaluated at least for $n = 7$. However, computation time and the large values involved preclude computer evaluation much beyond this.

Consider now the relative number of functions for the various networks. The following can be determined:

1. The fraction of n-variable fanout-free functions which are cascade realizable becomes arbitrarily close to 0 as n approaches infinity for two cases, i.e.

$${F_C}^{AON}(n) = o(F_{FF}^{AON}(n))$$

and

$${F_C}^{AOEN}(n) = o(F_{FF}^{AOEN}(n))$$

.

2.

$$F_{FF}^{AOMN}(n) = o(F_{FF}^{AOEN}(n))$$

showing that the EXOR in combination with ANDs, ORs, and NOTs produces more functions than the majority gate. This is an interesting fact in view of the fact that $F_{FF}^{EN}(n) = 2$ for all n, while $F_{FF}^{MN}(n) \to \infty$ as $n \to \infty$.

3. The number of strictly fanout-free functions $F_{FF}^{AON}(n)$ represents a vanishingly small fraction of the number of functions realized by AOEN nets, verifying a conjecture by Kodandapani and Seth [339].

4. The relative number of functions for large n can be quite different than for small n.

$$F_C{}^{AOEN}(n) = o(F_{FF}^{AON}(n))$$

and

$$F_C{}^{AOEN}(n) = o(F_{FF}^{AOMN}(n))$$

However, for small n a different situation exists. For $2 \leq n \leq 6$ the AOEN cascades realize more functions than either the AON or AOMN fanout-free networks.

The asymptotic approximations for these networks suggest that the asymptotic behavior of cascades and fanout-free networks with *general* component module sets have the same characteristics of the specific examples shown here. If this is the case, the addition of a distinct module to the module set in general enlarges the function set by an arbitrarily large amount, as the number of variables approaches the infinity.

16.2 Bender's and Butler's Enumeration

In 1979 Intern. Symp. on Multiple-Valued Logic Bender and Butler derived a recursion relation is derived for the number of functions realized by fanout-free networks of multi-valued gates. An asymptotic approximation to this relation shows that when the number of logic levels grows, the number of functions realized is nearly independent of the network structure and is equal to $c\ L^n$, where n is the number of inputs and c and L are parameters that depend on the number of logic levels. Fanout-free networks are shown which realize any function of the input variables.

16.3 Maruoka's and Honda's networks of flexible cells.

Maruoka and Honda discuss logical networks of flexible cells [414]. The flexible network is a network composed of flexible multi-valued cells that can be adjusted to perform any function. The network is assumed to have a fixed-interconnection pattern and to have a fixed-input variable assignment. The following question is answered: "What is the necessary and sufficient range of the flexibility of the cells' functions to obtain all the output functions the circuit could realize?" Furthermore, the relation between the cells that realize the same output function is derived and the number of different realizable output functions is counted.

16.4 Universal Modules.

Yau and Tang [728] proposed several tree-type Universal Logic Modules (ULMs) that are based on Shannon Decomposition Theorem. The disadvantage of the realization of arbitrary functions using the universal logic modules of Yau and Tang is that, in general, it results in large trees of modules.

Although better pin bounds exist for other modules [726], [521], the arbitrary n-variable functions are easy to realize by constructing trees of k-variable modules $k \leq n$. Unfortunately, the number of these modules used in such a tree depends on the order of Shannon decompositions. In [727] Yau and Tang pose the problem of minimizing the number of modules in such trees.

Cheung and Ehrich investigated chain decompositions which, when they exist, reduce the number of modules [137]. A decomposition theorem and algorithm are given in [137] for reducing the size of such trees, using what is called the "chain decomposition". The basic assumption of the authors was that good realizations are obtained by minimizing the fan-out at each stage of the decomposition. Although the paper deals with three-variable IULM3 decompositions, their theorem can be readily generalized to ULM-k for $k > 3$. The module realizes any n-variable function by successive expansions of $k-1$ variables for ULM-k when $k < n$ each residue function is again expanded.

The following theorems are used.

Lemma 16.1 *(Curtis).*
If $f = F(N_1(A,B),C) = G(N_2(A,C),B)$ *then* $f = K_1(A) * K_2(B) * K_3(C)$
*where * is OR, AND, or EXOR operator.*

Lemma 16.2 *(Curtis).*
If $f = F(N_1(A,B),C,D) = G(N_2(A,C),B,D)$ *then* $f = H(L,D)$
where $L = K_1(A) * K_2(B) * K_3(C)$ *and * is AND, OR, or EXOR operator.*

Theorem 16.1 *(Curtis).*
Let f be a non-degenerate n-variable function. If f has two chain ULM-3 (CULM3) realizations obtained by expanding with respect to two different pairs of variables, then the expansion with respect to either pair of variables will yield identical CULM3 structures.

The length of the CULM3 realization depends on the residue function rather than the choice of the expansion variable pairs. As a result, in determining the CULM3 realizability of the function, if there exists such a pair of variables, one does not have to look for other pairs, since the decomposition theorem indicates that all realizations will yield the same ULM3 structure. A similar proof has been established when IULMM-k are used for $3 < k < n$ [136].

The algorithm of Cheung and Ehrich is the following. A function is first tested for single variable decompositions. Then tests for various types of two-variable decompositions are applied to the function successively. The algorithm is non-exhaustive to these tests for a chain ULM3 realization. Applying this algorithm enables to determine the realizability of ULM3 chain decompositions. However, it provides no information about how to carry out the realization efficiently when there is no chain decomposition at a particular stage. There is no guarantee that an initial fan-out of the tree realization might not ultimately result in a better realization. However, there seems to be no procedure, short of exhaustion, for determining when this is the case.

16.5 Irredundant Tree Networks of Chakrabarti and Kolp.

The paper by Chakrabarti and Kolp [130] is concerned with the synthesis of irrendundant tree networks with two-input single-output flexible cells. An algorithm is developed which tests whether a given Boolean function is tree realizable. If it is tree realizable, the best tree is generated which realizes the function. It is shown that for each realizable function there exist a non-trivial unique partition from which a best tree can be constructed. Finally the number of functions realizable by irredundant trees is determined. The best tree is the one having the minimum number of levels.

Definition 16.1 *(Tree-realizable function) (Chakrabarti and Kolp).*
Any two variable function is tree-realizable. A Boolean function f of n variables $(n > 2)$ *is said to be tree realizable iff there exists a two-block partition of f*
$P = \{B_1, B_2\}$
such that
$f(X) = F(K_1(B_1), K_2(B_2))$
and K_1 *and* K_2 *are tree-realizable.*

Theorem 16.2 *(Chakrabarti and Kolp).*
Let P be a simple partition of f (i.e. $f(X) = F(\Psi(V), V')$. Then f is tree realizable iff F and Ψ are tree-realizable.

Theorem 16.3 *(Chakrabarti and Kolp).*
If f is tree-realizable, then there exists a nontrivial unique partition A of f, and A and the Boolean operation $*$ are unique.

Theorem 16.4 *(Chakrabarti and Kolp).*
Let f be a given tree-realizable function which is expressed by its unique form as:

$$f(X) = F_1(B_1) * F_2(B_2) * \ldots * F_m(B_m)$$

Let also $F(Y) = y_1 * y_2 * \ldots * y_m$ and $d(y_i) = l(F_i)$. Then $d(F(Y)) = l(f)$.

17 Decomposition of Multiple-Valued Logic Functions.

The decomposition of Multiple-Valued Functions has been discussed by:

- Thelliez, [660] (only for ternary logic).
- Walliuzzaman and Vranesic, [677].
- Fang and Wojcik [211].
- Luba, Mochocki and Rybnik [396, 394, 393]
- Abugharrbieh and Lee, [7, 8]

Similarly to the binary case, there are two types of decomposition for multiple-valued functions:

1. AC-like decomposition,
2. Dietmeyer-like decomposition.

The first approach is to extend the AC binary approach. This approach is represented by Walliuzzaman and Vranesic, [677], and Thelliez, [660], for ternary logic. Especially the partition-based approach of Luba allows him to create an algorithm that is a straightforward adaptation of his former binary algorithms.

The second approach is represented by Fang and Wojcik, [211]. Their paper is a very good reference for some non-FD approach to Boolean Decomposition. We will call this approach a *compositional approach to multiple-valued logic synthesis.*

Both the above approaches have their characteristic advantages and disadvantages. The first approach may require generating of all partitions. Next, function representation for each partition must be examined to determine if it possesses the specific decomposition property that is being considered. Since only a very small number of functions will have a specific property [114], if the function does not have the desired property, the analysis does not yield a design. Hence the designer will have to consider alternative decompositions or select some other approaches to design the function. Finally, in multiple-valued systems, the entire classical decomposition approach is considerably more complex than for binary systems because of the associated combinatorial explosion (this is not necessarily true in the case of Luba's decomposition, but there are no comparisons of binary versus multiple-valued decomposers yet).

The disadvantage of the second method is its inherent strong bias. Its advantage may be that it will appear easy to convert it to the Abductive Networks developed by ABTECH Corporation [6, 439].

17.1 The Compositional Approach of Fang and Wojcik.

The method of Fang and Wojcik follows the approach of Dietmeyer [176], but it is different in several key respects.

In the paper by Feng and Wojcik, the complex problem of identifying whether or not the function is decomposable is greatly simplified, because the decomposition is considered only in terms of available components. The components may consist of a set of building blocks or a single universal logic module. All functions can be decomposed using this approach. In the worst case, a canonical form of the function is generated, leading to a tree structure to implement the function.

The basic idea is to develop a systematic procedure which emphasizes function decomposition with modularity. They apply a top-down approach which seeks to identify common sub-functions. The decomposition of the target function into sub-functions is considered in terms of available components. The components must constitute a functionally complete set. The set can be a collection of building blocks, or a single universal building block such as an m-variable multiplexer, or an n-variable PLA. The technique of Dietmeyer was also applied to multiple-valued logic by Antoni Michalski in his 1980 Ph.D. in Poland [426].

The technique of Dietmeyer is based on using building blocks that are determined *after* the search of partition matrices for a classic decomposition property. Since the decomposition may not exist, the set of building blocks will not be found.

While in AC approach a function is rewritten in terms of new variables (functions), Wojcik and Feng do not attempt to rewrite the function in terms of sub-functions which are next used as new variables in the logic equation of the function. Rather, they identify which of the available components used to implement the sub-functions can be interconnected to represent the target function.

In AC decomposition a binary function:

$$f(x_1, x_2, x_3, x_4) = x_1x_3 + x_2x_3 + x_1x_4 + x_2x_4$$

is rewritten to:

$$f(x_1, x_2, x_3, x_4) = F(A(x_1, x_2), B(x_3, x_4))$$

where:

$$F(A, B) = A\ B$$

$$A(x_1, x_2) = x_1 + x_2$$

$$B(x_3, x_4) = x_3\ +\ x_4$$

In the approach of Feng and Wojcik for this example, if a module composed of a two-input OR gate is used as the available component, then the function may be rewritten as:

$$F(x_3, x_4) = x_3G(x_1, x_2) + x_4G(x_1, x_2) = G(x_1, x_2)(x_3 + x_4)$$

where:

$$G(x_1, x_2) = x_1 + x_2$$

is a sub-function.

The approach uses a predefined collection of modules that are used to design an arbitrary function. The approach is applicable to any radix and indeed can be applied to binary functions as well.

Feng and Wojcik developed the synthesis method along with several techniques to minimize the number of components used in the design. The objective of the minimization was to reduce the number of building blocks used from a predefined component collection, but not to minimize the number of basic logic components from which the building blocks are composed. These techniques do not guarantee an optimal solution, namely the least number of the building blocks. Furthermore, optimality is a function of many criteria, such as the number of modules, the complexity of module interconnections, the number of logic levels in the design, etc.

Further work is needed to study the interaction among the optimization techniques and their ability to produce an optimal design. It is also necessary to develop other techniques that will contribute to achieving the optimization that is desired. Although some work on these kinds of compositional methods has been done in the past, recently there have been no papers applying such ideas in frameworks of modern representations and search methods.

It is this our opinion, that the approach of Feng and Wojcik would give very good results if implemented with BDDs or KDDs (This kind of techniques have not been tried with DDs yet).

17.2 Tokmen's Approach to Disjoint Decomposability of Multi-valued functions by Spectral Methods

.

The decomposition methods based on spectral analysis were investigated by Tokmen [664] and more recently by Poswig [516]. It was shown that if the Walsh spectrum of a function is evaluated using an FFT-like algorithm, the execution time of a program can be reduced significantly [701]. Other good methods can be found in [208, 209, 128]. However as the number of variables increases, the execution time still grows and it is not sure what are the practical limits of using these methods to decomposition.

The method of Tokmen, [664], is defined as follows. Given is an m-valued function. The method makes use of the given function $f(X)$ and of the functions (gates) with which it is desired to realize the decomposition. The method involves the solution to a set of m^k simultaneous equations each with m unknowns for the m-valued case. The decomposition is possible if each and all sets of simultaneous equations have unique solutions. The method may be applied to functions of arbitrary size and is implementation independent.

Further, if it is found that the decomposition using the desired gates is possible, then the detection and proof of this decomposition fully defines the spectrum and hence functional relationships of the next level (reminder) function h. The detection of such simple disjoint decomposability of a given multi-valued function may be considered as the first stage in the design of combinational multi-valued logic circuits. Tokmen refers to a paper by Fricke [227], and a book by Karpovsky [323].

17.3 Waliuzzaman's Approach to Multi-Valued Functions Decomposition.

The approach is presented in [677]. The standard Curtis-like approach of investigating all possible decompositions to find the best one becomes prohibitively time consuming for systems with $m > 4$. The break-count procedure presented by Walliuzzamam allows for easy selection of decompositions which lead to simple implementation.

Theorem 17.1 *(Walliuzzaman).*
A switching function $f(X)$; $X = x_1, x_2, \ldots, x_n$ has a simple disjunctive decomposition given by $F(Y, h(Z))$, $Y = y_1, y_2, \ldots, y_{n-s}$, $Z = z+1, z_2, \ldots, z_s$ if and only if its partition matrix $M(f, Y|Z)$ has column multiplicity $\mu \leq m$.

The proof is obvious, since in such case each column type is encoded with one logic value of the output of the predecessor function.

Theorem 17.2 *(Walliuzzaman).*
If $f(X) = f(A, B, C)$ can be expressed as $f(A, B, C) = F(h(A, B), C) = G(N(A), B, C)$ then $f(A, B, C) = F(K(N(A), B), C)$ where $K(N, B)$ is a uniquely determinable function of N and B for which $K(N(A), B) = h(A, B)$.

Theorem 17.3 *(Walliuzzaman).*
If $f(A, B, C) = F(h(A), B, C) = G(N(B), A, C)$ then $f(A, B, C) = H(h(A), N(B), C)$ where $H(h, N, C)$ is a uniquely determinable function of h, N and C.

These theorems are just modifications of well-known Ashenhurst and Curtis theorems to mv logic. Walliuzzaman's paper refers to papers by Arango, Santos, and Chacur [22], and Merril [425]. It seems to me that presently this paper is only of historic value since more powerful and general methods have been described in later years.

17.4 Multiple-Valued Decomposition by Abugharrbieh and Lee.

The objective of Abugharrbieh's and Lee's research [7, 8] was to generalize the Shen et al [607, 608, 609] binary function decomposition algorithm to m-valued functions with $m > 2$. They obtained the necessary condition for the decomposability for m-valued functions and used it in generating candidate bound sets. A fast method for testing the necessary condition using partial partition tables (PP tables) was also created, whereby the decomposability of the function can be determined for a randomly chosen function in $(Nm)^3$ time.

A fast algorithm for disjoint decomposition that does not require exponential time, proposed by Shen and Kellar [607, 608, 609], applies Jacobian operator to switching functions to test if they satisfy the necessary condition. This operator requires switching function to be given as an algebraic expression in a canonical form using AND, EXOR, and NOT operators. Shen and Kellar showed that a randomly chosen function of n variables has a very small probability of being decomposable and the time needed to conduct such a decomposition test is $O(n^3)$.

Abugharrbieh and Lee developed an algorithm that is capable of testing m-valued functions for $m > 2$. They generalized the necessary condition developed by Shen et al for the multi-valued logic. Instead of evaluating the Jacobian operator, a different method is used to test the necessary condition for decomposability. Furthermore the method operates on functions which may be given either by truth tables or algebraic expressions. They showed that for n-variable m-valued randomly chosen functions the execution time is $O(nm)^3$ which is considerably less than in other approaches to mv decomposition: [318, 162]. In the case where a function has a single disjunctive decomposition, the execution time is $O(n^3m^n)$.

Theorem 17.4 *(Abugharrbieh and Lee).*
An m-valued function $f(X)$ has a simple disjunctive decomposition with a bound set X_1 and a free set X_2 if and only if the column multiplicity μ (the number of distinct columns) of $T(X_1 : X_2)$ is less than or equal to m.

Theorem 17.4 is the same as the Theorem 1 of Walliuzzaman.

Theorem 17.5 *(Abugharrbieh and Lee).*
Let $f(X) = g(h(X_1), X_2)$ be a decomposable m-valued function with n variables. If $\{x_i, x_j\}$ is included in X_1 and x_k is a member of X_2, $i \neq j$, then all the PP tables $T(x_i, x_j : x_k, e)$, $0 \leq e \leq m^{n-3} - 1$ have column multiplicity $\mu \leq m$.

This indicates a necessary condition for a function $f(X)$ to be decomposable.

Theorem 17.6 *(Abugharrbieh and Lee).*
Let $f(X)$ be an m-valued function with n variables. If x_1, x_2 and x_k are members of X, $i \neq j \neq k$ and at least one PP table $T(x_o, x_j : x_k, e)$, where $0 \leq e \leq m^{n-3} - 1$, has column multiplicity $\mu > m$ then if $f(X) = g(h(X_1), X_2)$ such that x_i, x_j is included in X_1 then x_k is a member of X_1.

Theorem 17.7 *(Abugharrbieh and Lee).*
Let $f(X)$ be an m-valued function with n variables. If B is the set of all AMB sets then any CB set can be obtained by a finite union of elements of B.

In lieu of the Jacobian operator, partial tables were used in testing of the necessary condition for decomposability. The method is easy to implement.

17.5 Luba's et al Approach to Multiple-Valued Decomposition.

Some new papers by Luba et al are on decomposing multiple-valued functions. They are related to Machine Learning, [398, 387]. Their approach to decomposition is similar to both classical Curtis and Walliuzzaman approaches. The difference is in that they apply the partition theory. The sufficient condition in this approach leads to the calculation of minimal cover of the maximum compatible classes.

Luba does not write in his papers how he selects the "bound set". In general, he does not devote much attention to the algorithmic search aspects. Some new results are obtained by using the method described by Selvaraj in his Ph.D. thesis [602]. The new idea is to combine the parallel and serial decompositions in a single program. In an unpublished paper, Piotr Sapiecha in Poland is dealing with "Simulated Annealing" algorithm to calculate the so-called reducts. In Luba's understanding they are nothing but the minimal sets of non-vacuous variables that Luba uses in his approach.

The really innovative asset of Luba's approach is his partition-based representation that allows for relatively easy determination of several properties. Most, if not all, of his theorems are restatements of known theorems to his notation, but the method seems to have a large potential for incompletely specified functions. It is still not clear which, if any, of his theorems and methods are really new. For instance, his products of partitions have close relation to cofactors, his covering of the compatibility graph corresponds to the coloring of the incompatibility graph in TRADE or to the set covering of Pedram's approach, and so on.

18 Decomposition of Information Systems.

Many problems related to decision making are of the following nature: Given a decision table - find all minimal decision algorithms associated with the table. This problem can also be seen as learning from examples, or, as it is often called, decision rules generation from examples - Grzymala-Busse, [265] [266] [267]. These problems have been investigated from a number of points of view: one of them is whether the whole set of attributes is always necessary to define a given partition of an universe, and the other concerns the simplification of decision tables, namely the reduction of condition attributes in a decision table.

The concept of information system decomposition can be treated as an aid in analyzing the existing system [615] or as a base for designing a new one [12, 616]. Quality of decomposition depends mainly on the analyst's experience. Some authors try to provide a formal approach to decomposition [483]. Decomposition approach similar in essence to one proposed earlier by Armstrong [25] was presented by Luba, Mochocki and Rybnik in 1993. Avoiding data redundancy is an important problem in the implementation of information systems. The problem is usually overcome by devising simpler rules and removing redundant rules, as well as minimizing the number of attributes.

The importance of decomposition in general system design is emphasized by Courtois [483]: "Decomposition has long been recognized as a powerful tool for the analysis of large and complex systems".

From the technical point of view, the decomposition phase is exactly the same as in the multiple-valued decomposition with no constraint on the number of values of the output from the predecessor block, so it will be not further discussed.

19 Decomposition of Fuzzy Logic Functions.

Decomposition of fuzzy logic functions was first discussed by Abraham Kandel in [314]. Later he found errors in his paper which, after removing, made his method completely impractical. Therefore, in the next paper by Francioni and Kandel [221], they propose a much improved method for disjoint decomposition.

The disadvantages of their new method include however the following:

1. Lack of intuitive explanation how their decomposition rules have been created makes it difficult to improve the algorithm.
2. The algorithm is strictly tabular, and no computer implementation exists.
3. The algorithm is quite complex and requires another program to minimize fuzzy functions.

The non-disjoint decomposition of fuzzy logic functions remains an open problem. Francioni and Kandel mention interesting examples of applications of fuzzy logic decomposition, including medical diagnosis, hardware realization of fuzzy logic circuits, pattern recognition, and document retrieval systems.

19.1 Introduction

19.1.1 Fuzzy Logic

Definition 19.1 *Fuzzy set, defined as A, is a subset of universe of discourse U, where A is characterized by a membership function $u_A(x)$. The membership function $u_A(x)$ is associated with each point in U and is the "grade of membership" in A. The membership function $u_A(x)$ is assumed to range in the interval [0,1], 0 corresponding to non-membership and 1 corresponding to full membership. The ordered pairs form $\{(x, u_A(x))\}$ to represent the fuzzy set member and the grade of membership, [732].*

Operations:

The intersection operation of two fuzzy sets uses the symbols: $\cap, *, \wedge$, AND, or min. The union operation of two fuzzy sets uses the symbols: $\cup, \vee, +$, OR, or max.

- Equality of two sets is defined as $A = B \leftrightarrow u_A(x) = u_B(x)$ for all $x \in X$.
- Containment of two sets is defined as A subset $B, A \subseteq B \leftrightarrow u_A(x) \leq u_B(x)$ for all $x \in X$.
- Complement of a set A is defined as $\overline{A}$, where $u_{\overline{A}}(x) = 1 - u_A(x)$ for all $x \in X$.
- Intersection of two sets is defined as $A \cap B$ where $u_{A\cap B}(x)$ =min$\{(u_A(x), u_B(x))\}$ for all $x \in X$ where $C \subseteq A, C \subseteq B$ then $C \subseteq A \cap B$.
- Union of two sets is defined as $A \cup B$ where $u_{A\cup B}(x)$ =max$\{(u_A(x), u_B(x))\}$ for all $x \in X$ where $D \supseteq A, D \supseteq B$ then $D \supseteq A \cup B$, [562].

Example 19.1 Let $X = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$ and consider the following two fuzzy sets A and B: $A = \{(3, 0.8), (5, 1), (6, 0.6)\}$ and $B = \{(3, 0.7), (4, 1), (6, 0.5)\}$

Then:
$A \cap B = \{(3, 0.7), (6, 0.5)\}$
$A \cup B = \{(3, 0.8), (4, 1), (5, 1), (6, 0.6)\}$
$\overline{A} = \{(1, 1), (2, 1), (3, 0.2), (4, 1), (6, 0.4), (7, 1), (8, 1), (9, 1), (10, 1)\}$

Identities:

Idempotency:

$$X + X = X,\ X * X = X$$

Commutativity:

$$X + Y = Y + X,\ X * Y = Y * X$$

Associativity:

$$(X + Y) + Z = X + (Y + Z),\ (X * Y) * Z = X * (Y * Z)$$

Absorption:

$$X + (X * Y) = X,\ X * (X + Y) = X$$

Distributivity:

$$X + (Y * Z) = (X + y) * (X + Z),\ X * (Y + Z) = (X * Y) + (X * Z)$$

Complement:

$$\overline{\overline{X}} = X$$

DeMorgan's Laws:

$$\overline{(X + Y)} = \overline{X} * \overline{Y},\ \overline{(X * Y)} = \overline{X} + \overline{Y}$$

Transformations:

Some transformations of fuzzy sets with examples follow:

$$\overline{x}b + xb = (x + \overline{x})b \neq b$$

$$xb + x\overline{x}b = xb(1 + \overline{x}) = xb$$

$$\overline{x}b + x\overline{x}b = \overline{x}b(1 + x) = \overline{x}b$$

$$a + xa = a(1 + x) = a$$

$$a + \overline{x}a = a(1 + \overline{x}) = a$$

$$a + x\overline{x}a = a$$

$$a + 0 = a$$

$$x + 0 = x$$

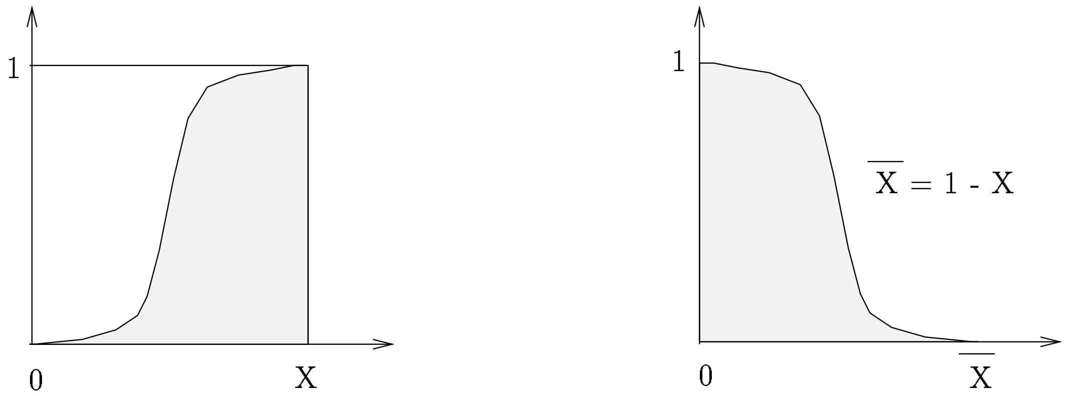


Figure 18: Membership Function and Inverse

$x * 0 = 0$

$x + 1 = 1$

$x * 1 = x$

Example 19.2

$$a + xa + \overline{x}b + x\overline{x}b = a(1 + x) + \overline{x}b(1 + x)$$
$$= a + \overline{x}b$$

Example 19.3

$$a + xa + \overline{x}a + x\overline{x}a = a(1 + x + \overline{x} + x\overline{x})$$
$$= a$$

Differences between Boolean logic and Fuzzy logic

In Boolean logic the values of a variable and its negation are always disjoint $(X * \overline{X} = 0)$ and $(X + \overline{X} = 1)$ because the values are either zero or one. In fuzzy logic the membership functions can be either disjoint or non-disjoint. The membership function is determined by the grade of membership and can be any value in the interval $[0, 1]$ from zero to one (it means, both arguments and values are in this interval). Fuzzy membership functions can be any function that can be realized in the interval from zero to one. For simplicity, the term 'grade of membership' of a variable in a set will be replaced by the term 'fuzzy variable'. An example of a fuzzy membership function X with its inverse membership function is shown in Figure 18.

Another fuzzy logic membership function, which is linear, can be defined by the function shown in Figure 19.

With the membership function from Figure 19 the inverse of the membership function is shown in Figure 20.

The fuzzy intersection of variables X and its complement $\overline{X}$ is not empty, or is not equal to zero because the membership functions are non-disjoint. In Boolean logic the intersection or AND of a variable and its complement is zero. The intersection of membership functions from X Figure 19 and its complement $\overline{X}$ Figure 20 is shown in Figure 21.

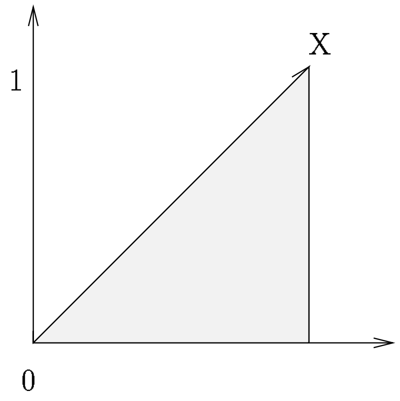


Figure 19: Membership function X

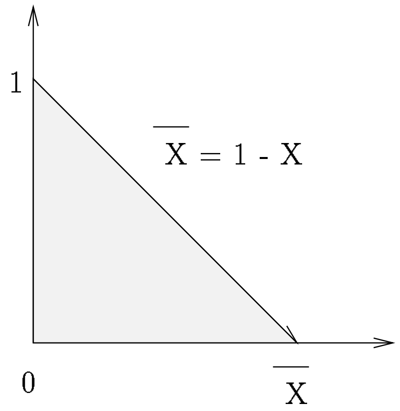


Figure 20: A complement of the membership function X

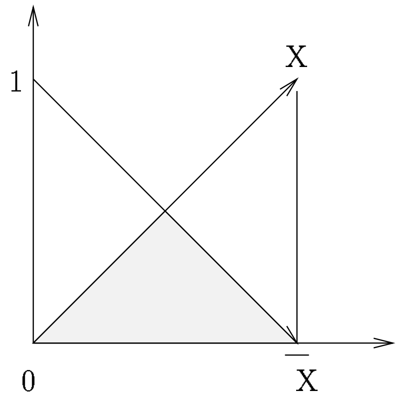


Figure 21: Intersection $X * \overline{X} \neq \emptyset$

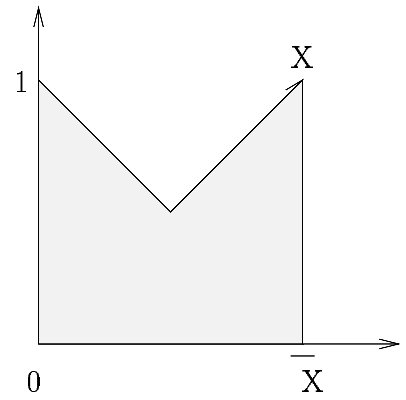


Figure 22: Union $X + \overline{X} \neq 1$

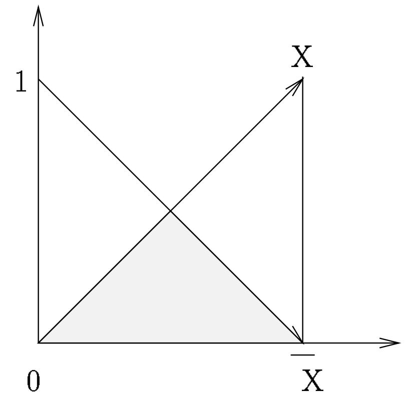


Figure 23: $X \oplus X = X * \overline{X} + \overline{X} * X = X * \overline{X}$

The fuzzy union of the variables X and its complement $\overline{X}$ is not equal to one because the membership functions are non-disjoint. In Boolean logic the union or OR of a variable and its complement is one. The union of membership functions from X Figure 19 and its complement $\overline{X}$ Figure 20 is shown in Figure 22.

The xor of the fuzzy variables X and X is shown in Figure 23 which is the same as the intersection of X and $\overline{X}$.

The xor of the fuzzy variables X and $\overline{X}$ is shown in Figure 24 which is the same as the union of X and $\overline{X}$.

19.2 The Problem of Decomposition of Fuzzy Functions

Theorem 19.1 *Let y, z be a partition on X. A completely specified Boolean function $f(X)$ possesses a simple disjunctive decomposition with bound set Y and free set Z if and only if its Y/Z decomposition map has a column multiplicity $v \leq 2$.*

The proof of this theorem is based on the Shannon's expansion theorem which proves that for a Boolean function f,

$f(x_1, x_2, ..., x_n) = x_1 f(1, x_2, ..., x_n) + \overline{x_1} f(0, x_2, ..., x_n)$[604].

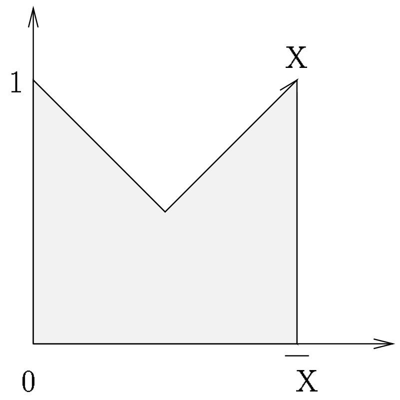


Figure 24: $X \oplus \overline{X} = X * X + \overline{X} * \overline{X} = X + \overline{X}$

yz \ wx	00	01	10	11
00				I
01				
10				
11	I	I	I	

Figure 25: Decomposition map of Example 4.

Unfortunately, the Shannon's expansion theorem (and therefore Theorem 19.1) does not hold true for fuzzy functions which are not disjoint binary functions but non-disjoint infinity-valued functions. The next example will shows that a function can be decomposed correctly for binary functions but does not decompose correctly for fuzzy functions.

Example 19.4 Let the function f be

$$f(w,x,y,z) = w\overline{x}yz + \overline{w}\,\overline{x}yz + \overline{w}xyz + wx\overline{y}\,\overline{z}.$$

The Y/Z decomposition map of f corresponding to the Boolean representation of the variables and to the partition $Y = \{w, x\}$, $Z = \{y, z\}$ is shown in Figure 25.

Therefore the Boolean function represented by f has a simple disjunctive decomposition of the form

$$F[G(w,x), y, z] = G\overline{y}\,\overline{z} + \overline{G}yz$$

where $G(w,x) = wx$.

However, if f is a fuzzy function, then the variables represent grades of membership in a set, $u(x_i)$, and take on values between 0 and 1 inclusive and not just 0 and 1.

Let $u(w) = 0.4, u(x) = 0.2, u(y) = 0.9$ and $u(z) = 0.9$.

Then $f(w, x, y, z) = w\overline{x}yz + \overline{w}\ \overline{x}yz + \overline{w}xyz + wx\overline{y}\ \overline{z}$

$= 0.4 * 0.8 * 0.9 * 0.9 + 0.6 * 0.8 * 0.9 * 0.9 + 0.6 * 0.2 * 0.9 * 0.9 + 0.4 * 0.2 * 0.1 * 0.1$

$= 0.4 + 0.6 + 0.2 + 0.1$

$= 0.6$

But

$F[G(w, x), y, z] = (wx)\overline{y}\ \overline{z} + \overline{(wx)}yz$

Using DeMorgan's Law and distributive law we obtain:

$= wx\overline{y}\ \overline{z} + \overline{w}yz + \overline{x}yz$

$= 0.4 * 0.2 * 0.1 * 0.1 + 0.6 * 0.9 * 0.9 + 0.8 * 0.9 * 0.9$

$= 0.1 + 0.6 + 0.8$

$= 0.8$

As the previous example shows Theorem 19.1 does not hold. Next graphical methods are used to correct the decomposition of fuzzy functions.

19.3 Graphical Representation of Fuzzy Functions for Decomposition.

In Boolean logic maps the symbols "1", "0", and "-" are used to describe a binary function. These symbols denote the values of cells (minterms). Since there is only a finite number of unique terms in a fuzzy function, a symbol I can be used to show if a term is present [413].

19.3.1 Fuzzy Maps

As presented by Schwede and Kandel [599], the fuzzy map may be regarded as an extension of the Veitch diagram [712], which forms the basis for the Karnaugh map [316]. Fuzzy maps pictorially describe the set of all fuzzy implicants which represent a fuzzy function. In a K-map of n variable minterms can be represented by 2^n areas in the map. A fuzzy map of n variables can be represented by 4^n areas in the map. The symbol I is used to represent a term in the fuzzy function, $F(x_1, x_2, ..., x_n)$. For two variables fuzzy map, the columns are labeled $x_1\overline{x}_1, x_1, \overline{x}_1, 1$ and the rows are labeled $x_2\overline{x}_2, x_2, \overline{x}_2, 1$ as shown in Figure. 26 [599].

The construction of fuzzy maps of min or + as intersection, and max or * as union, the area where I is placed is easy to determine. The intersection and union of two variables are shown in Figure 27a and 27b, respectively.

The column and row headings are conventionally replaced with quarternary numbers representing the binary headings. There are four combinations for each variable x_i, i in $1, 2, ..., n$ variables, to be represented in the headings of the rows and columns.

	$X_1\bar{X}_1$	X_1	$\bar{X}_1$	1
$X_2\bar{X}_2$				
X_2				
$\bar{X}_2$				
1				

Figure 26: Fuzzy Map

	$X_1\bar{X}_1$	X_1	$\bar{X}_1$	1
$X_2\bar{X}_2$				
X_2				I
$\bar{X}_2$				
1		I		

$X_1 + X_2$

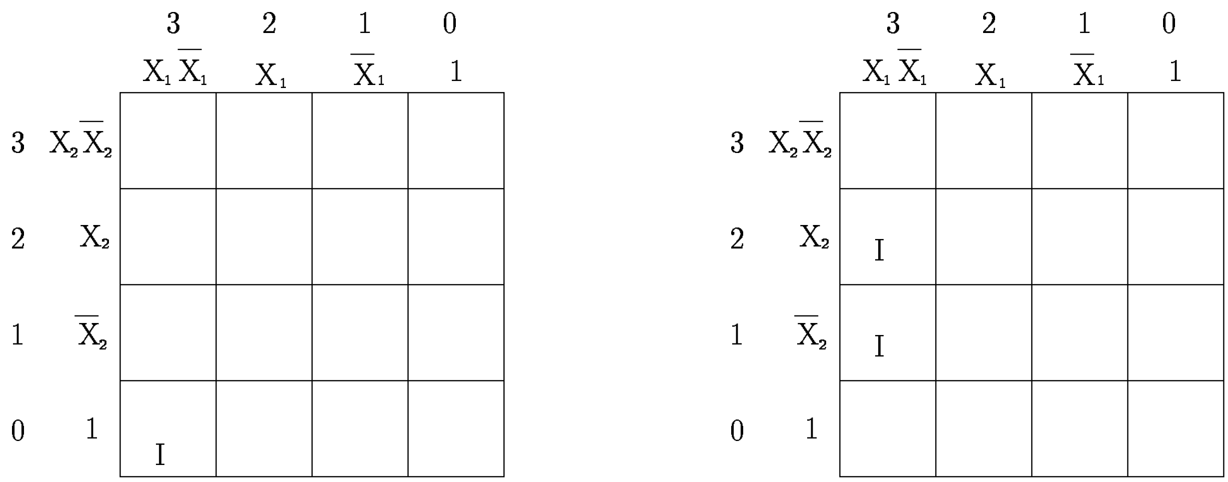
(a)

	$X_1\bar{X}_1$	X_1	$\bar{X}_1$	1
$X_2\bar{X}_2$				
X_2		I		
$\bar{X}_2$				
1				

$X_1\ X_2$

(b)

Figure 27: Max and Min representations using fuzzy maps with (n = 2)



(a) (b)

Figure 28: $F(x_1, x_2) = x_1\overline{x}_1$, $F(x_1, x_2) = x_1\overline{x}_1x_2 + x_1\overline{x}_1\overline{x}_2$

1. This heading is vacuous in x_i. The pair $x_i\overline{x}_i$ is denoted by 00 and is represented by 0.
2. This heading includes $\overline{x}_i$ but not x_i. The pair $x_i\overline{x}_i$ is denoted by 01 and represented by 1.
3. This heading includes x_i but not $\overline{x}_i$. The pair $x_i\overline{x}_i$ is denoted by 10 and represented by 2.
4. This heading includes x_i and $\overline{x}_i$. The pair $x_i\overline{x}_i$ is denoted by 11 and represented by 3.

Fuzzy map representation has important properties which distinguish it from Boolean maps. One of the most obvious of these is that the squares (each representing a different implicant) are not disjoint in the sense that the absence of an I in a square does not necessarily exclude the associated implicant from an equivalent representation of the function. Conversely, the presence of an I does not imply essentially the associated implicant. The two different representations of the same function Figures 28a and 28b are examples of these properties. The two fuzzy maps are of size $n = 2$ and the function in Figure 28a is the same as the function in Figure 28b because x_2 is vacuous so $F(x_1, x_2) = x_1\overline{x}_1$ can be expanded to $F(x_1, x_2) = x_1\overline{x}_1x_2 + x_1\overline{x}_1\overline{x}_2$.

19.3.2 S-Maps

S-maps are another way to arrange two-variable fuzzy maps for n variables. To construct an n-variable S-map, whole one- or two- variable fuzzy maps are treated as though they were squares of an S-map on $n-1$ or $n-2$ variables. This method is just iterated for n variable S-maps. These subsets of the logical space are called sub-maps and are a very important feature of S-maps. As in fuzzy maps, the binary headings for the columns and rows are converted to a quarternary representation as shown in Figure 29. The sub-map boundaries are indicated by the vertical solid lines.

S-maps have some important properties as follows:

1. The dominant square of a fuzzy map is the square representing the minterm 1.
   The dominant square of the entire S-map is designated 0 while the dominant square of any sub-map is the lowest numbered square in that sub-map. The dominate square of the entire S-map in the S-map shown in Figure 29 is square $X_1 = 0$, $X_2 = 0$, and $X_3 = 0$. The sub-map dominate squares are $X_1 = 3$ $X_2 = 0$ $X_3 = 0$, $X_1 = 2$ $X_2 = 0$ $X_3 = 0$, and $X_1 = 1$ $X_2 = 0$ $X_3 = 0$.

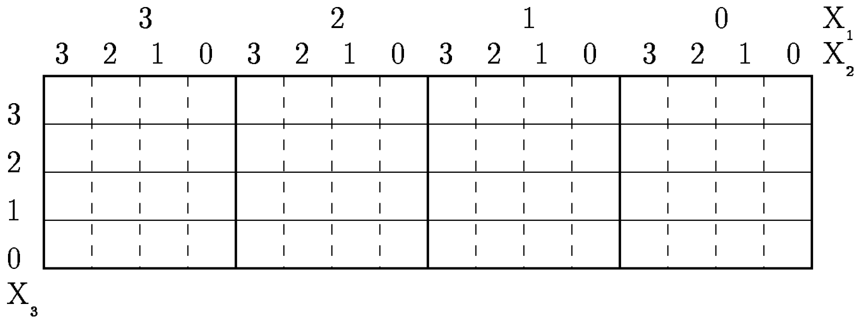


Figure 29: S-map for n = 3.

Theorem 19.2 *(Kandel* [315]*).*
A sub-map subsumes (is included by) a second sub-map if the minterm represented by the dominant square of the first subsumes the minterm represented by the dominant square of the second sub-map.

2. **Theorem 19.3** *(Kandel* [315]*).*
Every subset (pattern of implicant squares) of a sub-map is included in the dominant square of a subsumed sub-map.

3. **Theorem 19.4** *(Kandel* [315]*).*
Any subset of a sub-map is included by the same pattern on a subsumed sub-map.

The same manipulations used on a two-variable map can be used on an n-variable S-map. On S-maps entire sub-map sized patterns behave as single implicants as on the two-variable map.

Both fuzzy maps and S-maps help to decompose fuzzy switching functions.

- Fuzzy map is used to find if a decomposition exists.
- S-map is used to determine the decomposition, it means, to calculate the G and H functions.

19.4 Fuzzy Logic Multiplexers

Fuzzy logic multiplexers are like binary logic multiplexers in that the values of one or more control variables influence what input data goes to a single output. They both have many inputs and one output. In binary logic multiplexers the control variable or variables select which input from the multiplexer is transmitted to the output. In binary logic multiplexers the only input that is transmitted to the output is selected by the control variable or variables. In fuzzy logic multiplexers the inputs can be transmitted simultaneously to the output selected by the inputs values and the control variable value(s). In the fuzzy logic multiplexer, the different input variables are compared with a constant 1, a control variable or variables, their complements, and the control variable or variables and their complements, in parallel, using minimum operations. The maximum of these minimum operations makes the output. A fuzzy logic multiplexer and its internal realization with fuzzy gates are shown in Figure 30.
This multiplexer works as follows:

1. The first input is compared with the constant 1, which is the highest value, because the control variable is vacuous so the input is the minimum value.
2. The second input value is compared to the inverted control variable $1 - X$.

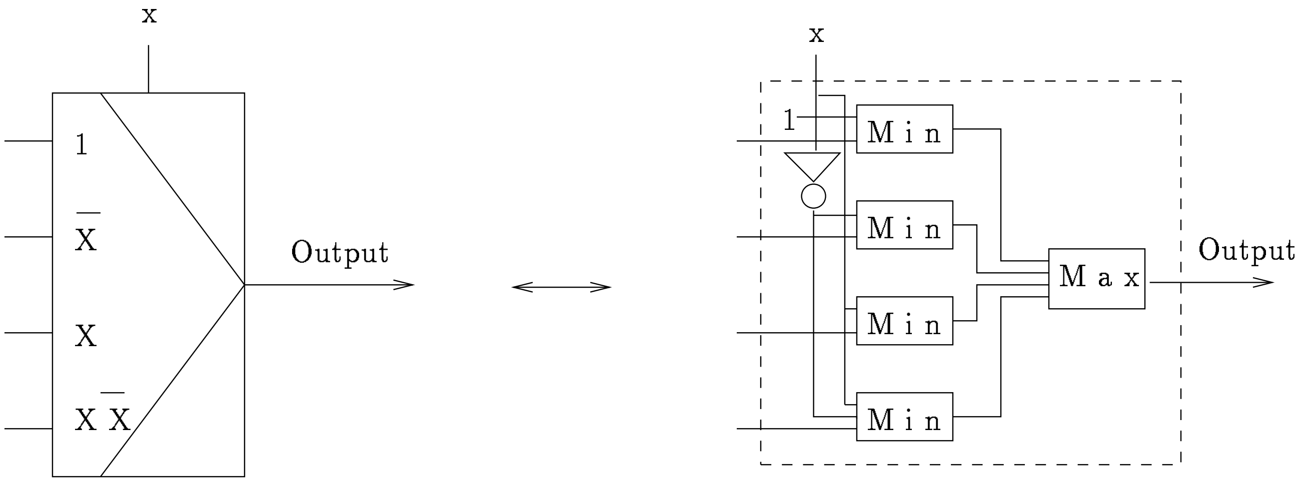


Figure 30: Fuzzy Multiplexer, (a) symbol, (b) internal structure using a Fuzzy Inverter, Min,and Max gates

3. The third input is compared to the control variable X.
4. The fourth input value is compared with the control variable X and the inverted control variable value $1 - X$.

After these four minimum comparisons are done the maximum of these four is the output.

Example 19.5
$f(w,x,y,z) = \overline{w}\ \overline{x}\ \overline{z} + w\overline{x}z + \overline{w}yz + wy\overline{z} = (\overline{w}\ \overline{z} + wz)\overline{x} + (\overline{w}z + w\overline{z})y$

If we let $G(w,z) = \overline{wz} + wz$

then $f(w,x,y,z)$ can be decomposed as follows:

$$f(w,x,y,z) = G\overline{x} + \overline{G}y = F[G(w,z),x,y].$$

The decomposed fuzzy circuit uses a fuzzy multiplexer as shown in Figure 31. The data inputs to the multiplexer with 2 address variables are labeled like in fuzzy maps with a quarternary representation. The function that the multiplexer needs to perform is $G\overline{x} + \overline{G}y$. The inputs to the multiplexer are G and $\overline{G}$ which needs to go to the multiplexer inputs $\overline{x}$ and y. The $\overline{x}$ input is labeled 10 because x is the first column and y is the second column. The 10 input label is translated to $\overline{x}$ and 1 in the quarternary representation the 1 and $\overline{x}$ is intersected to equal $\overline{x}$. The y input is labeled 02 because in translating from the quarternary representation 0 is 1 and 2 is y, 1 intersected with y results in y.

19.5 Decomposition of fuzzy functions

Definition 19.2 *A fuzzy switching function, $f(X)$, is said to be* **normal** *in x_i if it is in disjunctive normal form where no minterm includes an $x_i\overline{x}_i$, and is written $f_i(X)$. If $f(X)$ is normal in y_i for every y_i in $Y = y_1, y_2, ..., y_n$, then $f(X)$ is said to be normal in Y and is written $f_y(X)$.*

Definition 19.3 *Let $\{Y/Z\}$ be a* **partition** *on the set of fuzzy variables X. A fuzzy decomposition chart for a function $f(X)$ with a $\{Y/Z\}$ partition of X is a reduced fuzzy map where the variables of Y define the columns, the variables of Z define the rows and the following columns are excluded: the*

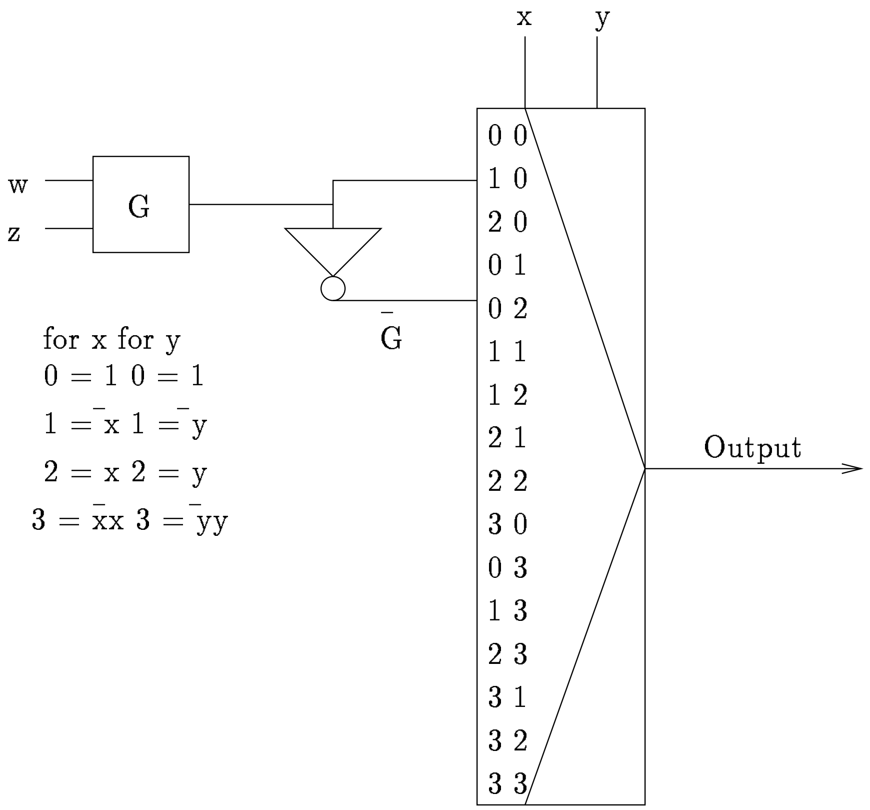


Figure 31: A Decomposition of a Fuzzy Function from Example 19.5 using a Fuzzy Multiplexer.

X_3 \ $X_1 X_2$	22	21	12	11	02	01	20	10
3			I		I			
2							I	
1			I		I		I	
0							I	

Figure 32: Fuzzy decomposition chart for 3 variables

00 *column or column heading 0 and the* 11 *column or column heading 3. A fuzzy switching function, normal in* Y*, is represented on a fuzzy decomposition chart by placing an* I *in the column and row representing each minterm of the function and leaving the other positions blank.*

Definition 19.4 *CAP is a* **unary operation** *denoted by the symbol* $\hat{}$ *such that:* $\hat{0} = 0$*;* $\hat{1} = 2$*; and* $\hat{2} = 1$*. This corresponds to permutation of input values.*

Definition 19.5 *Columns in a fuzzy decomposition map are said to be* **I-equivalent** *if and only if all I's or blanks in every row of one column is the same as another column. Every column is I-equivalent to itself.*

Example 19.6 In the fuzzy decomposition chart for 3 variables shown in Figure 32, where x_1 and x_2 are bound and x_3 is free, columns 12 and 02 are I-equivalent. Column 20 is not I-equivalent to any other column. And columns 22, 21, 11, 01 and 10 are all I-equivalent.

Definition 19.6 *The number of distinct columns in a fuzzy decomposition chart which are non-vacuous in I is called the* column distinction count *and is designated by c. (This count does not include blank columns.)*

Definition 19.7 *A fuzzy decomposition chart represents a* trivial case *of fuzzy switching function normal in* Y *to be decomposed if and only if all the columns containing an I are I-equivalent.*

Definition 19.8 *These functions are used for column compatibility later on in Definition 19.10.*

Let $T = \{0, 1, 2\}$.

COMPL 1 is a function $f : T^4 \longrightarrow T^2$

such that

$$f(a, b, c, d) = \begin{cases} \hat{b}\hat{c} & \text{if } b = c \\ bc & \text{otherwise} \end{cases}$$

x_j \ x_i	3	2	1	0
3				
2		a		
1				$\bar{a}$
0			$\bar{a}$	g

Figure 33: DIP 1 $a(x_i, x_j) = x_i x_j$, $\overline{a} = \overline{x}_i + \overline{x}_j$

x_j \ x_i	3	2	1	0
3				
2				$\bar{b}$
1			b	
0		$\bar{b}$		g

Figure 34: DIP 2 $b(x_i, x_j) = \overline{x}_i \overline{x}_j$, $\overline{b} = x_i + x_j$

COMPL 2 is a function $f : T^4 \longrightarrow T^2, T^2$

such that

$f(a, b, c, d) = bc, da.$

Note. When referencing these functions, ab and cd will represent column headings and should be arranged so that $ab < cd$ with respect to their decimal values.

Definition 19.9 *A* decomposition implicant pattern *(DIP) is one of the 5 implicant patterns shown in the S-maps of Figure 33, 34, 35 36, and 37 where a, b, c, d and e represent sub-map patterns; g is any sub-map pattern or can have no sub-map pattern; and no implicants appear in either the 3 row or the 3 column or any other place except g.*

The next definition is based on the representation of the DIP function on fuzzy decomposition charts as opposed to S-maps. This definition provides a procedure to determine whether or not a fuzzy function corresponds to some DIP.

x_j \ x_i	3	2	1	0
3				
2			c	
1				$\bar{c}$
0		$\bar{c}$		g

Figure 35: DIP 3 $c(x_i, x_j) = \overline{x}_i x_j$, $\overline{c} = x_i + \overline{x}_j$

x_j \ x_i	3	2	1	0
3				
2				$\bar{d}$
1		d		
0			$\bar{d}$	g

Figure 36: DIP 4 $d(x_i, x_j) = x_i \overline{x}_j$, $\overline{d} = \overline{x}_i + x_j$

x_j \ x_i	3	2	1	0
3				
2		$\bar{e}$	e	
1		e	$\bar{e}$	
0				g

Figure 37: DIP 5 $d(x_i, x_j) = x_i \oplus x_j$, $\overline{d} = x_i \odot x_j$

X_3 \ X_1X_2	22	21	12	11	02	01	20	10
3	I					I		I
2								
1	I					I		I
0	I							

(a)

X_3 \ X_1X_2	22	21	12	11	02	01	20	10
3		I	I	I				
2	I	I	I	I				
1	I	I	I	I				
0								

(b)

Figure 38: Illustration of compatible columns for a Fuzzy function

Definition 19.10 *The columns of a fuzzy decomposition chart are* compatible *if and only if:*

1. *for any ab column where $a = 0$ or $b = 0$, if column ab is I-equivalent to column ba then the only other column non-vacuous in I is the column represented by COMPL 1 (a, b, b, a)*

   or

   if column ab is I-equivalent to column $\hat{b}\hat{a}$ then
   the only other column non-vacuous in I is the column represented by COMPL 1 $(a, b, \hat{b}, \hat{a})$ or ab is I-equivalent to $\hat{a}\hat{b}$ and all other columns are vacuous of I's;

   or

2. *(if all ab columns where $a = 0$ or $b = 0$ are vacuous in I then) for any ab column, where $a \neq 0$ and $b \neq 0$, if column ab is I-equivalent to column $\hat{a}\hat{b}$ then*
   the only other columns non-vacuous in I are the columns represented by COMPL 2 $(a, b, \hat{a}, \hat{b})$ which must also be I-equivalent.

Example 19.7 In Figure 38a the columns are compatible. In Figure 38a from Definition 19.10.1, the ab column where $a = 0$ or $b = 0$, is the column 01. It is I-equivalent to ba column 10. Then the only other column non-vacuous in I is the column represented by COMPL 1 (a, b, b, a) = COMPL 1 $(0, 1, 1, 0)$ $= f(a, b, c, d) = \hat{b}\hat{c} = 22$ which is the only other non-vacuous column in I. The columns in Figure 38a are compatible from Definition 19.10.1.

Example 19.8 In Figure 38b the columns are not compatible. In Figure 38b column 21 is I-equivalent to 12, column 21 is I-equivalent to 11, and column 12 is I-equivalent to 11. From Definition 19.10.2 (if all ab columns where $a = 0$ or $b = 0$ are vacuous in I then) for any ab column, where $a \neq 0$ and $b \neq 0$, if column ab or 21 is I-equivalent to column $\hat{a}\hat{b}$ or 12 then the only other columns non-vacuous in I are the columns represented by COMPL 2 $(a, b, \hat{a}, \hat{b})$ = COMPL 2 $(1, 2, 2, 1) = f(a, b, c, d) = bc$, $da = 22$, 11 which must also be I-equivalent. But columns 22 and 11 are not I-equivalent so these columns in Figure 38b are not compatible.

Theorem 19.5 *If a fuzzy switching function in Y, $f_y(X)$, is decomposable on a $\{Y/Z\}$ partition of X, $Y = \{y_1, y_2\}$, then it represents a trivial case or it can be represented by a decomposition implicant pattern (DIP) with the bound variables as the outside row and column headings.*

As an example the labels in Figure 29, X_3 is the outside row and X_1 is the outside column. This next theorem is used to determine if a fuzzy switching function can be decomposed or not.

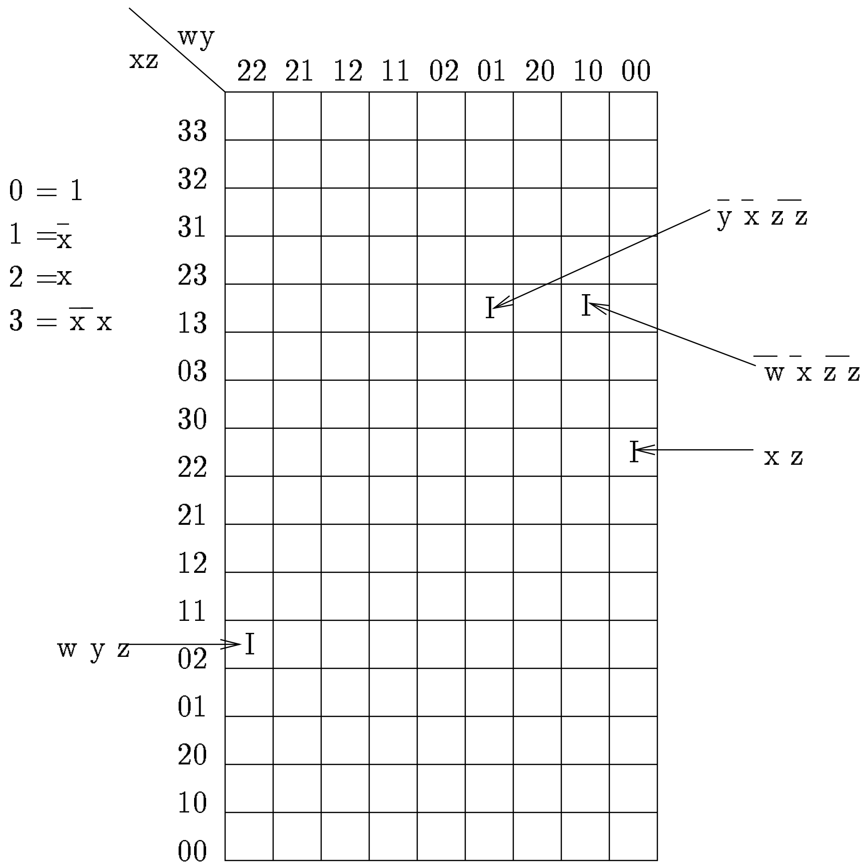


Figure 39: Fuzzy decomposition chart for $f(w, x, y, z)$

Theorem 19.6 *Let $\{Y/Z\}$ be a partition on the fuzzy set X where $Y = \{y_1, y_2\}$. A completely specified fuzzy switching function normal in Y possesses a fuzzy simple disjunctive decomposition with fuzzy bound set Y and fuzzy free set Z if and only if:*

1. *its Y/Z fuzzy decomposition map represents a trivial case, or*
2. *its Y/Z fuzzy decomposition map has columns that are all compatible from Definition 19.10 and the column distinction count $c \leq 2$.*

The function in Figure 38a is a decomposable function because the column distinction count is $c \leq 2$ and the columns are compatible while in Figure 38b the columns are not compatible so the function is not decomposable.

Example 19.9 Let

$$f(w, x, y, z) = \overline{x}\ \overline{y}z\overline{z} + xz + \overline{w}\ \overline{x}z\overline{z} + wyz$$

Let $Y = \{w, y\}$ and $Z = \{x, z\}$. Since $f(w, x, y, z)$ is normal in Y, its fuzzy decomposition map is as shown in Figure 39.

The column distinction count for this chart is 2. The columns 01 and 10 are I-equivalent and the only other column non-vacuous in I is column 22. Since COMPL 1 $(0, 1, 1, 0) = 22$, the columns 22, 01,

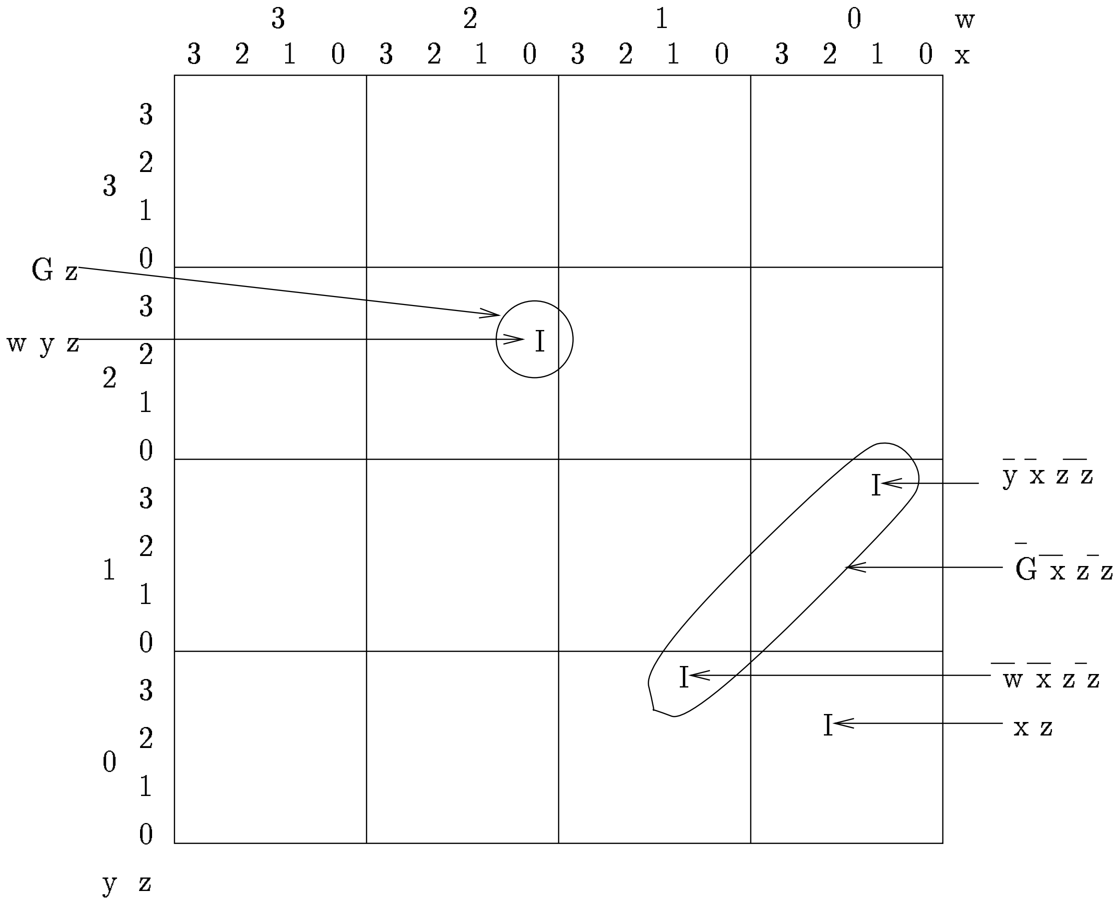


Figure 40: S-map of $f(w, x, y, z)$ representing DIP 1.

and 10 are compatible.
Therefore, by Theorem 19.6, $f(w, x, y, z)$ is decomposable with w and y as bound variables and x and z as free variables. Figure 40 shows the S-map of this function. According to Theorem 19.5, $f(w, x, y, z)$ can be represented by DIP 1 Figure 33 where the outside row y corresponds to x_i and the outside column w corresponds to x_j.
Using Table 6 this DIP implies:

$$G(w, y) = wy,\ \overline{G}(w, y) = \overline{w} + \overline{y}.$$

$$f(w, x, y, z) = (wy)z + (\overline{w} + \overline{y})\overline{x}z\overline{z} + xz$$

Therefore substituting $G(w, y) = G(Y)$ and $\overline{G}(w, y) = \overline{G}(Y)$

$$f(w, x, y, z) = F[(G(w, y), x, z)] = G(Y)z + \overline{G}(Y)\overline{x}z\overline{z} + xz$$

The decomposed fuzzy circuit is shown is Figure 41.
Now let $g = G(w, y) = wy$.

Then

$h(X)$	$\overline{h}(X)$	DIP
x_i	$\overline{x_i}$	-
$x_i x_j$	$\overline{x}_i + \overline{x}_j$	1
$\overline{x}_i \overline{x}_j$	$x_i + x_j$	2
$\overline{x}_i x_j$	$x_i + \overline{x}_j$	3
$x_i \overline{x}_j$	$\overline{x}_i + x_j$	4
$x_i x_j + \overline{x}_i \overline{x}_j$	$x_i \overline{x}_j + \overline{x}_i x_j$	5

Table 6: Two variable matching DIPs.

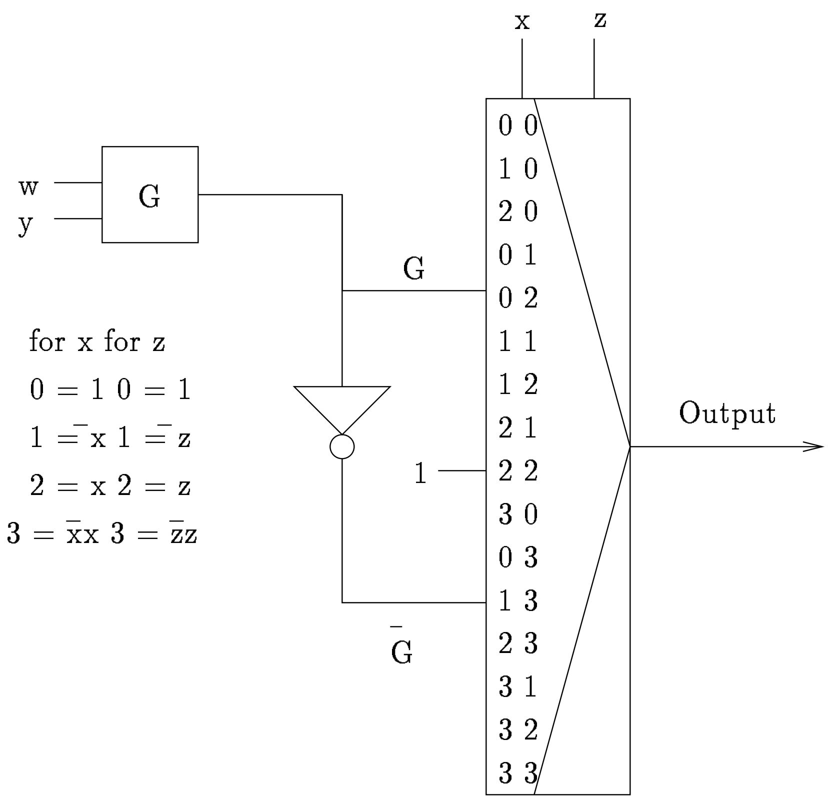


Figure 41: The decomposed fuzzy circuit

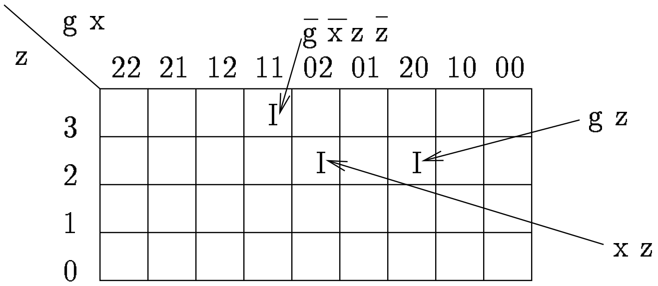


Figure 42: $F(g, x, z)$ map

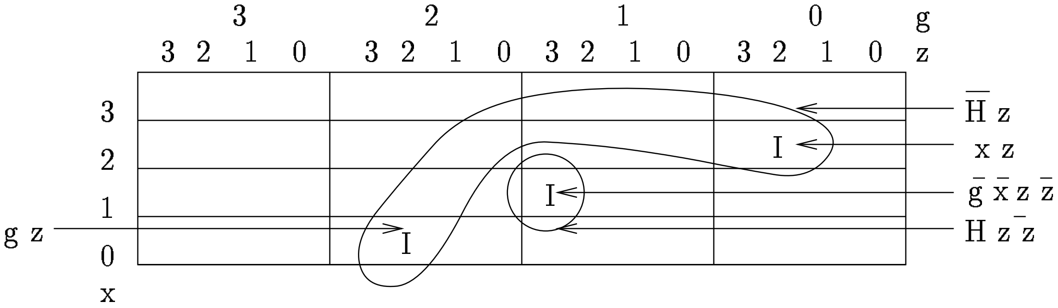


Figure 43: S-map of $F(g, x, z)$ representing DIP 2

$F(g, x, z) = gz + \overline{g}\ \overline{x}z\overline{z} + xz.$

The fuzzy decomposition map for $F(g, x, z)$ with g and x as the bound variables is shown in Figure 42. Again, the column distinction count is 2. The columns 02 and 20 are I-equivalent and the only other column non-vacuous in I is column 11. Since COMPL 1 (0, 2, 2, 0) = 11, the columns 11, 02, and 20 are compatible. Therefore, by Theorem 19.6, $F(g, x, z)$ is decomposable with g and x as bound variables and z as a free variable.

The S-map is represented by $F(g, x, z)$ is shown in Figure 43. This gives according to Theorem 19.5, $F(g, x, z)$ can be represented by DIP 2 Figure 34 where the outside row is x corresponds to x_i and the outside column g corresponds to x_j.
From the Table 6

$H(g, x) = \overline{g}\ \overline{x}$, $\overline{H}(g, x) = g + x$.

$g = G(w, y) = wy$, $\overline{g} = G(w, y) = \overline{w} + \overline{y}$

so by De Morgan's Law and distributive law we obtain:

$H(w, y, x) = \overline{(wy)}\overline{x} = (\overline{w} + \overline{y})\overline{x} = \overline{w}\ \overline{x} + \overline{y}\ \overline{x}.$

$f(w, x, y, z) = H(w, y, x) = (\overline{g}\ \overline{x})z\overline{z} + (g + x)z$

And therefore

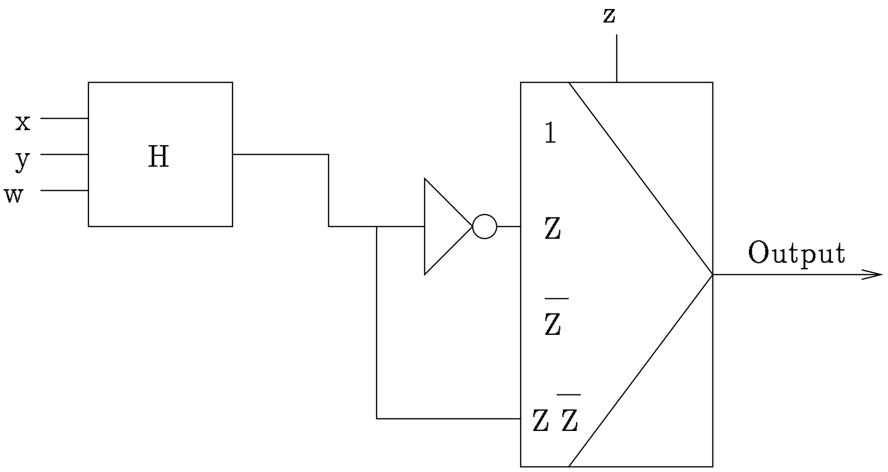


Figure 44: The decomposed fuzzy circuit

$$f(w,x,y,z) = F[H(w,x,y),z] = H(w,y,x)z\overline{z} + \overline{H}(w,y,x)z \quad [221]$$

The decomposed circuit is shown in Figure 44.
Substituting in H and G we get:

$$f(w,x,y,z) = F[H(w,x,y),z] = H(w,y,x)z\overline{z} + \overline{H}(w,y,x)z = (\overline{g}\ \overline{x})z\overline{z} + (g+x)z$$

$$f(w,x,y,z) = (\overline{g}\ \overline{x})z\overline{z} + (g+x)z$$

$$= (\overline{w}\ \overline{x} + \overline{y}\ \overline{x})z\overline{z} + (wy+x)z$$

This decomposed function takes 5 not, 6 min, 3 max operations while the original function, $f(w,x,y,z) = \overline{x}\ \overline{y}z\overline{z} + xz + \overline{w}\ \overline{x}z\overline{z} + wyz$, takes 6 not, 9 min, and 3 max operations. Decomposing this function makes the circuit simpler by one fuzzy not, and three fuzzy min operations.

19.6 Another Decomposition Example. Application of decomposition:

An example of using decomposition will start from a decision table. Such tables are used in medical diagnostics. A fuzzy decision table can be used to represent symptoms and test results which define a particular medical condition. A decision table can represent a fuzzy valued switching function. The decomposition could produce a function made up of variables representing symptoms and tests, or just of the tests, which could be evaluated by personnel other than a doctor. The doctor evaluates the patient's conditions without having to evaluate all the variables. Decomposition can be used to isolate functions which when evaluated could determine whether further tests are even necessary. Less tests are better for the patient by being less expensive and by avoiding tests which involve a high risk for the patient. For an example, Table 7 is a very simple version of a fuzzy decision table which represents conditions for diabetes. The variables represent the results of certain tests which are made and evaluated in order to determine if a person has diabetes. Let:

- Variable x be represented by the amount of sugar present in the urine based on a urine analysis.
- Variable y be represented by the amount of sugar present in the blood derived by a blood test.

Tests	Rule No. 1	Rule No. 2
Urine analysis		x
Blood test	y	
Glucose tolerance test	$\overline{z}$	$\overline{z}$
Positive Diabetes	[1 - 0.6]	

Table 7: Fuzzy decision table

- variable z be represented by the tolerance level of sugar by the body based on the results of a glucose tolerance test.

The rules that determine if a person has a diabetes are:

1. Rule 1. If a person has a high level of sugar in the blood and a low level of tolerance, this person may have diabetes.
2. Rule 2. If a person has a large amount of sugar in the urine and a low level of tolerance, this person may have diabetes.

The value in the table represents the grades of membership of the function which determines when someone has diabetes.
The fuzzy valued switching function which represents this decision table is

$$f(x, y, z) = y\overline{z} + x\overline{z}$$

If we let $G(x, y) = x + y$ then this function can be decomposed as follows:

$$f(x, y, z) = (x + y)\overline{z} = G(x, y)\overline{z} = F[G(x, y), \overline{z}]$$

Since the limit of f for the diabetes condition is a grade membership of 0.6, if $G(x, y)$ has a grade membership less than 0.6 then the glucose tolerance test need not be administered to deduce that the person does not have diabetes.[221]

Concluding, Example 19.9 shows that decomposition of a fuzzy function works correctly, but there is more considerations to be made in order to have a function decomposable. This makes decomposition even more rare to occur than a binary decomposition. When decomposition does work the amount of operations is reduced. We are investigating improvements and generalizations to the presented methods of fuzzy function decomposition.

20 The Decomposition of Continuous Functions.

The only paper about generalizing the AC decomposition to continuous functions is one by Ross et al. [546]. However, the method of orthogonal decision diagrams for continuous variables, developed by Pierzchala, Perkowski and Grygiel, should be also easily adaptable to AC decomposition, because of the close link of orthogonal and AC decompositions [510].

Perkowski, Pierzchala and Grygiel developed methods to realize Galois Field operations for small fields in analog/mixed *Field Programmable Analog Array - FPAA*. Paper [510] gives an example of realization of GF(4) operations. In this way, Galois Field type of expansions can be realized in hardware in an analogous way to Galois(2) expansions, i.e. AND/EXOR canonical forms. The regular array structure of the FPAA allows also to realize structures based on Orthogonal Expansions which are not Galois Expansions. The structure of the corresponding circuit resembles a PLA as well, and has columns corresponding to orthogonal functions over GF(2^n), where the functions are realized by cascades of Galois Addition and Galois Multiplication gates, and pass gates. Each column is Galois-multiplied by a constant, and next these values are Galois-added in rows. Thus, the column is a generalization of an AND-term of a PLA, and a row is a generalization of an OR-term (interestingly, a very similar array structure allows also to realize fuzzy logic circuits). Further, the trellis expansions of functions can be also generalized to continuous logic, where obviously variable repetition is required in a general case. Several other generalizations to other algebras, including fuzzy logic, as well as their realizations in regular array structures have been also described in another paper (not published yet). This is a fruitful area of our current research.

In another, yet unpublished paper, they observe that general decision diagrams, universal cells, regular structures and decompositions are so general, that they apply not only to binary logic, but also to multiple-valued logic, Galois Fields and other algebras (fields and some rings). They are also applicable to continuous, i.e. analog logic, for instance fuzzy logic. This helped to create the concept of universal analog-digital "tissue" which would allow to realize a very wide category of logics and other formal systems in hardware. A practical realization of these efforts is the concept of Field Programmable Analog (Mixed) Array which is for analog (mixed) circuits what FPGA is for binary circuits [510].

Perkowski and Hong [486] and Luba and Lasocki [397] presented a case of decomposing a function with various numbers of values in each variable. The recent research of the author is on the expansion of these methods to include also continuous and fuzzy variables. It is done using generalized decision diagrams that have nodes corresponding to binary, multiple-valued, continuous and fuzzy variables.

21 The Generalized Decomposition.

Although Pattern Theory (PT) group interests are only in combinational logic decomposition, from the mathematical point of view the decomposition of Finite State Machines is a close issue. Historically, the decomposition methods have been applied to state machine design for many years, starting with works of Hartmanis and Stearns in 1960's and 1970's. Hartmanis and Stearns created a very general and computationally convenient mathematical apparatus, called *"Partition Algebra"*, and proved powerful theorems about parallel, cascade and other decompositions of finite state machines. It was, however, observed that these methods are inefficient for practical state machines, and rarely a good decomposition exists for a random machine. In general, the criticism and the reception of these methods in industry were initially similar to that of the reception of AC decomposition in case of combinational circuits. However, there has been recently an increased interest in state machine decomposition methods in companies such as Philips, Siemens, and Synopsys and in top research universities. Moreover, there exists a feeling that new decompositions and techniques can be developed that will be useful for decomposing both switching functions and state machines [312, 313].

The partition algebra has been used by Luba et al for combinational decomposition, leading to a totally new approach and practical results of a very good quality. Recently, a new generalized theory of decomposition of both state machines and combinational logic has been developed by Jozwiak et al. [312, 313]. It is yet uncertain to us, whether the theory of Jozwiak does indeed bring new practical decompositions in case of combinational circuits (other than a common presentation framework with sequential circuits). However, it definitely allows to create much more general decompositions of sequential machines by considering more decomposition types, including some quite complex new types that are able to perform decomposition ("splitting") of internal states.

It is possible to extend the current model of Pattern Theory Group to state machines. This would be the simplest and most natural extension of the current model, because the state machines include the combinational logic, and there is no popular model of computability model that would be located in the hierarchy between the two. If ever the Pattern Theory group would decide to make a state machine generation their most basic model of algorithm Machine Learning then the classical model of state machine decomposition would be is a good candidate. Especially the "Generalized Decompositions" of Jozwiak are good candidates because of their similarity to the current decomposition model investigated by the Pattern Theory group, and with respect to the use of the generalization of Luba's partition-based decomposition model at PSU.

22 Representation of Functions.

In this and following sections, we will present partial problems of decomposition. The subsequent four sections will discuss the four most important issues:

1. Representation of problems.
2. Variable partitioning.
3. Column minimization.
4. Sub-function encoding.

As we observed in previous sections, many decomposition programs were successful due mainly to some new approaches to problem representation: cube calculus, Reed-Muller forms, Binary Decision Diagrams, Walsh Transforms, or partition calculus based representations. We hope that these were not the last words in representation and that among many new and interesting new general representations of Boolean functions, there are some that may be used to create superior programs for Boolean Decomposition.

Because we believe that representation is THE most important aspect of successful decomposers, we will devote more attention in this section to the representation problem. Below, we will present the main research results in sections corresponding to various representations that have been already investigated for some other applications. Because Boolean methods are usually slower than algebraic methods, an effective representation and efficient manipulation of functions is the key to the success of these methods.

For representing Boolean functions, the following methods have been used in decomposition programs:

1. Karnaugh maps: Ashenhurst, Curtis, Ross et al, Torkelson, Luba.
2. Cubes: Hwang and Owen, (IEEE Transactions on CAD, May 1992), Karp [549], Perkowski/Brown [486], Wan/Perkowski [678], Bibilo [70], Steinbach [83] (a variant called TVL - ternary vector lists that has a different coding of symbols 0, 1, and X and uses disjoint cubes).
3. Canonical Positive Polarity Reed-Muller Forms: Shen and Kellar, Trachtenberg and Varma.
4. Walsh Transforms: Karpovsky, Trachtenberg and Varma, Stankovic.
5. OBDDs: Chang and Marek-Sadowska [132], Lai, Pedram, and Vrudhula [351], and Sasao [580].
6. Edge-Valued Binary Decision Diagrams (EVBDDs) by Lai, Pedram, and Vrudhula [351].
7. Partitions of minterms: Luba, Selvaraj.
8. Partitions of cubes: Luba [398], Selvaraj [601, 602].

The following representations of binary functions are close to the above and were shown to have some advantages in other problems, so they remain possible prospective candidates:

1. Canonical AND/EXOR and Orthogonal forms, other than the Canonical Positive Polarity Reed-Muller Forms.
2. Spectral Transforms: all AND/EXOR transforms, Arithmetical Transform, Adding Transform, Orthogonal Binary Transforms, Orthogonal Multiple-valued Transforms.
3. Decision Diagrams: Zero-Suppressed, Moment, Functional, Kronecker, Orthogonal, Multiple-Valued.

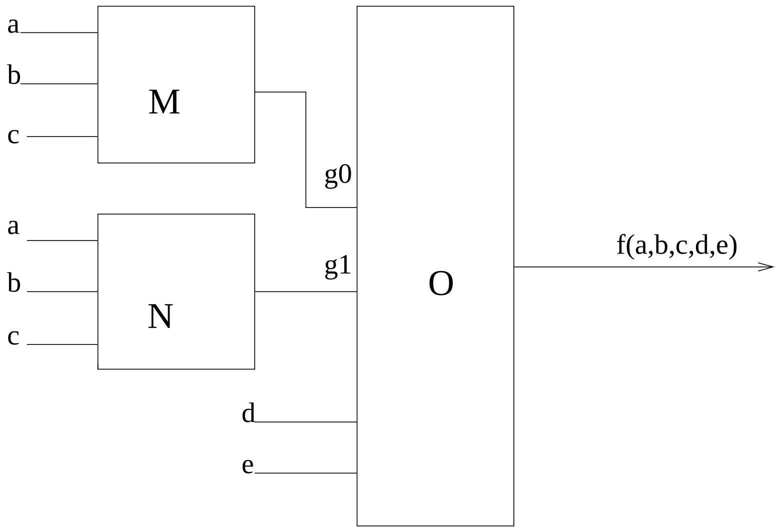


Figure 45: A schematic diagram of the decomposition

4. Array of Cubes representing an ESOP.

For representing multiple-valued functions the following methods have been used:

1. Maps, expressions Walliuzzaman and Vranesic, [677].
2. Expressions, Wojcik and Fang, [211].
3. Maps, Abugharbieh, [7].
4. partitioned minterms: Luba, [398].
5. multiple-valued cubes and minterms: Sasao 1989, [579].
6. Multiple-Valued diagrams [579].

Single output functions have been discussed in: Ashenhurst, Curtis, Ross, Steinbach. Multi-output functions have been presented in: Luba [398], Pedram [351], He and Torkelson [285], and Karp [318].

In the following subsections we will present the most prospective representations.

22.1 Various Types of Decision Diagrams.

Reduced, ordered Binary Decision Diagrams (BDDs) provide a compact and canonical representation of Boolean functions. These data structures are used in the decomposition program by Lai, Pedram and Vrudhula [351]. When the function is incompletely specified, there are two ways to represent it using BDDs. The first is to generalize the BDD to 'Ternary DDs' that have three terminal nodes, standard nodes 0 and 1, and a '*' node corresponding to the don't care value of the function. This method was applied by Lai, Pedram and Vrudhula [351].

The other method, as far as we know not used in any program yet, would be to represent functions 'q' and 'r' of XBOOLE or functions ON and OFF of TRADE, each with one standard BDDs. In case of multi-output functions a shared OBDD can be used, or any other shared diagram (Functional, Zero-suppressed, with negated edges, Kronecker, etc). This would allow to use standard DD packages and 're-use' all (or most) ideas of XBOOLE and TRADE, which propose two different approaches, that can be now totally unified on an algebraic level.

Pedram et al use also their new data structure, called Edge-Valued Binary Decision Diagrams (EVBDDs) [350]. EVBDDs provide a representation of Boolean functions over the integer domain and have been shown to be useful for verification of arithmetic functions. Another similar and even more powerful decision diagrams have been recently created by Brayant and Chen, 1994, and called Binary Moment Diagrams. These topics are the areas of recent research.

22.2 Approach of Lai, Pedram and Vrudhula based on Binary Decision Diagrams.

Lai, Pedram and Vrudhula proposed in DAC '93 one of the most successful approaches to AC decomposition [351]. Their approach is applicable to both disjunctive and non disjunctive decompositions, both completely and incompletely specified Boolean functions, single-output and multi-output.

Their general theory uses a new algorithm, based on both BDD and EVBDD representations (two variants), for generating the set of all bound variables that make the function decomposable.

A function $f(x_0, \ldots, x_{n-1})$ is said to be decomposable under bound set $\{x_0, \ldots, x_{i-1}\}$ where $0 < i < n$ if f can be transformed to $f(g_0(x_0, \ldots, x_{i-1}), \ldots, g_{j-1}(x_0, \ldots, xi-1), x_{i-k}, \ldots, x_{n-1})$. Then, if $k = 0$, the function is said to be disjunctively decomposable, otherwise it is non disjunctively decomposable.

Consider a simple example of a function $f(a,b,c,d,e)$. Let *bound_set* $= \{a,b,c\}$ and *free_set* $= \{d,e\}$. Here $k = 0$ hence a disjunctive decomposition is applied. Figure 45 shows a schematic diagram of this decomposition.

If two or less than two g functions exist for the bound set (a,b,c), the decomposition exists. Else, there is no decomposition. Ordered Binary Decision Diagrams (OBDDs) are used to represent functions.

Notations

- The left edge of the BDD represents the true edge or 1

- v represents function $f(x_0, \ldots x_{n-1})$ and BDD rooted by node v

22.2.1 Disjunctive Decomposition on BDDs

In this section disjunctive decomposition with BDD's used as the method of representation will be explained and the case of completely specified functions is presented.

Example 22.1

Let $f = x_0x_1x_2x_3 + x_0x_1x_2\overline{x}_3\overline{x}_4 + x_0x_1\overline{x}_2\overline{x}_4 + x_0\overline{x}_1x_2\overline{x}_4 + x_0\overline{x}_1\overline{x}_2\overline{x}_3 + \overline{x}_0x_1x_2\overline{x}_4 + \overline{x}_0x_1\overline{x}_2\overline{x}_3 + \overline{x}_0\overline{x}_1x_2x_3 + \overline{x}_0\overline{x}_1x_2\overline{x}_3\overline{x}_4 + \overline{x}_0\overline{x}_1\overline{x}_2\overline{x}_3$. The corresponding decomposition chart is as shown in Fig 46.

Looking at Figure 46 we can see that there are 3 distinct columns. Hence we know that there is a simple disjunctive decomposition with the chosen bound and free sets. But how can this be found by looking at the BDD?

The Pedram's method is this:
Draw the BDD with such a variable ordering that the variables from the Bound Set are on the top of the BDD, and then they are followed by the variables from the Free Set. Figure 47 shows the BDD of our example.

Figure 48 shows how the columns of the decomposition chart correspond to paths along the BDD. If one looks at path 101 along the BDD it results in $\overline{x}4$ which is the same as column 101. This illustrates the correspondence of the decomposition chart and the BDD. Figure 49 explains what a cut set is.

One important assumption of Pedram's method is that the variable ordering of the BDD must correspond to the bound set. If it does not correspond, a rotate operation must be used to move the bound variables to the top.

x3x4 \ x0x1x2	000	001	010	011	100	101	110	111
00	1	1	1	1	1	1	1	1
01	1	0	1	0	1	0	0	0
10	0	1	0	1	0	1	1	1
11	0	1	0	0	0	0	0	1

Figure 46: A Decomposition Chart to Example 21.2

Now a complete example will illustrate how to perform a disjunctive decomposition on BDDs.

Example 22.2 Let us assume a Boolean function of 5 variables f(a,b,c,d,e). Let {a,b} be the bound set and {c,d,e} be the free set. Assuming a simple disjunctive decomposition with two g functions, the decomposition would look like shown in Figure 50.

M and N are called the predecessor blocks and O is called the successor block. The entire process is described in Figure 51, 52 and 53.

22.2.2 Finding predecessor and successor blocks.

The steps involved in finding predecessor and successor blocks are the following:

STEP 1: Construct a BDD.

STEP 2: Find the best cut of the BDD.

STEP 3: Encode each node in the cut set, i.e g,h,i

Let g=10, h=01, i=00. Since we have used two bits for encoding we have two g functions $g0$ and $g1$ as shown in the Figures 52, 53 and 54.

At this point we know that the successor function has inputs of $g0, g1, x3, x4$. Hence the original cut set of the BDD will remain the same, and variables g0 and g1 will stand on the top of the BDD. This is shown in Figure 54.

Both Ashenhurst and Curtis decompositions are easily found from the value of the cut. This technique is next extended to a non disjunctive decomposition, and to multiple-output functions in a standard way.

Pedram's method was also applied to incompletely specified functions. It is interesting how well is this method suited for decomposition of multi-input, multi-output functions with a large number of don't cares? Figure 55 shows an example of an incompletely specified function, and Figure 56 shows the corresponding 'ternary' BDD with 3 terminal nodes - 1,0 and X.

It can be seen from this BDD that what is needed is some kind of technique to combine compatible nodes to the smallest number of nodes. For instance, familiar graph coloring or set covering methods can be used, so that the nodes of the BDD will be combined, just like combining columns in the Karnaugh

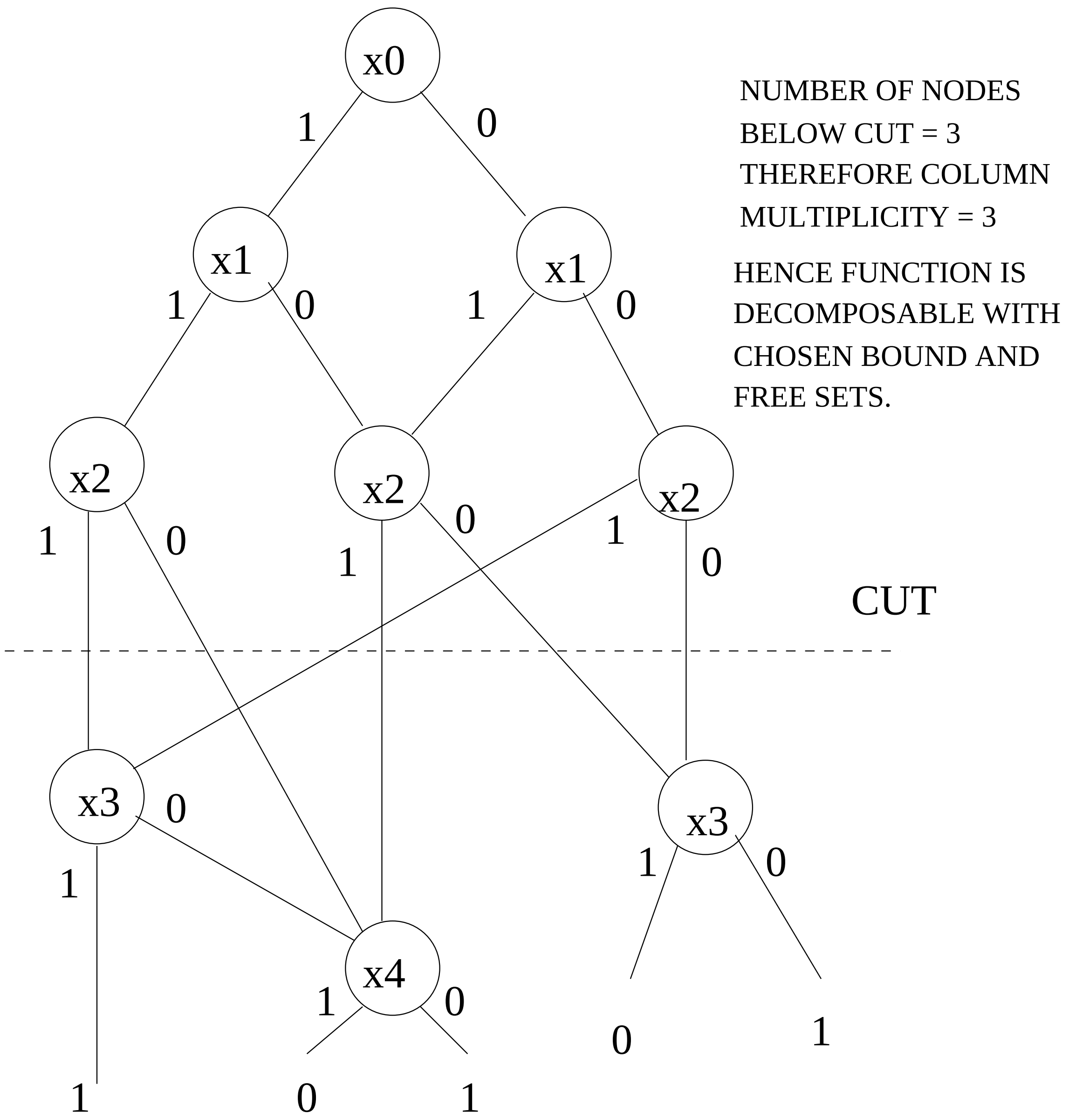


Figure 47: BDD of the decomposition chart

x3x4 \ x0x1x2	000	001	010	011	100	101	110	111
00	1	1	1	1	1	1	1	1
01	1	0	1	0	1	0	0	0
10	0	1	0	1	0	1	1	1
11	0	1	0	0	0	0	0	1

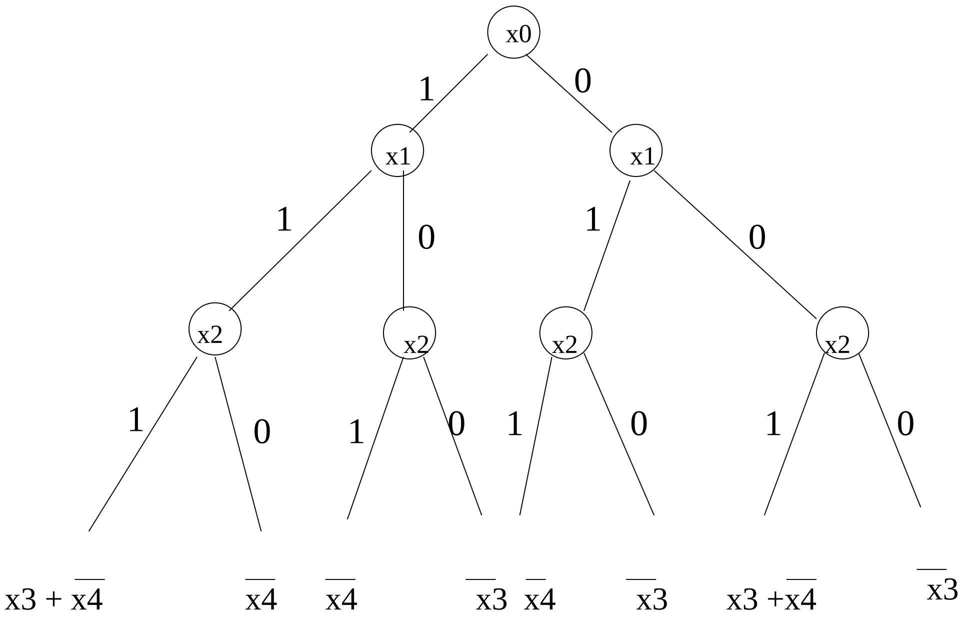


Figure 48: Correspondence between columns of the decomposition chart and BDD

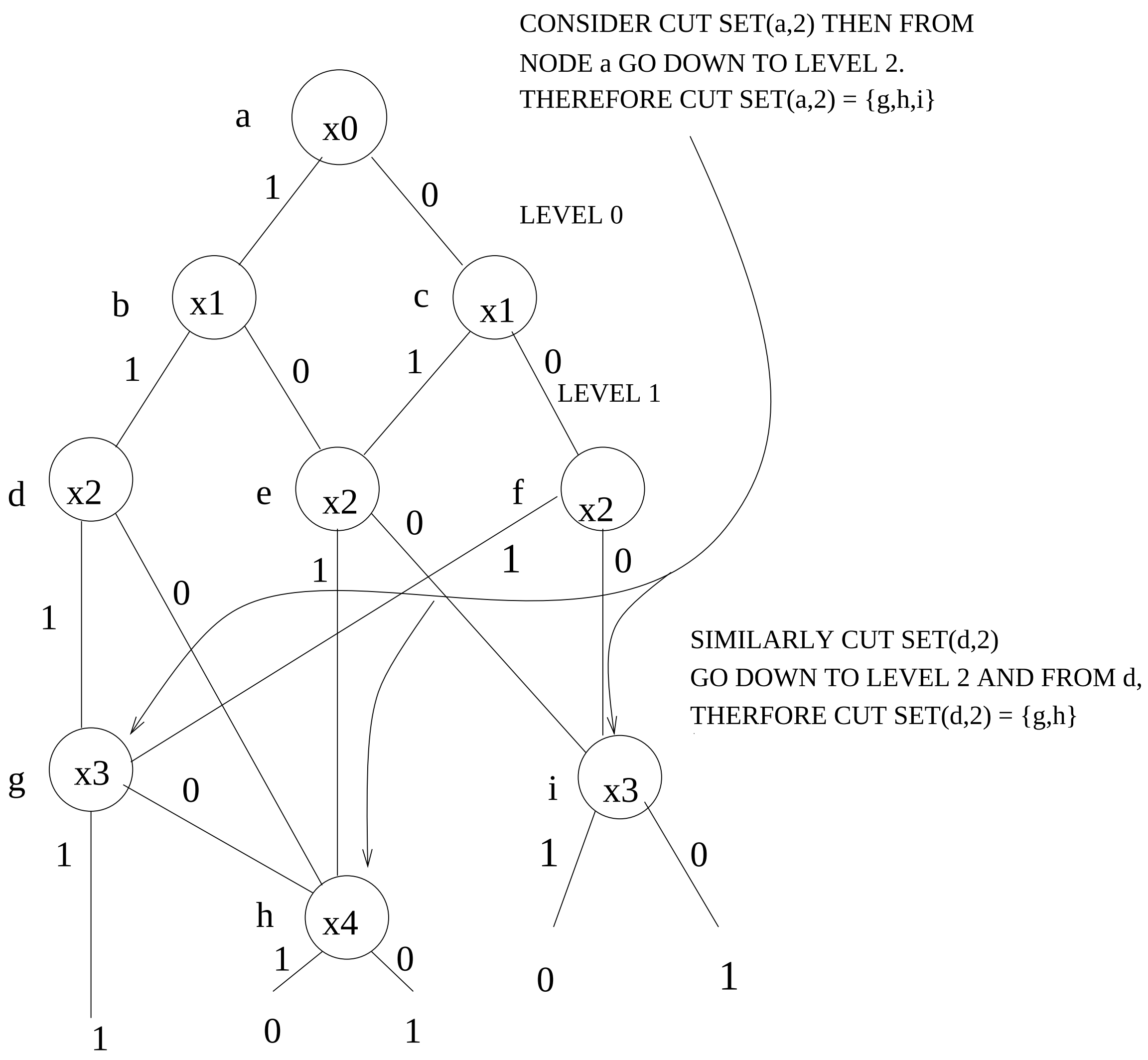


Figure 49: Explanation of a cut set

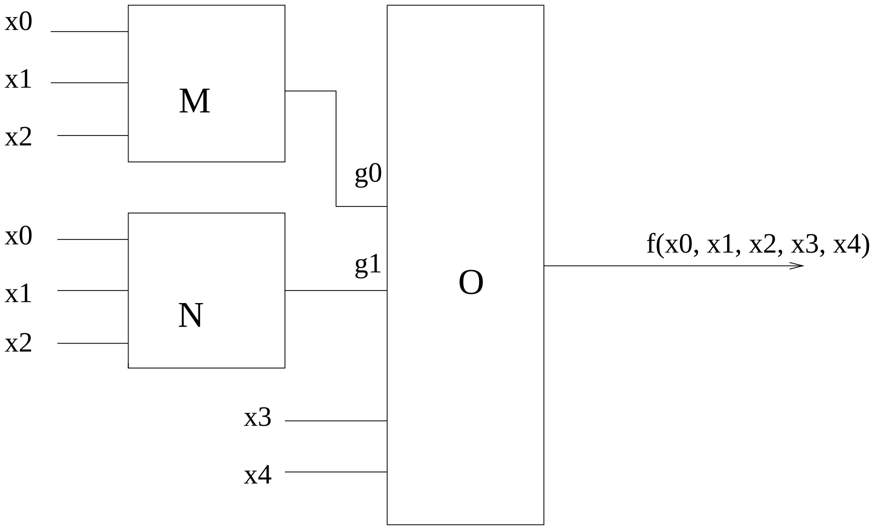


Figure 50: Structure of the decomposition with two g functions

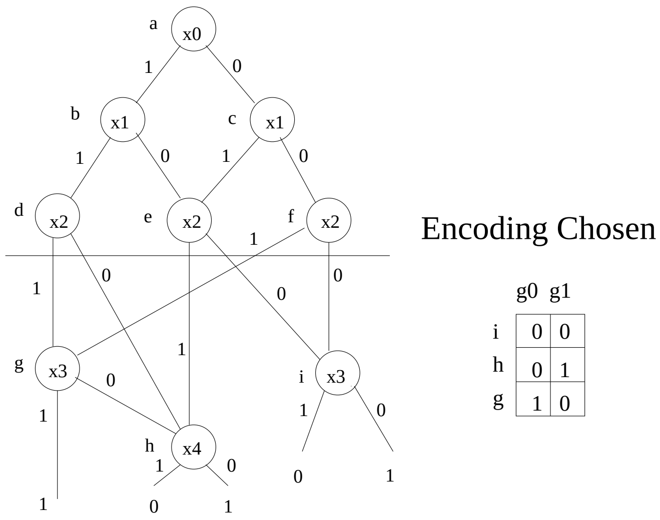


Figure 51: A BDD Example

so now we draw the BDD for g0 by making the nodes
g=1 h=0 i=0

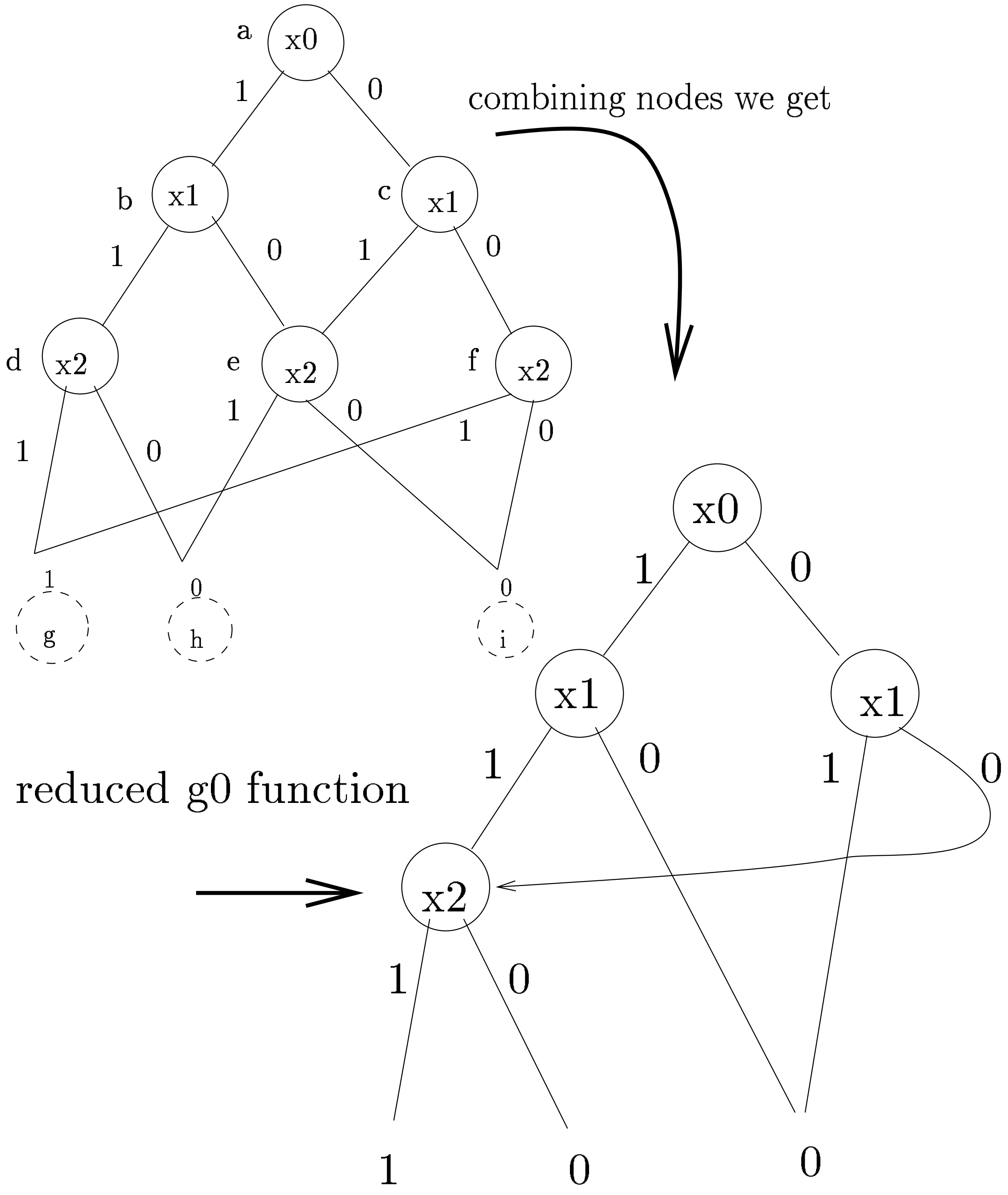


Figure 52: Finding the predecessor block

Similarly we get the g1 function by encoding

i=0 h=1 g=0

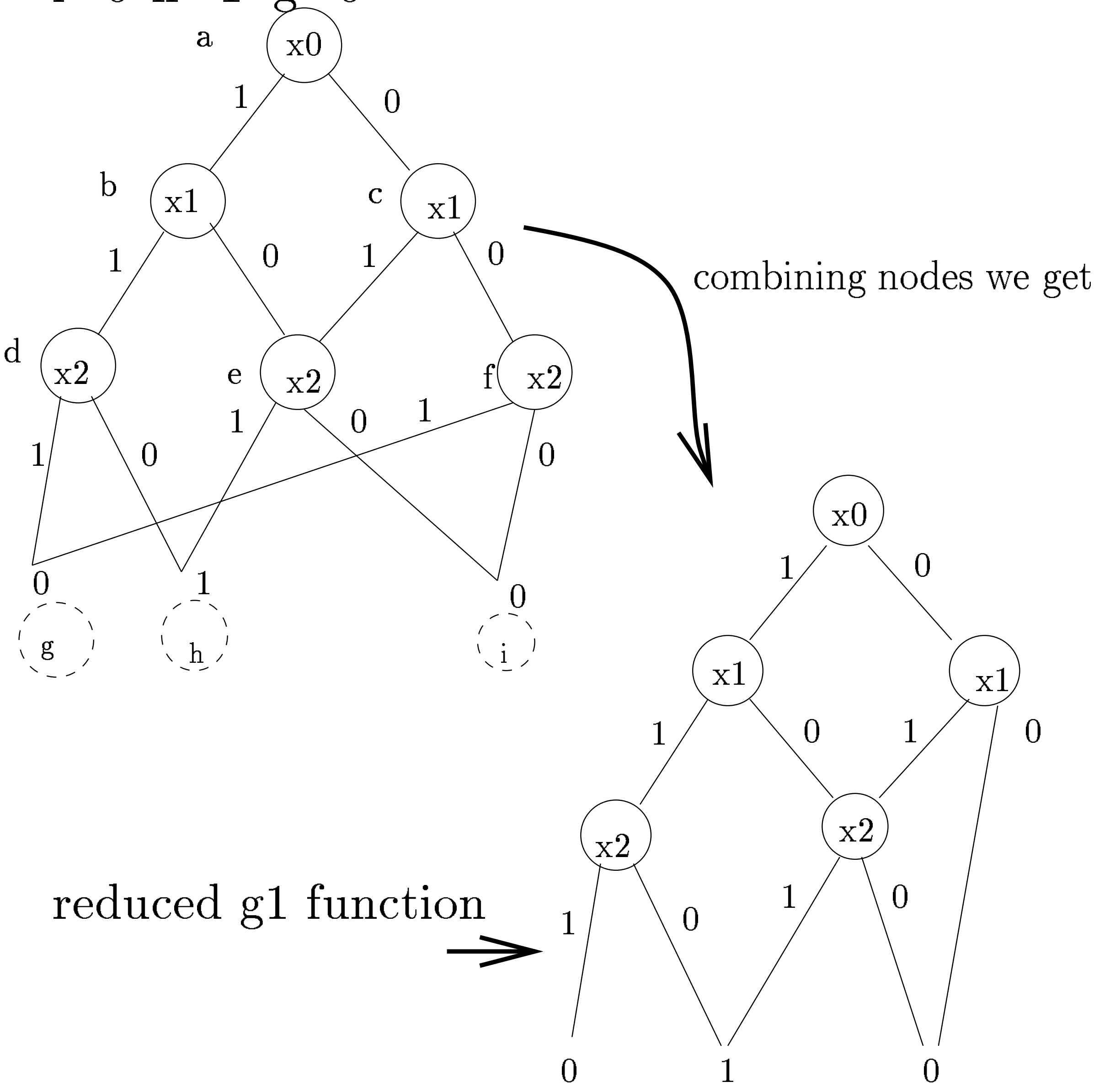


Figure 53: Finding the predecessor block

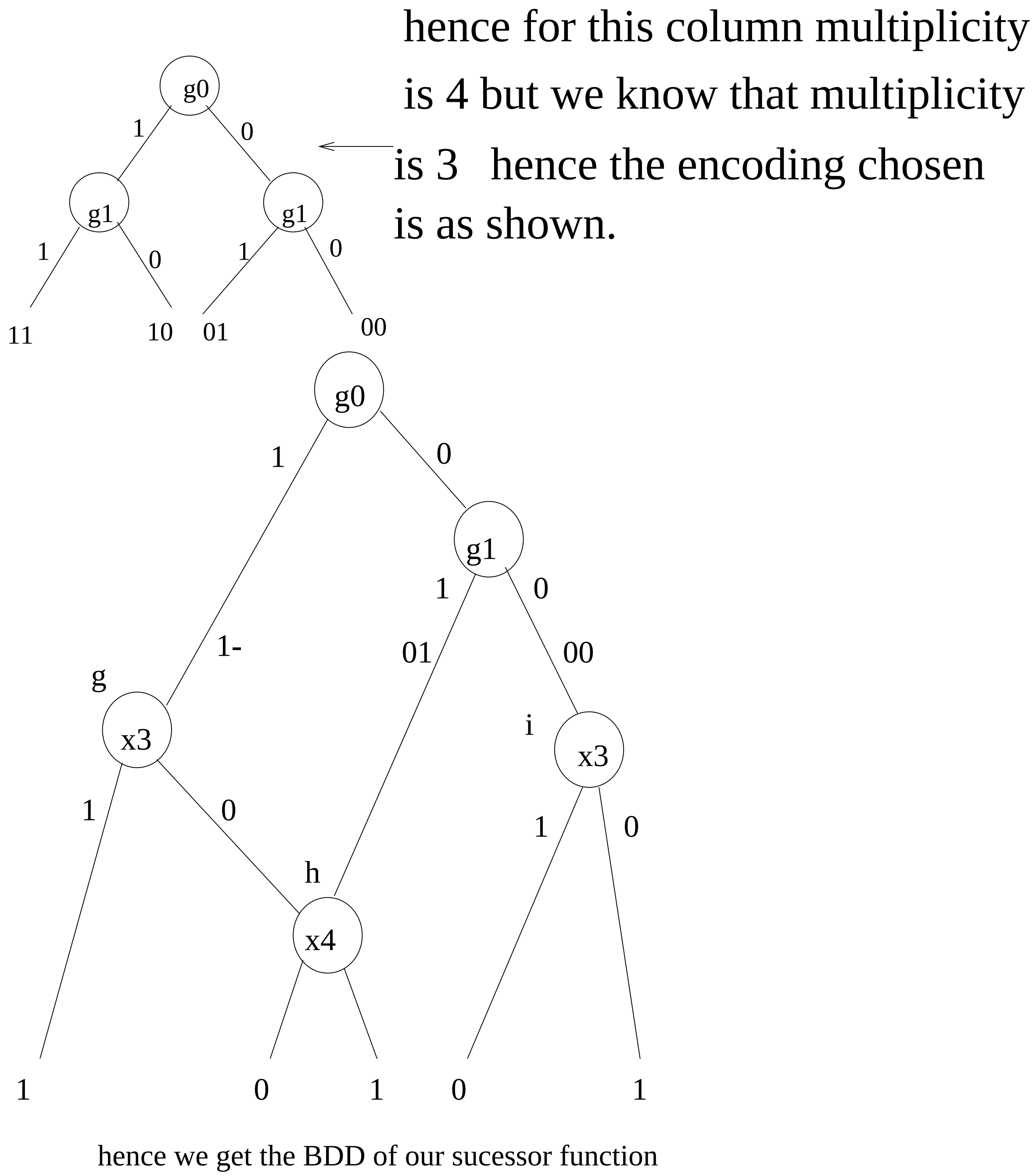


Figure 54: Finding the successor block

ab \ cde	000	001	010	011	100	101	110	111
00	x	1	0	1	1	0	1	0
01	x	1	1	x	1	1	x	x
10	x	0	1	0	0	1	0	1
11	x	1	x	x	0	x	x	x

Figure 55: Decomposition chart of an incompletely specified function

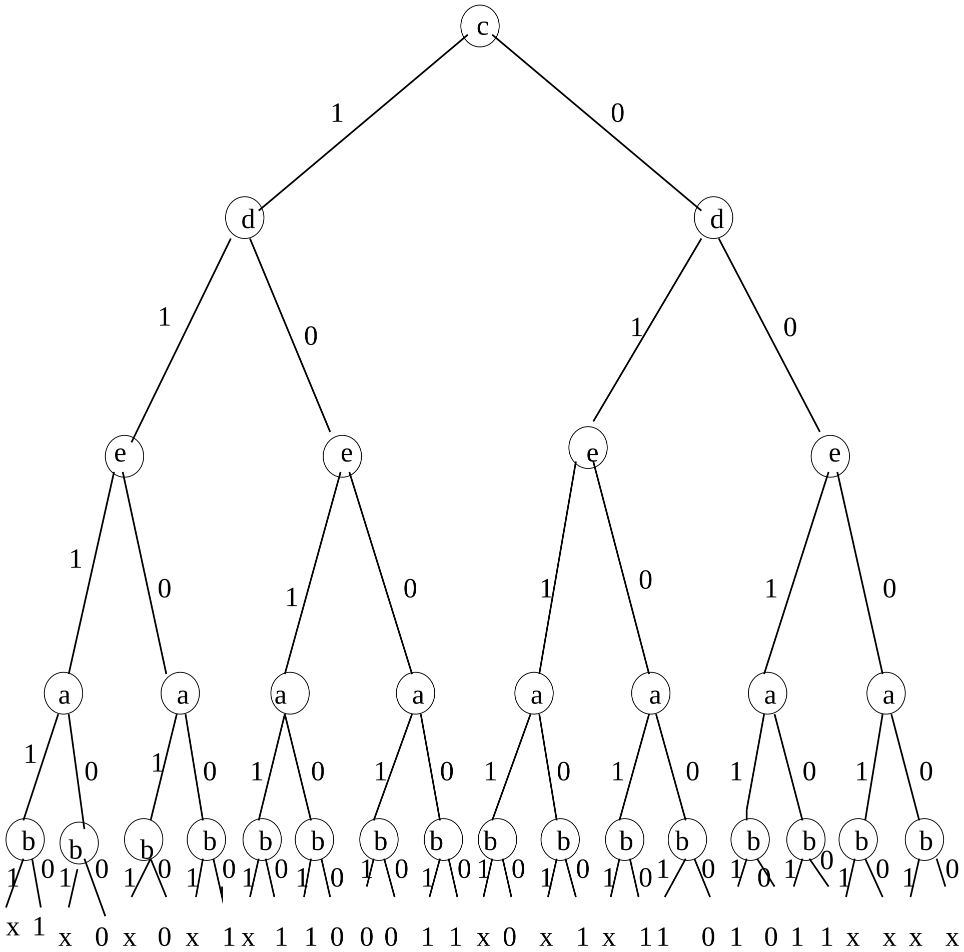


Figure 56: BDD with 3 nodes for an incompletely specified function

map so as to reduce the column multiplicity. One possibility would be to create an Incompatibility Graph and apply graph coloring, another possibility is to create Compatibility Graph, maximum cliques, and solve the coloring table. One more approach would be to create a new method performing 'column minimization' in a smart heuristic way directly on the BDD.

When functions are incompletely specified, an analogous difficulty occurs as in Roth-Karp and Perkowski-Brown-Wan algorithms. The authors propose a set covering algorithm, but do not describe it in detail, so it is perhaps a standard one from MIS tools. The encoding method is also not presented or even mentioned.

While Roth-Karp use NP-hard problem, and Perkowski use NP-hard graph-coloring problem, Lai et al are faced with performing analogous task on BDDs. Since BDD can represent only a completely specified function, they first generalize the BDD to a graph that has three types of terminal vertices: 0, 1, and DC, to represent the constant function 'don't care'.

Next, they compute the cut-set as before. Since each node in cut-set may represent an incompletely specified function, they need to compute the compatibility between any two nodes in the cut-set so that a minimal k can be found. The determination of compatibility between two BDD nodes, or the compatibility between their corresponding functions, is carried out by algorithm *is-compatible.* After this step, the construction of computability graph and the computation of minimum clique cover is the same as in the Roth-Karp algorithm.

From the point of view of the size of functions that can be handled, the Pedram's method is the top decomposition method in the world today. They are able to solve all decomposable forms for functions such as vg2 which has 25 inputs and 8 outputs, or even e64 that has 65 inputs and 65 outputs. These are clearly one of the largest examples of AC decomposition ever achieved. Perhaps, THE largest. However, for the predecessor and successor functions there is no encoding scheme used so the random encoding chosen is not guaranteed to be a good encoding.

Concluding, one can state that the only innovative idea of their paper is the BDD representation and the concept of cut-set in BDD, as well as using variable sifting techniques to find the good bound sets.

Hence Pedram's method can be improved by combining Luba's method, Steinbach's method, or any other method that proposes new decompositions, conditions, algorithms or heuristics. Also, if some encoding method is used it will greatly improve the quality of decomposed blocks.

22.3 New Classes of AND/EXOR Decision Diagrams.

The Direct Acyclic Graph (DAG) based representation and decision diagrams which are more general than BDDs have been recently created and applied. While Hurst in 1985 [302] only mentioned a similar approach, recently both a theory and several synthesis tools have been developed in our group, [695, 591, 590, 696], [697, 567, 568, 569, 698, 179, 498, 499], and also by Sasao [579] Steinbach [55], Rosenstiel [326, 598], and Marek-Sadowska [132, 561].

Perkowski et al [590, 592, 498] formulated the new class of Decision Diagrams, called *Kronecker Decision Diagrams (KDD).* These Decision Diagrams are the generalization of the popular Binary Decision Diagrams (Bryant, 1986) and Functional Decision Diagrams, Kebschull, Schubert, and Rosenstiel, [326], and are more compact than both of the former Decision Diagrams. The method is much more efficient than FDDs, and can be applied to very large functions given in multi-level net-lists.

In addition, KDDs, similar to BDDs and FDDs, provide a canonical representation of functions and can be applied in many areas. Currently, BDDs have been used in many applications in logic synthesis, verification, testing, modeling and simulation. KDDs, while being more compact, can also be utilized in many of such applications and thus can provide a major improvement over the current techniques in these areas. They can also drastically cut on the number of nodes in the Decision Diagram for very large functions that up to now have not been able to be represented by BDDs.

For any Decision Diagram to prove to be useful, the compactness of the representation has to be compared with the ease of construction and manipulation. A package for representation and manipulation of functions has been developed [179, 55], which shows the compactness of the KDDs together with ease of manipulation and construction. It has been shown that for the standard, hard, benchmark examples, KDDs are on the average 35% more compact than Binary Decision Diagrams, with some reductions of up to 75% being observed. The minimization scheme is based on the state of the art minimization schemes for BDDs, namely, dynamic variable ordering with sifting algorithm (Rudell, IWLS 1993, [558, 559]). Here the sifting is performed for both the order of variables as well as the type of decompositions.

Furthermore, a class of functions was presented for which both BDD and FDD representations are exponential in size but KDD is of polynomial size [179]. This property together with the canonicity and the ease of construction and manipulation distinguishes the major significance of the KDDs. Although, thanks to their canonicity, the KDD seem to be the most prospective of the introduced by our group diagrams as a general purpose representation of Boolean Functions, we found that other diagrams are best for mapping to fine-grain FPGAs. Recently Ho and Perkowski developed a concept of Free Kronecker Decision Diagrams [499]. They showed experimentally that it gives very good results on mapping to Atmel 6000 series FPGAs. The difference between the Kronecker Functional Decision Diagrams and the Free Kronecker Decision Diagrams (FKDD) is that in the former diagrams all nodes at certain level have the same variable and the same (one of three) expansion type. We generalized this concept to *'Pseudo-Kronecker Decision Diagrams'* where in each level we have still one variable, but all three types of expansions are possible in nodes. This has been generalized even further, to FKDDs, where there can be many different variables in a level. The tree or a graph is thus no longer 'ordered'. Although it is more difficult to create a program for generating this type of diagrams, they reduce significantly the number of nodes, and result in better mappings to FPGAs. It is possible to make these free diagrams canonical, implement operations on them, and treat them as the general-purpose function representations.

In another development, Perkowski et al observed that the totally symmetric Boolean functions can be realized in totally symmetric and regular decision diagrams that have only local connections. They call such flat and regular diagrams, the *trellis diagrams*. A question can be now asked: 'can the functions that are not symmetric be realized in trellis diagrams?' Interestingly, the answer is 'yes', if the variables of the diagram are repeated. This leads them to the concept of 'diagrams with repeated variables'. The current work is on the fundamental problem to create such diagrams for a given Boolean function, namely, what should be the (possibly repeated) order of input variables, so that the total number of variables (and the lattice area) will be minimized.

Finally, new classes of diagrams, called canonical *'Boolean Ternary Kronecker Decision Diagrams (BKTDD)'* have been created, that can theoretically be better than all other diagrams [498]. For selecting the expansion type, one has 1 choice in BDDs and original FDDs, 2 choices in modified FDDs, 3 choices in KDDs, and 12 choices in BKTDDs. There is no danger of loosing good choices in BKTDDs, since the three standard expansions of the KDD are still available in them. Therefore, the exact BKTDD is always not worse than the exact KDD.

Some of the introduced families, such as the KDDs, include all types of nodes from the BDDs and FDDs. It is then obvious that KDD diagrams are always more compact. The important questions are only: 'how much percent decrease in the number of nodes can be obtained by constructing KDDs for industrial benchmark logic functions? Can one represent with the KDDs some large functions than are not able to be represented by BDDs?' The numerical results are very good for some benchmarks, especially for incompletely specified and arithmetical functions. For the functions with a large number of input variables our algorithm can still be significantly improved.

KDD diagrams allow also to represent some especially constructed large functions that can not be represented by BDDs and FDDs. An example of such a function is given in (Becker, 1993). This is one of the areas of the current research.

22.4 Canonical Fixed Polarity Reed-Muller Forms

As presented in one of earlier sections, a very successful approaches to decomposition by Shen and Kellar [607, 608, 609] and Trachtenberg and Varma, 1989 [701] were based on Positive Polarity (Canonical) Reed-Muller Forms. There exists a family of more general forms, called Canonical Fixed Polarity Reed-Muller Forms (FPRMs), that are, however, very close to Positive Polarity RM Forms. They find several applications in logic synthesis, and allow for more compact function representation.

They were used as internal representation of Boolean functions in design automation systems; they were used as a convenient model to discuss some important algorithms, they were used as an aid in studying function properties and function classification (Marek-Sadowska, 1994 [561]); and finally they were useful to generate efficiently other types of EXOR Circuits, such as ESOPs, or Generalized Reed-Muller Forms.

FPRM forms can be also relatively efficiently realized in cellular arrays. Their realizations are similar to PLAs and are based on the concept of regularity to combine the synthesis and physical placement and routing stages. In many applications the AND/EXOR realizations of the circuits require less layout area than their AND/OR counterparts [494, 622, 623, 579]. The AND/EXOR PLAs often require fewer products than AND/OR PLAs (Sasao and Besslich, 1990 [578]). The major advantages of this logic stem from the information processing capabilities of EXOR gate.

For the specific AND/EXOR realizations, efficient heuristic algorithms for Fixed Polarity Reed-Muller canonical forms have been created [567, 569]. Fast techniques for identification of a minimal realization of Boolean functions in fixed polarity AND/XOR canonical forms were introduced. This scheme is then used in a Generalized RM canonical form minimization technique. Identification of a minimal realization would reduce the number of cells needed in the regular array implementation. Fast algorithm allows to approximately minimize functions larger than in the methods from the literature.

Many of the theorems and properties found in [567, 569] for FPRM forms are of general nature and can find applications in EXOR Logic Synthesis outside the topic of FPRM minimization.

22.5 Generalized Reed-Muller Forms.

The Generalized Reed-Muller Forms (GRM) forms are another canonical AND/EXOR forms that are more powerful than FPRMEs. As shown experimentally in [147, 148] and recently by Tsutomu Sasao, the good quality solutions, or exact solutions, for GRM forms are closer to the minimal ESOPs, than any other canonical forms investigated in the literature.

In papers by Csanky, Perkowski et al [147, 494, 148] a theory of such forms has been developed, as well as an efficient heuristic algorithm based on it. These papers suggest also an exhaustive exact method for GRMs. In [147] several useful concepts of general nature have been introduced, that have been next used by Marek-Sadowska [561]. For instance, the prime cubes, that have some similarity to prime implicants in SOP minimization. Interestingly, the concepts introduced by us have been recently found useful by other researchers for other applications. For instance, T. Sasao uses them to create an exact algorithm for GRM minimization [579], and Malgorzata Marek-Sadowska uses them for classification of Boolean functions and matching [561].

Although Sasao's algorithm, being an exact one, creates not worse and often better solutions than the algorithm from [494], the older algorithm remains still useful, since it is faster and can solve larger functions in an approximate way.

These forms can be used both as a representation and also in some special types of decompositions and investigations of symmetry and other properties useful in decomposition.

22.6 Canonical AND/EXOR Multiple-Valued Input, Binary Output Forms.

The ideas of minimal Fixed Polarity RM forms were expanded to Multi-Valued Input Kronecker Reed-Muller (MIKRM) forms. While the FPRM forms are based on two expansions: Positive Davio for variables of positive polarity, and Negative Davio for variables of negative polarity, the binary Kronecker Reed-Muller (KRM) forms use three expansions: Positive Davio, Negative Davio, and Shannon. Therefore, there is 3^n KRM forms for a function of n variables. Each binary variable in a KRM form can have one of three polarities; to each polarity there corresponds a non-singular (orthogonal) binary array. The concept of polarity can be further extended for multiple-valued logic. For a multiple-valued variable there are as many polarities as corresponding orthogonal matrices. Each polarity corresponds to an expansion and an universal cell. These concepts lead to the canonical Decision Diagram and the Flat Multi-valued KRM representations of multiple-valued input functions. These diagrams and representations can find similar applications to the previously mentioned AND/EXOR forms.

22.7 Approximate Minimization of ESOPs.

Approximate Minimization of ESOPs is a very important problem to which much research has been devoted over the last 30 years. It has such a relation to the AND/EXOR PLAs as the classical two-level SOP minimization problem has to the AND/OR PLAs. Similarly to SOPs, creating a quasi-minimal ESOP can become the first stage of a multi-level minimization, and can be also used as a general-purpose representation of Boolean functions.

The first efficient program of this type was called Exorcism [284]. Exorcism was based on application of iterative-improvement of two, introduced by us, cube calculus operations on EXOR-Array cubes, called *primary crosslink* and *secondary crosslink.* This approach has been generalized to handle don't cares, and to multiple-valued input functions, making it thus the first multiple-valued ESOP minimizer [487]. In 1990 Tsutomu Sasao created a similar program, called EXMIN, which was at that time in all respects superior to Exorcism, except for the handling of don't cares [581]. In efforts to further improve Exorcism, more cube reshaping and cube combining operations have been added to it, and the distance of the cubes to which the operations can be applied has been limited. New measure on cubes, called cube difference, which was used to evaluate prospective application of cube operations, has been also introduced. Further experimentation with large examples led to the invention of a new cube calculus operation, called *exorlink.* This operation allowed for very efficient realization in software, and it also includes all the well-known operations of both Exorcism and EXMIN2 (in the meantime Tsutomu Sasao improved EXMIN and renamed it EXMIN2), plus many other new cube operations. Based on the notions of difference and distance of cubes, the authors were able to execute only some particular cases of exorlink operation to selected subsets of cubes, decreasing thus substantially the search space. Finally, since 1993 Exorcism is able to obtain better results than EXMIN2. Moreover, the new version, Exorcism-mv-2 can assume any number of values for each input variable, and these numbers can be different for any particular variable. The program has also no inherent constraints on the size of the function. In addition, it improves much the handling of the incompletely specified functions, which property is missing in EXMIN2. Exorcism-mv-2 was compared with the incompletely-specified function ESOP minimizer created by J. Saul, and very good results were obtained on benchmarks of incompletely specified functions taken from industrial circuits (MCNC, ISCAS). Those circuits had from about 10% to 80% of don't cares. However, working recently in Wright Labs of Wright-Patterson Air Force Base on applying Exorcism-mv-2 to Machine Learning applications, the author found that its performance on 'learning' benchmarks, although better than some of the popularly used 'learning' programs such as C4.5, is still unsatisfactory on larger examples. Analyzing these cases, the author came to the conclusion, that the inferior behavior of Exorcism-mv-2 is caused by the extremely high percent of don't cares that is typical for Machine Learning benchmarks, and can be as high as 99.99999%. We have done some small improvements to Exorcism-mv-2 which improved its operation on many examples, but comparing its operation to a Boolean Ashenhurst/Curtis Decomposer we feel, that much further improvements are possible. This opens up a new general research area, namely designing Boolean and multiple-valued minimization algorithms for *strongly unspecified functions* typical for Machine Learning, data-base, image recognition and AI applications.

In conclusion, Exorcism-mv-2 is currently the *best heuristic ESOP minimizer available*, especially for the case of multi-output strongly unspecified Boolean and multiple-valued input functions. In addition, the author sees clearly now, how both its speed and the quality of the results can be further improved. Exorcism can be used both to create ESOPs as a representation for decomposition, and to minimize the non-decomposable blocks in AC decomposition. This is one of the current research areas.

22.8 Orthogonal Transforms, Fundamental Theorem and Its Applications.

Paper [496] presents a theorem that forms a foundation to all well-known and all possible new canonical circuits with EXOR output gates. Let M be a $2^n \times 2^n$ binary matrix with columns corresponding to minterms and rows corresponding to a family of Boolean functions of n variables. $M[i,j] = 1$ means that the Boolean function of row i includes the minterm corresponding to column j. If the rows are linearly independent with respect to a bit-by-bit EXOR operation, then the family is called *'orthogonal family of Boolean functions'*. The functions from the family are called *'orthogonal'* functions. The theorem states that for any orthogonal family of 2^n Boolean functions f_i of n variables represented as a $2^n \times 2^n$ matrix M, there exists a canonical three-level realization:

$$F = f_0 \cdot S_0 \oplus ... \oplus f_{2^n-1} \cdot S_{2^n-1}$$

where functions f_i are the given orthogonal functions, and coefficients S_i are determined by multiplying matrix M^{-1} by the vector of minterms FV of function F.

Each such canonical expansion creates also an universal cell that can be used in multi-level trees. Analogously as in case of BDDs and KDDs, such trees can be factorized to Directed Acyclic Graphs (DAGs), so that the concept of generalized functional decision diagrams, called Orthogonal Decision Diagrams can be created. Such diagrams include all the known canonical decision diagrams from the literature as their special cases [497].

The generalization of these concepts for multi-output incompletely specified functions F has been also created. These concepts can be used to create efficient decision diagrams and programs to minimize non-decomposable blocks in AC decompositions.

22.9 Logic with Multiple-Valued Inputs and Multiple-Valued Outputs.

An efficient algorithm for ternary Reed-Muller expansions under fixed polarities is presented in [214]. It requires less computation than all the previous algorithms, including those by Chen and Wu, Yang, Green and Harking. The algorithm, which is based on a recursive matrix, can be easily extended to any higher radix, demonstrating that perhaps many other of the described here methods, created by the author, which were developed for binary case or multiple-valued input, binary output, can also be extended to multiple-valued-input, multiple-valued-output logic ('truly multiple-valued'). The algorithm can be applied to both truly multiple-valued circuit logic hardware implementation, or to multi-level realization of binary encoding of this logic, leading thus to circuits with relatively high EXOR component, plus AND and OR gates. All such decision diagrams and canonical forms may be used as representations in decomposition of mv functions.

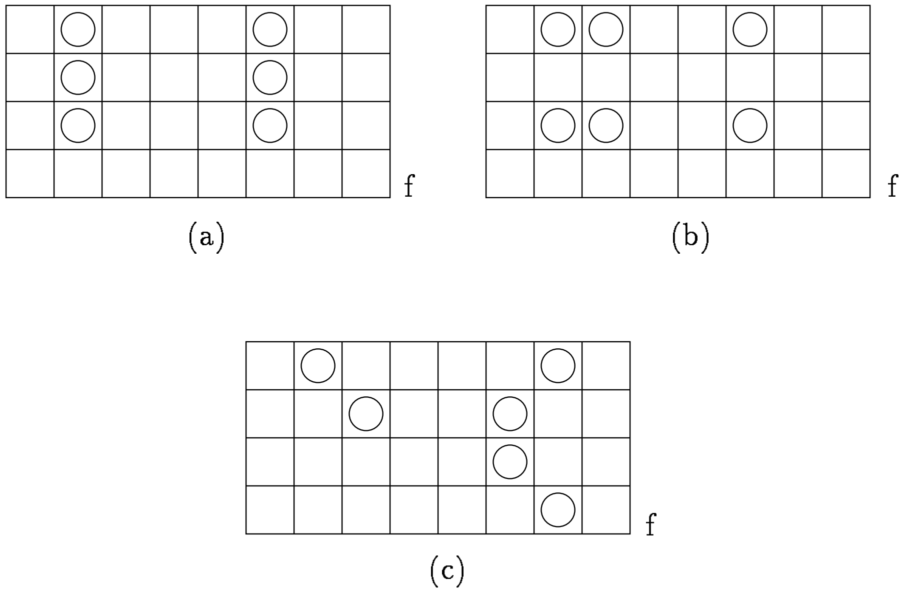


Figure 57: Cube arrangement

23 The Variable Partitioning Problem.

In this section we will present the Variable Partitioning approach applied in TRADE. Good heuristics for variable selection (partitioning) is also in [215].

In this section a method operating on cubes will be presented. It is called the *Pair Weighting Method*. This heuristic method will produce no more than four 'good' partitions which are to be used for decomposition. The basic idea of this method is to arrange cubes (minterms) in the Karnaugh map in such a way that they become more concentrated in either certain columns or rows, like the arrangements in Figure 57a and (b), but not like that in Figure 57c. For a given function, the number of minterms is fixed, the number of cubes after minimization is fixed as well. There are six minterms in Figure 57a. Under that partition, the minterms are concentrated in two columns. In Figure 57b it is the same function. Under that partition, the minterms are concentrated in two rows. In Figure 57c it is the same function again, but the minterms are diverged across the Karnaugh map. Clearly, the column multiplicities under the partitions shown in Figure 57a and (b) are less than that in Figure 57c.

How can we put more cubes in certain columns or rows? We first analyze the relative positions between two cubes in the Karnaugh map. There are four possible relative positions between two cubes as shown in Figure 58. The rectangles stand for cubes. The shaded areas from both cubes are the possible parts that can be put into the same columns or rows in the Karnaugh map. The question then arises, under what partition do the shaded areas from both cubes reside in the same columns or rows? The next example is used to show how to find that partition.

The ON set of function f in Figure 59 is:

```
        abcd
cube1   0-0-
cube2   1110
```

The remaining cubes belong to the OFF set.

The column multiplicity is three under this partition ($ab \mid cd$, bound set cd, free set ab). The Intersection of the two cubes results in the '*Intersection Cube*' cube0.

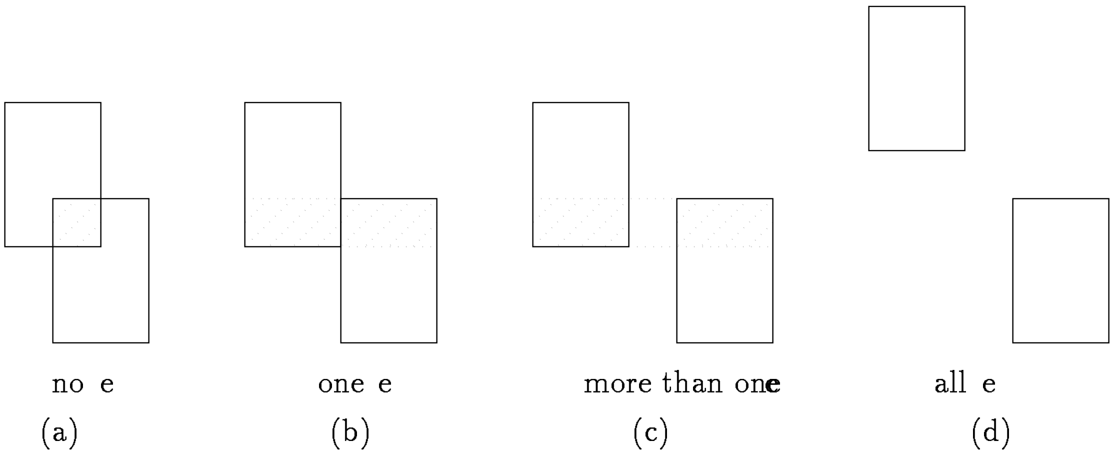


Figure 58: Relative positions between two cubes

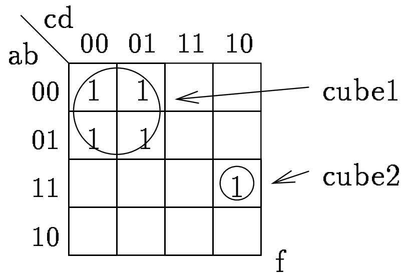


Figure 59: Karnaugh map of function f

```
          abcd
cube1     0-0-
cube2     1110
---------------
cube0     e1e0
```

Because there are two ϵ's, these two cubes have the relative position as shown in Figure 58c. The comparisons of cube0 with cube1 and the cube0 with cube2 result in the Partition Cubes C1 and C2, respectively:

```
          abcd                          abcd
cube0     e1e0               cube0      e1e0
cube1     0-0-               cube2      1110
--------------               --------------
C1        NNNN               C2         NYNY
```

The *Partition Cube* is formed as follows: If the corresponding bits of the two cubes are the same, there is a Y in the Partition Cube. Otherwise N. Next, according to the Partition Cubes one attempts to form partitions that would put as many shaded areas from both cubes into the same columns or rows as possible. The Partition Cubes are ignored if they have the value of all Y's or all N's. Therefore, C1 is ignored. Group variables corresponding to Y in C2 into a group, group b, d, forming a partition $ac \mid bd$. Under this partition, part of cube1 and part of cube2 reside in column 10 ($b = 1$, $d = 0$) of the Karnaugh map.

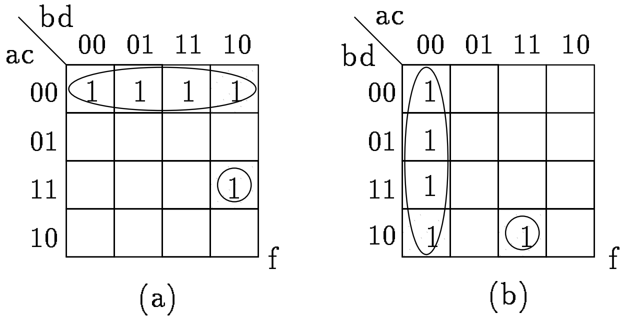


Figure 60: Karnaugh maps for the 'best' partition

ab \ cde	000	001	011	010	110	111	101	100
00	0	1	0	0	0	0	1	0
01	0	0	1	0	0	1	0	0
11	0	0	0	0	0	1	0	0
10	0	0	0	0	0	0	1	0

f

Figure 61: Variable partitioning example

These are minterms 0010 and 1110 as shaded in Figure 60a. This is just what we wanted: both cubes have a part in one column. We form another partition $bd \mid ac$ by grouping variables corresponding to N in C2 (a, c) in one group. Under this partition, part of cube1 and part of cube2 reside in row 10 $(b = 1, d = 0)$ of the Karnaugh map. They are minterms 1000 and 1011 (shaded as well) in Figure 60b. Again, this is what we aimed at: both cubes have a part in one row.

Clearly, the bound set $\{b, d\}$ is a better partition which produces a column multiplicity of two, while bound set a, c results in a column multiplicity of three. In summary, first all Partition Cubes are calculated for each pair of variables. Each Partition Cube forms a partition. Number of appearances of each pair is counted, and according to this number the final partitions are formed.

Now a more complex example will be used to explain the detailed procedure of variable partitioning.

Figure 61 shows the Karnaugh map of function f with the column multiplicity $\mu(ab|cde)$ of five. Its Espresso format input file is as follows:

```
.i 5
.o 1
.ilb a b c d e
.ob f
.type fr
00-01 1
-0101 1
01-11 1
-1111 1
----0 0
1-0-- 0
-0-1- 0
-1-0- 0
```

```
.end
```

'Variable partitioning example'

(1) Calculate Intersection Cubes. For ON set, they are:

```
00-01    00-01    00-01    -0101    -0101    01-11
-0101    01-11    -1111    01-11    -1111    -1111
-----    -----    -----    -----    -----    -----
00101    0e-e1    0e1e1    0e1e1    -e1e1    01111
```

For OFF set, they are:

```
----0    ----0    ----0    1-0--    1-0--    -0-1-
1-0--    -0-1-    -1-0-    -0-1-    -1-0-    -1-0-
-----    -----    -----    -----    -----    -----
1-0-0    -0-10    -1-00    1001-    1100-    -e-e-
```

(2) Calculate Partition Cubes. For ON set, they are:

```
00101    00101    0e-e1    0e-e1    0e1e1    0e1e1
00-01    -0101    00-01    01-11    00-01    -1111
-----    -----    -----    -----    -----    -----
YYNYY    NYYYY    YNYNY    YNYNY    YNNNY    NNYNY

0e1e1    0e1e1    -e1e1    -e1e1    01111    01111
-0101    01-11    -0101    -1111    01-11    -1111
-----    -----    -----    -----    -----    -----
NNYNY    YNNNY    YNYNY    YNYNY    YYNYY    NYYYY
```

For OFF set, they are:

```
1-0-0    1-0-0    -0-10    -0-10    -1-00    -1-00
----0    1-0--    ----0    -0-1-    ----0    -1-0-
-----    -----    -----    -----    -----    -----
NYNYY    YYYYN    YNYNY    YYYYN    YNYNY    YYYYN

1001-    1001-    1100-    1100-    -e-e-    -e-e-
1-0--    -0-1-    1-0--    -1-0-    -0-1-    -1-0-
-----    -----    -----    -----    -----    -----
YNYNY    NYNYY    YNYNY    NYNYY    YNYNY    YNYNY
```

(3) Calculate Relationship Factors.

1. Create four triangle tables as shown in Figure 62, call them ON-Y Table, ON-N Table, OFF-Y Table and OFF-N Table.

2. Initiate all their cells to zero. The value in each cell represents a weighted *Relationship Factor*. between the variables corresponding to the row and column labels of the table. Later on, we will sort the Relationship Factor Table. In order to keep the correct correspondence between the value and the variable pair it represents, we attach two more storage units to each cell to store the two variables that the Relationship Factor corresponds to. So the Relationship Factor Table is, in fact, a 3-tuple list. For example, the ON-Y Table in Figure 62 is a list: $\{(a,b,2),(a,c,4),(a,d,2),(a,e,8),(b,c,2),(b,d,4),(b,e,4),(c,d,2),(c,e,8),(d,e,4)\}$.

ON-Y Table

	b	c	d	e
a	a-b 2	a-c 4	a-d 2	a-e 8
b		b-c 2	b-d 4	b-e 4
c			c-d 2	c-e 8
d				d-e 4

ON-N Table

	b	c	d	e
a	a-b 2	a-c 0	a-d 2	a-e 0
b		b-c 2	b-d 8	b-e 0
c			c-d 2	c-e 0
d				d-e 0

OFF-Y Table

	b	c	d	e
a	a-b 3	a-c 9	a-d 3	a-e 6
b		b-c 3	b-d 6	b-e 3
c			c-d 3	c-e 6
d				d-e 3

OFF-N Table

	b	c	d	e
a	a-b 0	a-c 4	a-d 0	a-e 0
b		b-c 0	b-d 5	b-e 0
c			c-d 0	c-e 0
d				d-e 0

Figure 62: ON-Y, ON-N, OFF-Y and OFF-N tables

3. The Partition Cubes obtained in the second step are weighted in the following way:

   - Group variables corresponding to Y's in the Partition Cube.
   - Select a pair of variables in the group and increase the value of the corresponding cell of the ON-Y Tables by 1.
   - Group variables corresponding to N's in the Partition Cube,
   - Select a pair of variables in the group and increase the value of the corresponding cell of the ON-N Tables by 1.
   - Execute the calculations for all pairs in the group.
   - Perform the same operations for the OFF-Y and OFF-N Tables.

The formula of Relationship Factor is:

$$RelationshipFactor \;=\; sum\ ((pair\ between\ variables\ i\ and\ j)\ ?\ 1\ :\ 0)$$

The summation is over all Partition Cubes and all pairs of variables in the Partition Cubes. '(pair between variables i and j)? 1 : 0' in the equation means that if variables i and j are a pair, take the value 1. Otherwise 0.

For example, if a Partition Cube C1 from the ON set is

```
    abcdefg

C1 = YNYNNYY
```

the cells *a-c, a-f, a-g, c-f, c-g* and *f-g* of the ON-Y Table and the cells *b-d, b-e* and *d-e* of the ON-N Table will be increased by 1.

According to above rules, we fill the ON-Y, ON-N, OFF-Y and OFF-N Tables as shown in Figure 62.

'ON-Y, ON-N, OFF-Y and OFF-N Tables'

(4) Form the 'best' partitions.

The Relationship Factor Table is sorted in decreasing order by the value of the Relationship Factors. The variable pairs with largest values in the ON-Y Table are collected until the required number of variables for the bound set is reached. Both cells *a-e* and *c-e* have the values of 8, so these two pairs are selected and the bound set $\{a, c, e\}$ is formed. For the ON-N Table, cell *b-d* has the largest value of 8. All the remaining are with the same value of 2. Cell *a-b* has the value of 2 and it shares the common variable b with set $\{b, d\}$. Set $\{a, b\}$ is then added to set $\{d\}$ creating a bound set $\{a, b, d\}$. The same operations are performed for the OFF-Y and OFF-N Tables to obtain another two bound sets $\{a, c, e\}$ and $\{b, d, e\}$.

The Karnaugh maps under these four partitions are shown in Figure 63. In fact, we have only three Karnaugh maps because there are only three different partitions out of these four partitions. Partition $\{bd \mid ace\}$ and $\{ac \mid bde\}$ result in a column multiplicity of two. While partition $\{ce \mid abd\}$ results in a column multiplicity of three.

We have checked that only partitions $\{bd \mid ace\}$ and $\{ac \mid bde\}$ can produce the minimum column multiplicity of two. There are ten $(\binom{5}{3} = 10)$ possible partitions out of these five-input function.

The pseudo-code for variable partitioning is shown in Figure 64.

bd \ ace	000	001	011	010	110	111	101	100
00		1	1			1		
01								
11		1	1			1		
10								

f

ac \ bde	000	001	011	010	110	111	101	100
00		1				1		
01		1				1		
11		1				1		
10								

f

ce \ abd	000	001	011	010	110	111	101	100
00								
01	1		1					
11	1		1			1		1
10								

f

Figure 63: Partitions after variable partitioning

```
variable_partitioning()
{
repeat for ON-Y, ON-N, OFF-Y and OFF-N tables
  {
  create Relationship Factor Table;
  sort Relationship Factor Table in decreasing order;
  bound_set = first pair in the queue;

  while (|bound_set| < maximum_bound_set_number)
    bound_set = bound_set U pair in the next position;
  }
}
```

Figure 64: Pseudo-code of variable partitioning

24 The Column Minimization Problem.

The problem to minimize the number of columns is discussed by all authors. Some of them discuss it as the set covering of a compatibility graph with nodes corresponding to sub functions of cofactors, some other authors represent it as graph coloring of an incompatibility graph that is a complement of the compatibility graph. These formulations are mathematically equivalent.

It seems, however, that there is something fundamentally wrong in both above formulations from the point of view of decomposing strongly unspecified functions for ML applications. We will discuss it in the next subsection.

24.1 Troubles with the Column Compatibility Problem.

This problem is not known from the literature and was observed in Wright Labs. Let us assume that we have 100 input variables and 99.99999999% don't cares. This means, most columns are columns of don't cares.

Such problem was observed for the parity function with 96 variables. There is no partitioning problem here and all the troubles to find a decomposable minimum solution of negated EXORs are due to bad column grouping. The question is: 'how to group columns to avoid these troubles?' The following ideas should be considered:

1. Do not combine columns of only don't cares with other columns.
2. Solve the column minimization problem together with the column encoding problem. For instance, in essence, we do not need to absolutely minimize the column multiplicity, only to get it below $k = 2^r$ (r = bound set cardinality - 1), so if maximum number of $r = 3$, 8 is as good as 5.
3. another approach - weighted covering, combine only columns that have 0's corresponding to 0's and 1's corresponding to 1's, and not 0's corresponding to -'s and 1's corresponding to -'s.
4. Use greedy algorithm that selects the largest clique, remove it, and iterate. Similarly to the 'disjoint star' covering algorithm of Michalski.
5. Combine step-by-step the most similar columns. Do not treat '-' as similar. Use the weights for:

   1) number of agreements.

   2) probability.

   3) probability that identical values are matched.

This is an area of our active current research. The goal is a reformulation of the fundamental Column Minimization Problem.

24.2 Graph Coloring.

The graph coloring approach was implemented in TRADE program. The problem of finding the smallest column multiplicity was reduced to the one of performing proper graph coloring with the minimum number of colors. *Graph coloring*, [139, 259] is one in which every two nodes linked by an edge are assigned different colors. *Minimum graph coloring* is one with the minimum number of colors.

Graph coloring is an NP-hard problem. There has been a substantial research in graph coloring in order to find the algorithms for a quasi-optimum solution with the fastest possible speed. The method presented here is a fast graph coloring method. This method has been programmed and tested on many examples, it resulted in excellent colorings. The method found exact colorings for graphs in [139] and even found better coloring than that that was claimed to be minimal in the book. The method is called the *Color Influence Method*.

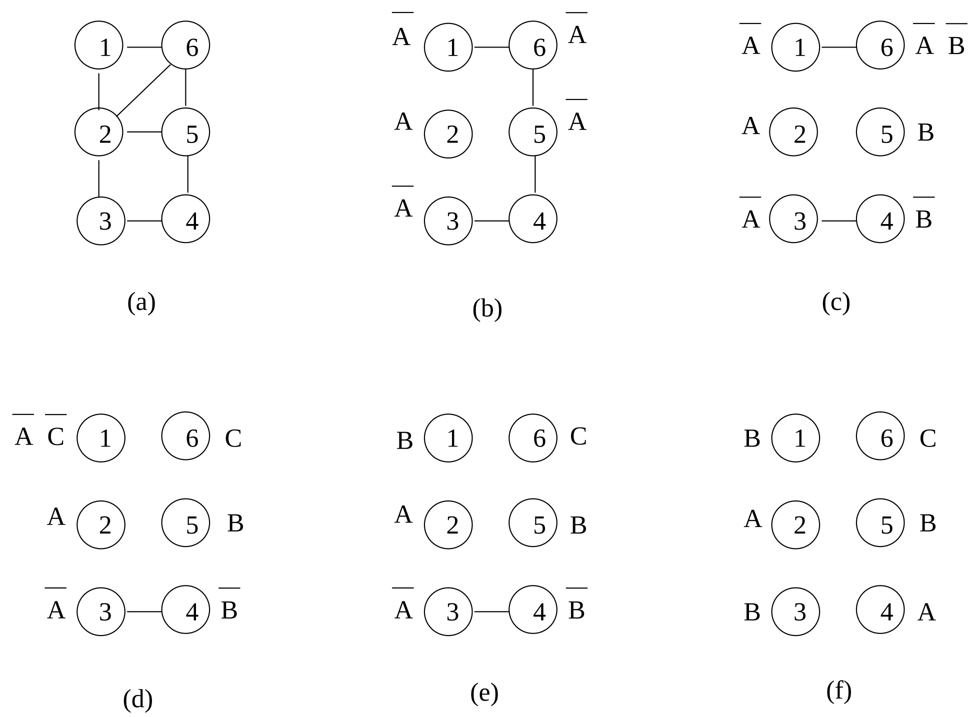


Figure 65: Graph coloring using Color Influence Method

The main idea of this method is to evaluate the influence of the color assignment to a node over the entire graph, and chose the color which results in a minimum influence. The *minimum influence* means that the color assignment to a node will produce a minimum increase of color-in-bar's.

The *color-in-bar's* (restrictions) are the colors that the node cannot be assigned with, which are denoted by $\overline{A}, \overline{B}, \ldots, \overline{A}\ \overline{B}, \overline{A}\ \overline{C}, \ldots$ as in Figure 65.

After each color assignment to a node, the complexity of the graph is decreased. This is a greedy method with global evaluation. The next example is used to illustrate this method.

Figure 65a shows a graph that need to be colored. Start from the node with the greater number of edges, that is node 2, assign color A to it. This color assignment results in that nodes 1, 3, 5 and 6 cannot be assigned with color A, denote this restriction on those nodes by color-in-bar $\overline{A}$'s, and remove all corresponding edges as shown in Figure 65b. Then, color the node with the most number of color-in-bar's. If there are more than one node with the same number of color-in-bar's, chose the node with the most number of edges. If there are still more than one node, evaluate the influence of each color assignment, and assign the node with a color which results in a minimum influence. If a node can be assigned with more than one color, the evaluation of the influence of each color assignment is also required. According to the rules stated above, nodes 5 and 6 are selected because they have the same number of color-in-bar's and the same number of edges. Assigning color B to node 6 will result in a restriction $\overline{A}\ \overline{B}$ on node 1 and a restriction $\overline{A}\ \overline{B}$ on node 5. While assigning color B to node 5 will result in a restriction $\overline{A}\ \overline{B}$ on node 6 and a restriction $\overline{B}$ on node 4. Because one $\overline{A}\ \overline{B}$ restriction and one $\overline{B}$ restriction result in less influence than two $\overline{A}\ \overline{B}$ restrictions, assigning a color to node 5 produces less influence than assigning a color to node 6. Node 5 is selected, color B is assigned to it, as shown in Figure 65c. The same way, color C is assigned to node 6, as shown in Figure 65d, and color B to node 1, as shown in Figure 65e. Nodes 3 and 4 are in the same condition, and have the same influence on the graph. If color B were assigned to node 3, color A or color C must be assigned to node 4. The final color assignment is shown in Figure 65f.

```
graph_coloring()
{
sort nodes in decreasing order by the number of edges;

color first node;
remove it;
mark restrictions;
remove edges;

while ((cib_set = largest_number_color_in_bar(node_queue)) != empty)
  {
  e_set = largest_number_edge(cib_set);

  if (|e_set| > 1)
    f_n = minimum_influence(e_set);
   else
    f_n = e_set;

    color f_n;
    remove f_n;
    mark restrictions;
    remove edges;
  }
}
```

Figure 66: Pseudo-code of graph coloring

The above algorithm was incorporated into a program, named *COLOR*, and was run on a networked SUN 4/670MP workstation. The program was tested on graphs with different number of nodes (N = 100 $\rightarrow$ 1000) and different edge percentages (P = 10% $\rightarrow$ 90%). The maximum number of edges in a graph is $\frac{N(N-1)}{2}$, N is the number of nodes in the graph. The edge percentage (P) is simply the percentage of the maximum number of edges. Edges in the graph are randomly generated.

By statistic analysis, it is found that the time (T) is proportional to the number of nodes (N) in a polynomial form $T \sim N^{2.5}$.

The graph coloring problem is then solved (approximately) in a polynomial time, and not in an exponential time. For small graphs, we are able to verify that the algorithm gives the minimum solutions. It is then hoped that it gives reasonably good results for larger graphs as well, but this claim cannot be verified, since the exact minimal optimizer is not yet available (we are working on it). The pseudo-code for graph coloring is shown in Figure 66.

25 Special Decompositions.

Because the general decomposition is difficult, one has to look to different ways of dealing with the problem. One of them may be to consider several particular cases of decomposition in more detail, and have a program that would evaluate and combine several efficient "mini-decomposers".

Applications of functional decomposition to test generation are described in [404].

Decomposition for minimization of Power Consumption is in the paper by Schneider, 1994, [597].

Paper [640] introduces new kind of decomposition that is useful for NAND and NOR gates. The paper is very similar to Perkowski's [485] paper which introduced the concept of dissected pairs for this task.

The following special structures of circuits for decomposers have been analyzed:

1. Maitra cascades,
2. Iterative circuits, pipelined logic,
3. Trees with repeated leaf variables,
4. Trees without repetitions of leaf variables,
5. Tandem networks of Butler.

26 Detection of Symmetry.

Detection of symmetry helps to simplify the variable partitioning problem. This was used in binary decomposition programs, but not in multiple-valued decomposers.

26.1 Approach of Stankovic and Moraga to Detection of Symmetry.

Stankovic and Moraga in [629] presented an approach to symmetry detection of mv functions.

In general, the problem of realization of a multiple-valued function is a relatively difficult task. Let us however first observe, that the examples of mv functions are naturally simpler than binary functions for the same tasks. For instance, assume that there is 5 different values of variable and we have 6 variables. In binary case we need 3 bits for a variable, times $6 = 18$ binary variables. In the mv case we need just six 5-valued variables.

In some cases, when the given mv function has some particular properties, for example, symmetry or partial symmetry with respect to a given pair of variables, this problem is greatly simplified.

Further, let us observe that if we have symmetric variables and we use them to create mv variables, the symmetry is preserved. If we have symmetric mv variables, it is not necessarily reflected in respective binary variables that encode them, so symmetry of mv variables is a more powerful property, and will perhaps contribute more to the synthesis of strongly unspecified functions.

Decomposition of an mv function in recognition of its symmetries will lead to significant simplifications of the synthesis procedure.

In practice, one of the advantages of knowing that a given function is symmetric or partially symmetric is the potential economy offered by the realization of such a function using a smaller number of components. A similar statement applies to the functional decomposition, in which case the synthesis of an n-variable function reduces to the synthesis of two or more auxiliary functions with fewer variables. In both cases, the time for performing the synthesis procedure is reduced as well.

The problem of symmetry detection and functional decomposition on a given pair of variables may be studied both in the original domain and in the spectral domain. The existing results derived in the original domain are primarily based on some algebraic transformations using tabular representation of mv functions representing the extensions of the corresponding techniques by Ashenhurst/Curtis or by the graph coloring techniques of Muzio [461].

In the spectral domain, methods for symmetry detection and functional decomposition are based on relationships between spectral coefficients [301], [434], [302], [299], [300], [191], [684, 685], [442], [626], [627], [628], and [664],

In functional domain, the problem of using symmetry in decomposition was discussed in [725, 175, 332, 629, 628, 626].

The efficiency of spectral methods is mainly supported by Fast Fourier transforms on groups (the Walsh transform for Boolean and the Chrestenson transform for mv functions). For an n-variable mv function there are $(n/2)$ different pairs of variables. Therefore, to detect all possible symmetries of this function (or to examine the possibility of the decomposition on all pairs of variables), we need to perform a great number of checks. In the practical applications this represents a tedious task related to high requirements on processor time and memory. This fact considerably reduces advantages of this method.

26.2 Use of Symmetry in Decomposition.

It is well known that the number of possible partitions for a function with n variables is 2^n, which makes it impractical to search all partitions for larger functions. However, if variables x_1 and x_2 are partially symmetric, then if a bound set $D \cup \{x_1\}$ has been considered, the bound set $D \cup \{x_2\}$ does not have to be considered. The more partial symmetries exist, the less bound sets need to be considered. These ideas lead to the improved decomposition algorithms for functions with partial and total symmetries.

Recently, Malgorzata Marek-Sadowska created a new efficient method of classifying Boolean functions [561]. This method can be used to design much more powerful decomposers, since it takes into account more relations on the variables, such as:

- nonequivalence symmetry,
- equivalence symmetry,
- mixed symmetries,
- skew-nonequivalence symmetry,
- other

In a similar but earlier study, Radomir Stankovic and Nebojsa Denic [628], introduced the same 8 symmetry classes as Sadowska, but used the apparatus of Gibbs derivatives instead of FPRMs.

Nobody has done the study of using symmetry of multiple valued functions in decomposition, but there are interesting papers by Butler and Schueller [119], Epstein [197], Epstein, Miller and Muzio [198], [199], Stojmenovic, Miyakawa and Tosic [641], Mukhophadyay [447], Muzio [462], Muzio, Miller, and Epstein [460].

Muzio [462] has studied the realization of general multiple-valued symmetric functions from fundamental symmetric functions. Specifically, any arbitrary symmetric function can be realized as the disjunction of fundamental symmetric functions. Some of these functions may be combined, improving the efficiency of the algorithm. The objective of the design is to realize a given function with as few combinations as possible. The problem discussed by Muzio was the worst case. That is, certain functions will require the maximum number of combinations, and it is desired to find this maximum number. Muzio computed the maximum for 3- and 4- valued systems and conjectured the values for general m-valued systems.

Butler [119] solved the general problem and showed that as in the 3- and 4- valued cases, the ratio of the maximum number of components to the total number of fundamental symmetric functions approaches one-half as the number of variables n increases. This has an analog in binary sum-of-products expressions. That is, the binary function with the greater number of product terms (the EXOR function) requires exactly one half of the total number of minterms.

If we apply the tree search paradigm in the space of variables for partitioning [506], then it would be relatively easy to apply standard operator equivalence techniques to remove descriptors from nodes. For instance, if variables a and b are symmetric, one of them is removed as a descriptor. Let us observe that this technique can be used dynamically, to the newly created subfunctions, and the goal of previous decompositions would be to increase the symmetries in the subsequent stages. This way a subset of all bound sets would be tested to obtain the exact decomposition (the decompositions that minimize the value of the multiplicity index μ).

26.3 Kim's and Dietmeyer Approach to Symmetric Decomposition.

Kim and Dietmeyer [332], [331] observed that current systems for multilevel synthesis from PLA-like macros have a difficulty with symmetric functions. Designs of totally symmetric functions generated from descriptions that lack global network structure, have on average more than twice as many literals as the best designs, while the designs of non symmetric functions have on average 20% more literals than the best designs.

Classical disjoint decomposition, which is generally considered for small networks, is particularly suitable for symmetric function synthesis.

The Boolean PLA decomposers of Devadas [171, 174], and Ciesielski [721, 722], overcome the two most important drawbacks of algebraic factorization methods: too much dependency on given expressions and the sequential generation of subfunctions. However, they have no special treatment for symmetric functions.

Kim and Dietmeyer developed a simple but effective heuristic method for synthesizing symmetric functions that detects and takes advantages of symmetry and is based upon classical disjoint decomposition theory. The program of Kim and Dietmeyer produced almost always the best designs known to these authors.

Taking advantage of symmetry in multilevel synthesis requires detection of symmetry sets of multiple-output function F. Extensive work on the detection of symmetry sets using the array format was reported by [175].

Detecting of symmetry in the arrays of F may be performed by sequentially testing the arrays of each f_i The detection of symmetry pairs forms the basis for detecting larger symmetry sets. Detecting symmetry pairs is described by the following theorem:

Theorem 26.1 *(Kim and Dietmeyer).*
$f = (ON, DC, OFF)$ is symmetric with respect to set $L = \{x_i, x_j\}$ if and only if $ON_{x_i\ \overline{x_j}} \wedge OFF_{\overline{x_i}\ x_j} = 0$ and $ON_{\overline{x_i}\ x_j} \wedge OFF_{x_i\ \overline{x_j}} = 0$

If OFF-conditions are not provided, tests based on the following corollary are usually preferred because computing the complement often takes more time than other operations.

Corrolary 26.1 *(Kim and Dietmeyer).*
$f = (ON, DC, OFF)$ is symmetric with respect to set $L = \{x_i, x_j\}$ if and only if
$[\ ON_{x_i\ \overline{x_j}} \subseteq ON_{\overline{x_i}\ x_j} \vee DC_{\overline{x_i}\ x_j}\ ]$ and $[\ ON_{\overline{x_i}\ x_j} \subseteq ON_{x_i\ \overline{x_j}} \vee DC_{x_i\ \overline{x_j}}\ ]$

Theorem 26.2 *(Kim and Dietmeyer).*
If f is a completely specified function, symmetry is an equivalence relation on the variables of symmetry. Thus if $\{a, b\}$ and $\{b, c\}$ are symmetry pairs, $\{a, c\}$ is also a symmetry pair, and $\{a, b, c\}$ is a symmetry set. If L is a symmetry set and $\{a, b\}$ is a symmetry pair where a from L and b not from L then $L \cup \{b\}$ is also a symmetry set.

However if f is not completely specified, detection is not easy. Symmetry is not even a compatible relation in general. The three-element symmetry set is a special case among larger symmetry sets, and the following theorem shows how in may be detected.

Theorem 26.3 *(Kim and Dietmeyer).*
$f = (ON, DC, OFF)$ is symmetric with respect to $\{a, b, c\}$ if and only if f is pairwise symmetric with respect to $\{a, b\}$, $\{b, c\}$ and $\{a, c\}$.

Dietmeyer introduced a new operator to find largest symmetry sets in an relatively efficient way.

It is well-known that if f is totally symmetric then the output of f depends only on the weights of the input vectors. This can be carried over to multi-level multi-output functions: if F is symmetric with respect to L, then every single-output function f_i in F depends only on the L-part weights of input vectors. This weight dependency suggests the disjoint decomposition.

$$F(X) = F'(W, X - L) \tag{13}$$

where $X = \{x_1, x_2, \ldots\}$ is the set of input variables and $W = \{w_1, w_2, \ldots\}$ is the set of intermediate variable names assigned to the outputs $w_i(L)$ of $W(L)$, which is a multiple-output function totally symmetric on L. The variables of L are called *decomposition variables* and the function $W(L)$ will be called *recoder*.

Iterative decomposition using their heuristic recoding techniques is efficient for totally symmetric functions. The authors write that expanding this approach for partially symmetric functions is an open problem, but they believe that many of their ideas will also work for partially symmetric functions. They want to select the best recoder type based on the analysis of functions to be decomposed. Let us hope that this approach will also be easily expandable for multiple-valued functions.

27 Maitra Cascades.

There are two types of Maitra cascades. The first one is a redundant cascade where one or more inputs are driven by the same Boolean variable. The second type is a non-redundant cascade in which every input is connected to a distinct Boolean variable.

The properties of Boolean functions which make them Maitra-realizable have been discussed in both spectral and function domains. The spectral approach seems to be an efficient method for testing and realizing any given Boolean function because the spectral coefficients provide global information about the function, which is more useful than the Boolean representation of the function.

The map [406], and the algebraic [370] approaches require the function to be in the minimum sum-of-product Boolean expression, which is a disadvantage.

The matrix approach of Sklansky [619] is better than the map and algebraic techniques. When the spectral approach is compared with the matrix method, it is observed that the procedure structure in spectral domain is more convenient for computer implementation, and it is not restricted by the number of true minterms or the number of input variables. This is then much better than in [619].

Let us observe that in general, the spectral approaches to various types of Boolean decompositions, not only AC one, but also pre- and post-processor decompositions, become now available thanks to the efficient Clarke's algorithm [128] to find a Walsh spectrum. Such algorithms were not useful practically before, since no efficient method to calculate the entire or partial spectra were available. The only disadvantage is that a method such as one by Clarke may be quite complex to implement.

Not much has been published about multiple-valued logic cascades, especially for incompletely specified functions.

27.1 Cascades and their Generalizations.

Let us recall the 96-input EXOR problem. If one would assume that the corresponding circuit is Maitra-cascade realizable, then more efficient decomposition algorithm can be found. For each input variable one of three possible operations, AND, OR and EXOR is assumed, and the function is folded with respect to this operation. If no folding is possible, another operation is tried, or the program backtracks. If the circuit is Maitra-cascade realizable, this algorithm gives the solution, regardless on the number of inputs and percent of the don't cares. Although this algorithm is exponential with the number of input variables, it is still more efficient than the general AC decomposition. This is the reason why we are interested in using some ideas based on Maitra cascades in the framework of the AC decomposition.

We plan:

1. Develop an algorithm to create Maitra Cascades, [406], for strongly unspecified functions. For such functions, because of the high percent of don't cares, there may be solutions for most real life benchmark functions. Prove properties of the algorithm; specifically, we will have to prove that if such a cascade exists, the algorithm is able to find it.

2. Generalize the concept of Maitra cascade with the above in mind.

3. Use the function folding approach of Perkowski to solve the simpler EXOR problem for strongly unspecified function of 96 variables (10,000 samples).

4. This is similar to the problem of removing vacuous variables, removing all vacuous variables may be not good, still removing such variables should remain as an option. Use a method similar to above to reduce the number of variables. If found, the totally vacuous variables should be definitely removed.

27.2 Spectral Approaches to Maitra Cascades.

Spectral methods have been also used for synthesis of cascades in the paper by Asaad and Bennett [26]. Asaad and Bennett define properties of Boolean networks that are realizable in non-redundant cascades. They use Walsh transform and give an algorithm for realizing a cascade based on these properties. Although their method is only for completely specified functions, it can be adapted to incompletely specified functions using techniques from Falkowski et al. [208, 209].

Synthesis of Maitra cascades by means of spectral techniques is also discussed by Stankovic, Tosic and Nikolic, [625].

By using the same approach as Mukhopadhyay and Stone [449], M. Stankovic et al give, in the spectral domain, a procedure that can be used to test an arbitrary function for cascade realizability and to generate a Maitra cascade for a function if one exists. The procedure determines one or more cells at the bottom of a cascade realization of the function and the residual function that drives this portion of the function. The procedure is applied iteratively to the successive residual functions, until a trivial residual function is found, since the applied method is based on the partial decomposition of the function. In the spectral domain, this decomposition is uniquely defined by the values of the determined pairs of spectral coefficients. A spectrum of the residual function is determined by applying the spectral translation to the spectral coefficients of the given function. These results permit to formulate a procedure that is suitable for computer processing. However, no experimental results are given in the paper.

Classification of Boolean functions by spectral means is also discussed in [257].

28 General Decomposition with EXOR Transformations.

The known decomposition methods are passive in the sense that they only test whether a function is decomposable or not. If it is, the decomposition is carried out. But what do we do if the function is not decomposable?

The method presented in TRADE is called a *Local Transformation Method.* This method can transform a function which is non decomposable (in the sense of Curtis) into several decomposable ones.

The basic idea of this method is to transform some columns in the original Karnaugh map to make them identical to some other columns in order to decrease the column multiplicity. It is carried out in three steps:

1. The output values of the Conflict Cubes (the Conflict Cubes are the cubes that make the two columns different) in one column are complemented in order to make the two columns identical, creating the Modified Karnaugh Map.

2. The so-called Modifying Karnaugh Map is created from the initial map and the Modified Map.

3. EXOR operation on the function described by the Modified Karnaugh Map and the function described by the Modifying Karnaugh Map is executed, creating the original function.

A *Modified Cube* is a name for the Conflict Cube which outputs have been complemented, both cubes are identical. An attempt is made in the first step to make the Modified Cubes as large as possible, and at the same time keep the number of the Modified Cubes as small as possible.

Like the cube -1 in columns 000 and 001 in Figure 67a, it makes these two columns different. The formula for calculating the Conflict Cubes between i-th and j-th columns is:

Conflict Cube Set $= (\mathrm{ON(i)} \sqcap \mathrm{OFF(j)}) \sqcup (\mathrm{ON(j)} \sqcap \mathrm{OFF(i)})$

where:
ON(i) and OFF(i) are the ON and OFF sets of i-th column.
ON(j) and OFF(j) are the ON and OFF sets of j-th column.

The formula states that the Conflict Cubes between the i-th and j-th column are the union of the intersection of the ON set of i-th column with the OFF set of j-th column and the intersection of the ON set of j-th column with the OFF set of i-th column.

The following example is used to illustrate this method and above terminologies. Figure 67a is an original Karnaugh map with the bond set $\{c, d, e\}$ and a column multiplicity of four. We intend to decompose this function into two sub functions, and each of them has a column multiplicity of no more than two.

We perform the local transformation to decrease the column multiplicity. First, the output values of cubes -10-1 and -1101 in Figure 67a are complemented. The resulting cubes are called *Modified Cubes* (they are the same as the Conflict Cubes). After modification, the resultant Karnaugh map is called the *Modified Karnaugh Map (Modified Function) f_{ed})* as shown in Figure 67b. Now the column multiplicity of the Modified Karnaugh Map is two.

Next, because the output value of the Modified Cubes have been complemented, the compensation must be made. To achieve this, another Karnaugh map is created with the positions corresponding to the Modified Cubes in the original Karnaugh map set to 1's and the others set to 0's. This is shown in Figure 67c. It is called the *Modifying Karnaugh Map* (*Modifying Function f_{ing}*). Notice that in this case the column multiplicity of the Modifying Karnaugh Map is two as well. If there are DC cubes in the original Karnaugh map, we keep them unchanged in both the Modified and Modifying Karnaugh Maps, because they will make both of these maps as amenable as possible for further minimization.

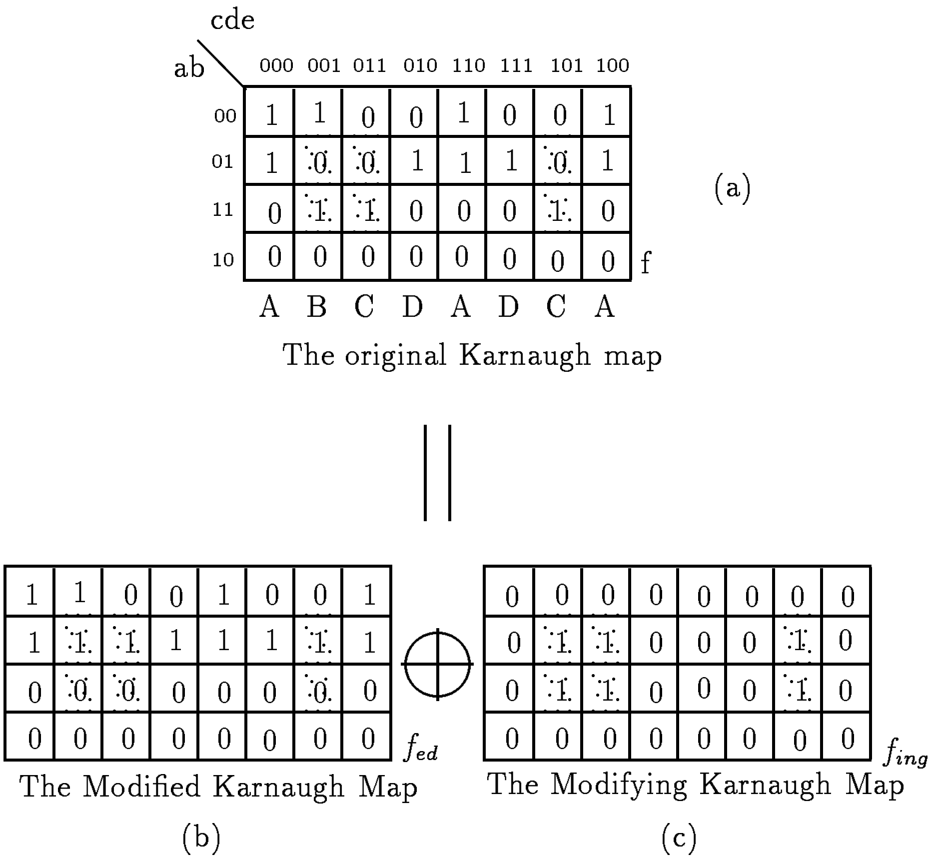


Figure 67: Local transformation

Finally, the EXOR operation of the function described by the Modified Karnaugh Map and the function described by the Modifying Karnaugh Map results in the function described by the original Karnaugh map. That is:

$$f = f_{ed} \oplus f_{ing}$$

The presented method changes a non decomposable function (in sense of the column multiplicity less than or equal to two) into two decomposable functions (with the column multiplicities of both equal to two). The method is called local transformation, but this local transformation is based on the global view of the entire function to make both Modified and Modifying Karnaugh Maps more simple.

The application of local transformation to the general implementation of FPGA mapping is shown in Figure 68. If a function is non decomposable, it will be transformed into two functions, a Modified and a Modifying function. If the Modified or Modifying function is non decomposable (in a more restricted condition), the local transformation should be applied again. Next, the function f from Figure 67a is used again as an example to show the detailed procedure of local transformation.

In Figure 67a there are only four different columns as shown in Figure 69. We denote these four columns by letter A, B, C and D, and call them (letter) column A, column B, column C and column D. Column A (letter column) includes three real columns: columns 000, 110 and 100 in the original Karnaugh map. Column B includes one column, column 001. Column C includes two columns: columns 011 and 101. Column D includes two columns: columns 010 and 111. "Number" in Figure 69 is the number of real columns that a letter column contains.

Create the Modification Factor Table as shown in Figure 70 and initiate all its cells to zero. The value in each cell is a weighted Modification Factor of the column pair corresponding to the row and column labels of the table. Cell $A \rightarrow B$ stores the Modification Factor of complementing the output value of all Conflict Cubes in the column A in order to make columns A and B identical. Cell $B \rightarrow A$ stores the Modification Factor of complementing the output value of all Conflict Cubes in the column B in order to make columns B and A identical.

Later on, the Modification Factor Table is sorted. In order to keep the correct correspondence between the value and the column pair it represents, two more storage units are attached to each cell to store the two columns that the Modification Factor corresponds to. The Modification Factor Table is, therefore, a list of 3-tuples. For example, the Modification Factor Table in Figure 70 is a list: $\{(A, B, 12)$, $(A, C, 24)$, $(A, D, 15)$, $(B, A, 4)$, $(B, C, 5)$, $(B, D, 8)$, $(C, A, 16)$, $(C, B, 10)$, $(C, D, 8)$, $(D, A, 10)$, $(D, B, 16)$, $(D, C, 8)\}$.

The Modification Factors are calculated in the following way:

- If the Conflict Cube is a minterm, the value of the corresponding cell in the table is increased by the number of input variables.
- If the Conflict Cube consists of two minterms, the value of the corresponding cell in the table is increased by the number of input variables minus one. This is because a larger cube can simplify to a greater extent both the Modified and Modifying Karnaugh Maps.
- If the Conflict Cube is composed of four minterms, the value of the corresponding cell in the table is increased by the number of input variables minus two, and so forth. That is:

  $$ModificationFactor = \sum (number\ of\ input\ variables - number\ of\ DCs\ in\ the\ Conflict\ Cube)$$

  The summation is over all Conflict Cubes in the modified column.

Let's fill the Modification Factor Table in Figure 70 now. The Conflict Cube between column A and B is the cube -1. If we change column A to make it identical to column B, we need to complement

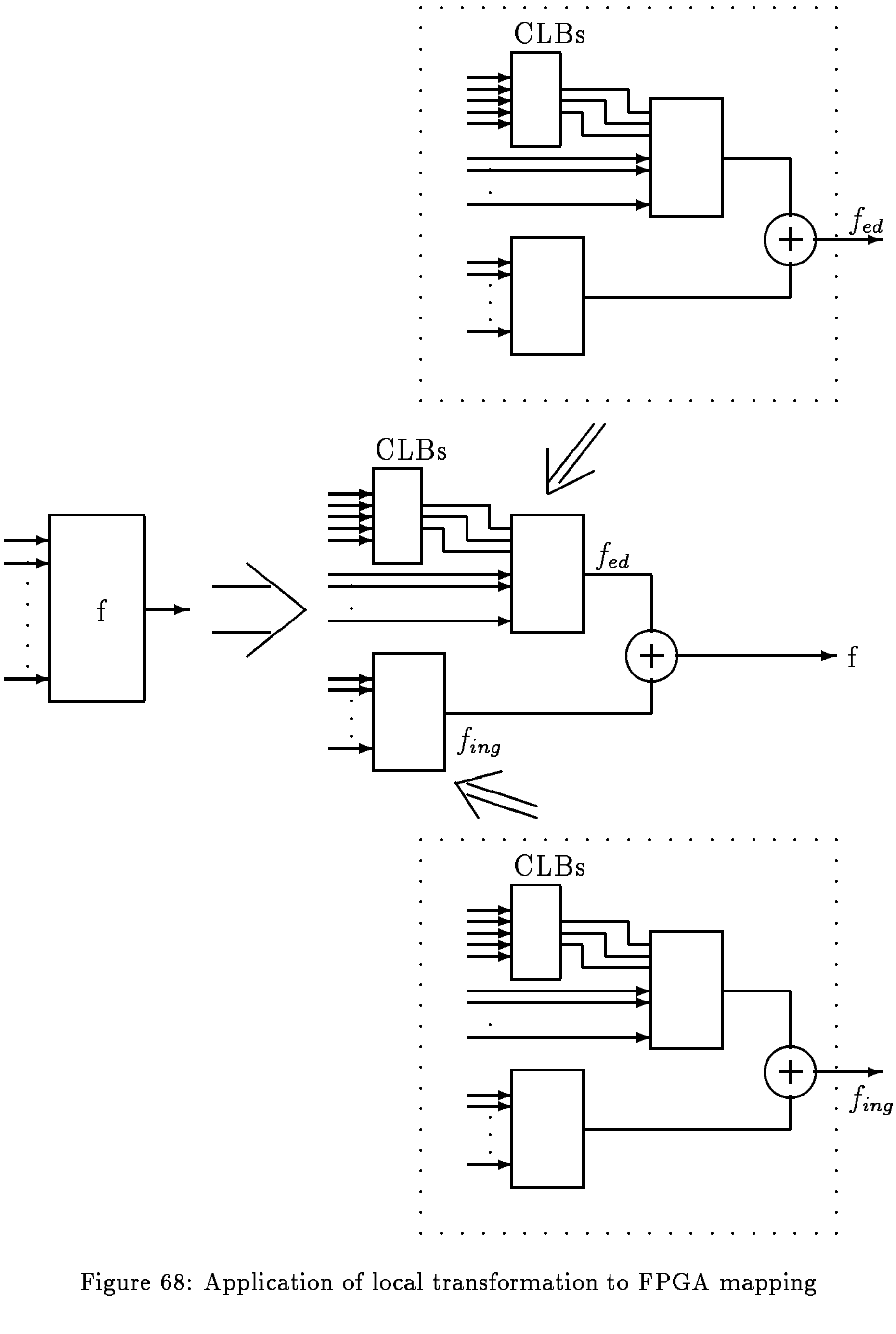


Figure 68: Application of local transformation to FPGA mapping

ab				
00	1	1	0	0
01	1	0	0	1
11	0	1	1	0
10	0	0	0	0
Column	A	B	C	D
Number	3	1	2	2

Figure 69: four different columns

	A	B	C	D
A		A→B 12	A→C 24	A→D 15
B	B→A 4		B→C 5	B→D 8
C	C→A 16	C→B 10		C→D 8
D	D→A 10	D→B 16	D→C 8	

Figure 70: Modification Factor Table

the output value of the Conflict Cube (cube -1 in column A). That is to change the output value of minterm 01 of column A in Figure 69 from 1 to 0 and the output value of minterm 11 from 0 to 1. Column A will have the vector [1, 0, 1, 0] after complementation of the Conflict Cube. This vector is the same as that of the column B. Cube -1 has one - (dash), the Modification Factor would be the number of input variables minus 1, that is four. Furthermore, column A includes three columns, columns 000, 110 and 100. All these three columns need to be modified. Therefore, the Modification Factor must be multiplied by three, and the final Modification Factor is twelve, as shown in cell $A \rightarrow B$ of the Modification Factor Table in Figure 70. Another way to make columns A and B identical is to complement the output value of the Conflict Cube, cube -1 in column B. That is to change the output value of minterm 01 of column B in Figure 69 from 0 to 1 and the output value of minterm 11 from 1 to 0, so both columns have the same vector [1, 1, 0, 0] after the modification. Column B includes only one column which is column 001, the Modification Factor is four as shown in Figure 70. Using the same reasoning the Modification Factor Table is completed.

The Modification Factor Table is sorted in an increased order of the Modification Factors. The columns which have smaller values of the Modification Factors are now modified until the required column multiplicity is reached. Cell B $\rightarrow$ A is selected because it has the smallest value of four. Complement the output value of Conflict Cube, cube -1 in column 001 of the Karnaugh map in Figure 67(a). The vector of column 001 is changed from [1, 0, 1, 0] to [1, 1, 0, 0]. At the same time, set the output value of cube -1001 of the Modifying Karnaugh in Figure 67(c) to 1. After modifying column B, the column multiplicity of the Modified Karnaugh Map is reduced to three. Our aim is to reduce it to two, so another modification will be carried out. The next smaller Modification Factor is five in cell B $\rightarrow$ C. But we cannot change column B to column C because we have changed column B to column A. The next smaller value is eight in cell $C \rightarrow D$. Complement the output value of Conflict Cubes, cube -1 in both columns 011 and 101 of the Karnaugh map in Figure 67(a), The vectors of the columns 011 and 101 are changed from [0, 0, 1, 0] to [0, 1, 0, 0]. At the same time, set the output value of cubes -1011 and -1101 of the Modifying Karnaugh in Figure 67(c) to 1. After this modification, the column multiplicity of the Modified Karnaugh Map is reduced to two.

The function f with a column multiplicity of four has been transformed into two functions: f_{ed} and f_{ing}, both of them have a column multiplicity of two. The relation between them is:

$$f = f_{ed} \oplus f_{ing}$$

The pseudo-code for local transformation is shown in Figure 71.

The above technique has been incorporated into TRADE. The basic steps of *TRADE* are as follows:

1. Read in the input file written in *Espresso* ".type fr" format

2. Select an output. Perform partition analysis (the number of bond set variables is fixed to five) to obtain the "best" partitions and Additional Partitions. *Additional Partitions* are the partitions whose bond sets consist of the variables that are the input variables of some CLBs in the CLB Pool, and these variables are also in the range of the input variables of the current decomposition. The *CLB Pool* is a list of all CLBs previously generated by the program. For example, if the input variables of the current decomposition are *a, b, c, d, e, f, g, h, i,* and in the CLB Pool there are three CLBs with the input variables of each: $\{a, b, c, d, e\}$, $\{c, d, g, h, i\}$ and $\{a, h, k, l, m\}$ respectively. Then the variable sets $\{a, b, c, d, e\}$ and $\{c, d, g, h, i\}$ can be used as the additional partitions, while *a, h, k, l, m* can not be used because there are no variables *k, l* and *m* in the current decomposition.

3. Execute decompositions using the "best" and Additional Partitions. Encode the bond set and try to use as many CLBs from the CLB Pool as possible. The way to efficiently reuse CLBs will be discussed later in this chapter. Graph coloring technique is applied at this stage to get a quasi-optimum don't care assignment. Select a partition which results in the smallest Cover

```
local_transformation()
{
create Modification Factor Table;
sort Modification Factor Table in increasing order;
modify column at the beginning of the queue;
change Modified Karnaugh Map;
fill Modifying Karnaugh Map;

while (column_multiplicity > required_column_multiplicity)
  {
  modify column at the next position
  change Modified Karnaugh Map;
  fill Modifying Karnaugh Map;
  }
}
```

Figure 71: Pseudo-code of local transformation

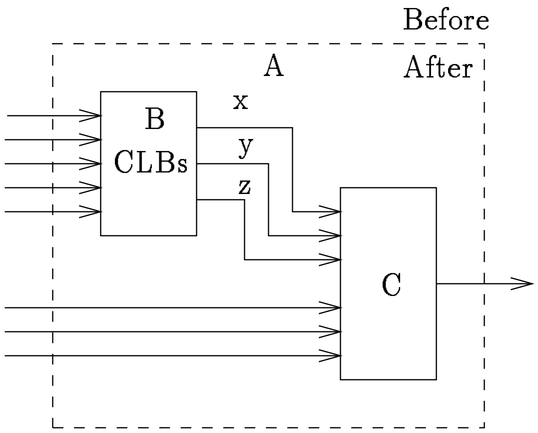


Figure 72: Before and after decomposition

Ratio. The *Cover Ratio* is a ratio of the number of newly created CLBs over the difference of the input variables before and after decomposition, that is:

$$CoverRatio = \frac{number\_of\_newly\_created\ CLBs}{inputs\_before\_decomposition - inputs\_after\_decomposition}$$

For example, in Figure 72, if CLB x and y exist in the CLB Pool, CLB z is a newly created one, then Cover Ratio = 1/(8 - 6) = 0.5.

If the function is non decomposable, perform the local transformation to make it decomposable.

4. Repeat steps 2 to 3 for the blocks left (block C in Figure 72) until all decomposed blocks are with five or less inputs.

5. Repeat steps 2 to 4 for all Modifying Functions which were created by local transformations.

6. Repeat steps 2 to 5 for all outputs.

Recently, there is work on incorporating many improvements to a new version of TRADE, to provide it with the possibility of obtaining better solutions with larger search space, and, in general, investigating trade-offs between the quality of the solution and the speed of obtaining it.

29 CLB Reusing

In the previous section, we have presented the encoding algorithm for the bond set, but we didn't mention the possibility that some CLBs in the CLB Pool might be reused. In TRADE, each previously generated CLB is recorded in a 32-bit long word, called a *CLB frame*. Because there can be up to five inputs to each CLB, thirty two bits are required to store all possible combinations of the inputs ($2^5 = 32$). For example, the logic of CLB h:

```
.names a b c d e h
10-01 1
11101 1
01-0- 1
```

is stored in the program as:

```
31                              0  Minterm position
00100000001000100011001100000000  CLB h
```

Each position in the above frame corresponds to a minterm. For example, minterm 11101 resides in the position 29. If a minterm belongs to the ON set, there is a 1 in the corresponding position of the frame. Otherwise 0. By proper coding it is possible to reuse CLBs in the CLB Pool.

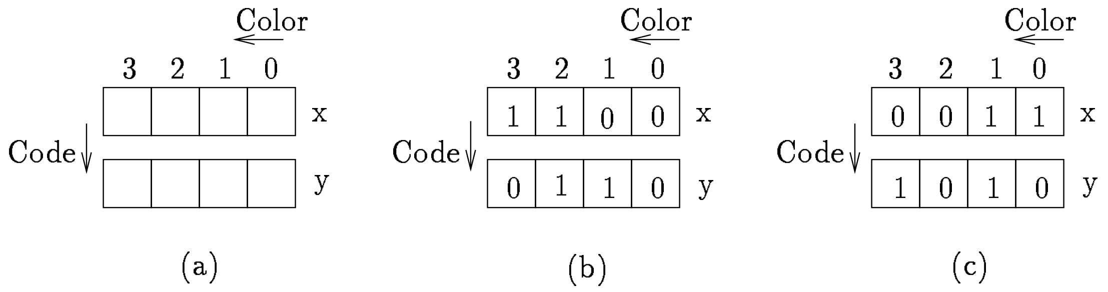


Figure 73: CLB reusing concept

The basic idea of CLB Reusing is originated from the following facts:

Suppose that the column multiplicity of a Karnaugh map is four (it is the same as saying that there are four colors). We would need two ($4 \leq 2^2 = 4$) variables (x and y as shown in Figure 73a) to encode the bond set.

We can code variables xy as 00, 01, 11, 10 as shown in Figure 73b. There are $4! = 24$ possible ways to code xy. But, if in the CLB Pool there is a CLB that can implement the pattern 0011 (call it x, this is the pattern for coding as shown in Figure 73c, not the internal storage frame). If we code y as 1010 (y is a newly created CLB) as shown in Figure 73c, we have realized the coding of xy by using one CLB in the CLB Pool and a newly created one. The y is coded in such a way that no two xy Codes are identical. Next example is used to show the detailed procedure of CLB Reusing algorithm. Suppose that we have a function with seven input variables a, b, c, d, e, f, g as shown in Figure 74.

After graph coloring, six colors are obtained as shown in Figure 75. In Figure 75, columns 0, 1, 3, 4, 7, 9, 12, 14, 15 and 17 belong to color 0, columns 18, 19, 22, 23, 25, 26 and 27 belong to color 1, and so forth. A careful reader will find that columns 20 and 29 are missing. They are DC columns and can be put in any color position.

Because there are six colors (column multiplicity is six as well), we need three CLBs ($3 > \log_2 6 = 2.585$) to encode the bond set. Suppose that in the CLB Pool there are three CLBs which have the same input variables as that in the current bond set, they are CLBs w, x and y.

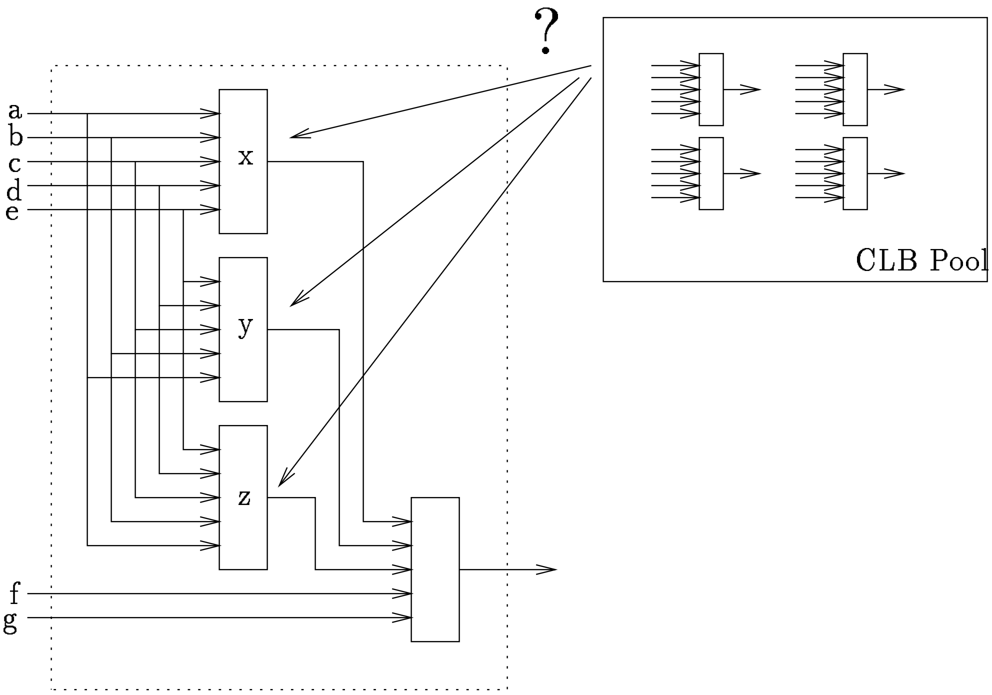


Figure 74: CLB reusing example

```
00010000101010010010100000001011   w
01001110111011101111011110011011   x
10011110110011000000000001100100   y
```

We try to use as many CLBs in the CLB Pool as possible. If the CLBs in the CLB Pool can fully cover the columns in some colors, they might be reused. The term "fully cover the columns in some colors" means that in the CLB frame all positions corresponding to the columns included in some colors must be exactly 1, while all the other positions must be 0. We test if the CLBs w, x and y can fully cover the columns in some colors in Figure 75. It is found that the CLB x can fully cover the colors 0, 1 and 3, because in the CLB x frame all positions correspond to the columns included in the color 0, 1 and 3 are 1, all the others are 0. CLB x has the pattern 001011 as shown in Figure 76. The same test results in that CLB y can fully cover the colors 1 and 2, and it has the pattern 000110 as shown in Figure 76. While CLB w can cover the columns 0, 1 and 3 in the color 0, but it cannot cover the columns 4, 7, 9, 12, 14, 15 and 17 in the color 0, this is not a full cover. Therefore CLBs x and y can be reused, but CLB w can not be reused. We add a new CLB (CLB z) and code it in such a way that each color has a different Code bit. The procedure for coding the newly added CLB is as follows:

> Start from the color position 0 in Figure 76. CLB y and CLB x give the bits 0 and 1 (Code 01. Code is read vertically, not horizontally), respectively. Put a 0 at the color position 0 of CLB z as shown in Figure 76. In the color position 1, the corresponding bits from CLB y and CLB x are 1 and 1 (Code 11) respectively. Because Code 11 is different from Code 01 in the color position 0, put a 0 again at the color position 1 of CLB z. The same way, put a 0 at the color position 2 of CLB z. In the color position 3, the Code is 01, it is the same as that in the color position 0. Increase the Code put in the color position 0 of CLB z (that is 0) by 1, this leads to a 1. So put a 1 at the color position 3 of CLB z. Perform the same operation for the color positions 4 and 5.

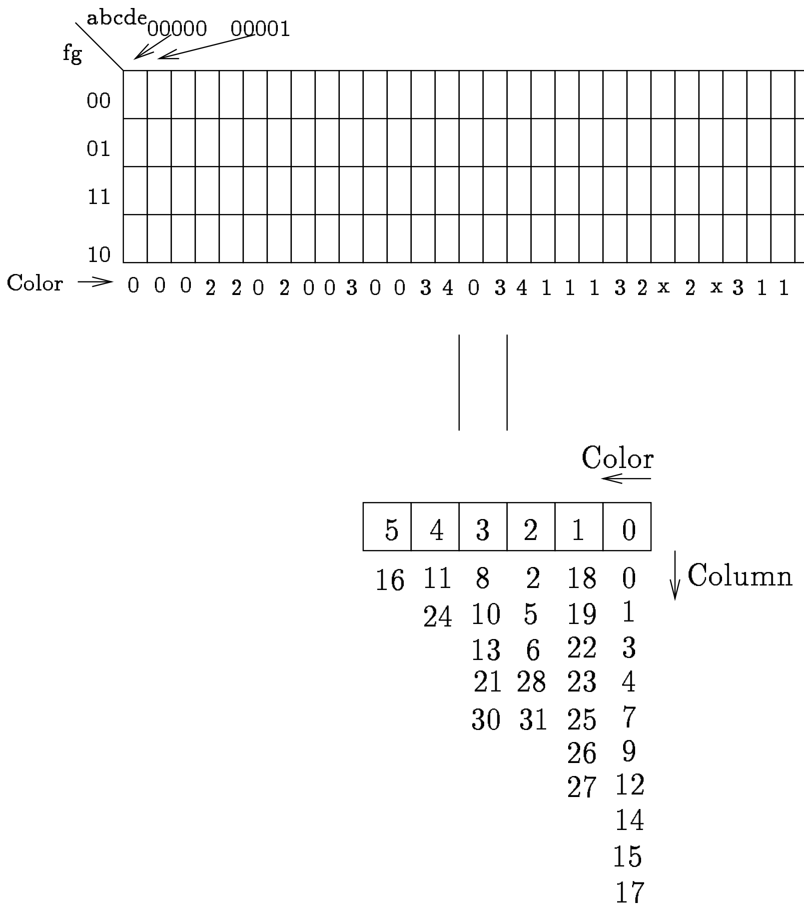


Figure 75: After graph coloring

Code ↓ / Color ←	5	4	3	2	1	0
CLB z	1	0	1	0	0	0
CLB y	0	0	0	1	1	0
CLB x	0	0	1	0	1	1
Column ↓	16	11 24	8 10 13 21 30	2 5 6 28 31	18 19 22 23 25 26 27	0 1 3 4 7 9 12 14 15 17

Figure 76: The final coding

The final coding of CLB z is 101000 as shown in Figure 76.
The newly generated CLB z has the internal storage frame as:

```
01000000001000010010010100000000    z
```

It is formed according to the pattern 101000 of CLB z. The color position 3 is 1 in the pattern of CLB z. Columns 8, 10, 13, 21 and 30 correspond to this color, so in the CLB z frame, the positions 8, 10, 13, 21 and 30 are filled with 1. The same operation is performed for the color position 5, and the internal storage frame of CLB z is obtained. By proper coding, we have reused two CLBs in the CLB Pool. The pseudo-code for CLB reusing is shown in Figure 77.

```
CLB_reusing()
{
collect reusable CLBs;
fully cover testing;

for (i = 0; i < total_color; i++)
  {
  if (reused_CLB_code_repeat(i) == true)
    code(i)++;

  new_CLB_code(i) = code(i);
  }
}
```

Figure 77: Pseudo-code of CLB reusing

30 Dealing with Vacuous Variables.

The advantage of removing vacuous variables during decomposition is reducing the size of the problem for the decomposition. This method has been used by Luba et al. Luba presents how to deal with many vacuous variables - he uses and efficient Boolean method adopted from Espresso.

In general the experience from the literature about vacuous variables is mixed or negative. Definitely a naive approach can give bad results, on the other hand some reduction may be very useful. This problem was not much analyzed and should be one of research tasks. Definitely, the totally vacuous variables should be always removed.

30.1 Creating Partitions.

Algorithm to select partitions in Flash starts with one variable in output block and n-1 variables in input block, this is the same as building BDD from the top. This gives a table with two rows and 2 to n-1 columns.

Another possibility is to start from n-1 variables (or n-2) variables in output block which would correspond to building BDD from the bottom. This approach corresponds to BDD building procedures by Haomin Wu and Ron Kohavi. The method by Craig Files is in essence very similar to the algorithm of Haomin Wu, but the second method is able to deal better with don't cares.

An algorithm presented by Perkowski, Dysko and Falkowski used a learning method with a modifiable weighted evaluation function a'la Samuel Checkers Program. This method can be perhaps further improved by having signature tables instead of linear weighted functions. Such approach was suggested by Armstrong [25].

Another technique close to variable ordering discussed in literature is called K-map folding. The question is "How to combine folding the K-maps with looking for partitions?" Folding has been used especially for incompletely specified functions, so these techniques might offer some new insights to variables' partitioning and ordering.

There is a large body of algorithms and theory of partitioning large and very large graphs which has been developed in the context of VLSI layout - Kernighan, Richie, Fiducia, Mattheyes, Szepieniec. The open question is this - "Can one use such algorithms for Boolean Decomposition?"

Another idea is to add VARIOUS biases to partitioning: symmetry, partial symmetry, other properties used for function classification, for instance as in the research of Malgorzata Marek-Sadowska [561]. Because in decomposition you need heuristics in any case, why don't we use various biases towards symmetry or other types of functions for which synthesis may be easier.

31 Is DFC a good measure of decomposition quality?

Is DFC-controlled Decomposition Search a Good Measure for Machine Learning? How is Flash compared to other methods? If one will use another cost function than DFC, how the learning quality would be affected?

Complexity minimization of a circuit by a decomposition controlled by DFC is in our opinion not a good method for some categories of functions, especially multi-output symmetric functions. This is because it will not create the circuit with actual minimum of DFC, or especially any other circuit complexity measure involving connection "complexity", like counting literals. Flash is not solving the problem because it can end up with large lookup table blocks which are practically non-realizable.

An example of weakness of Flash in this respect is a set of all symmetric functions of 5 variables. In standard symmetric realization it requires 1+2+3+4+5 demultiplexers and 1+2+3+4 2-input OR gates. Since some functions of 5 variables are non-decomposable, LOOKUP Table realization would require at least two lookup table of 5 inputs each, a total of DFC = 32 * 2. May be, however, this problem does not exist asymptotically, i.e. for large functions.

Another related problem is efficient realization of recorder functions in case the total symmetry of a subset of variables was used in the decomposition. The problem is well known from the literature, for instance it is discussed by Kim and Dietmeyer [332].

It is believed [537] that DFC calculated in terms of 2-input gates is close to the number of literals. This belief should be verified on examples.

32 Binary Decision Diagrams versus Decomposition.

A point that is rarely made (besides the papers of Tsutomu Sasao, and Massud Pedram et al) and which we found very important, is the direct relation of the decision diagrams, networks of multiplexers, and AC decompositions. The cut in an ordered decision diagram (with a root node at the top) separates variables into two distinct, non-overlapping groups: variables that are above the cut, and those that are below the cut. The variables that are above the cut correspond to the free variables in AC decomposition, and using another terminology, to address variables of the output multiplexer. The variables that are below the cut correspond to the bound set variables in AC decomposition, and in other terminology, to the input variables that drive the data-inputs of the output multiplexer.

This relation is very useful for AC Decomposition for the following reasons:

1. There is a big body of sophisticated methods and algorithms for efficient variable ordering in Decision Diagrams. All those methods can be used either directly or indirectly. A Direct Method is to find the order of variables for partitioning as it were a decision diagram, but actually without implementing a diagram as a data structure in memory. An order of variables a, b, c, d,... corresponds to a sequence of sets: a, a,b, a,b,c, a,b,c,d... These sets can be either sets of bound variables, or sets of free variables. This is the approach used by Haomin Wu and Marek Perkowski [697, 698], and by Craig Files [215].

   Quinlan and other authors who work on decision diagrams are using entropy for variable ordering. Is it better than other methods for variable ordering? There are several other authors that use entropy for variable grouping and ordering, but there are no papers that would compare these approaches. Moreover, entropy based methods are used in general systems to separate variables to overlapping subsets, and thus such methods can be used in non-disjoint decompositions. There are no papers with this observation, not any experimental comparisons. This is one of our areas of research.

   Another and simpler approach is to use the BDD ordering methods in the decomposition just by applying a BDD program from SIS or other package to find a good diagram. Then one can create a sequence of variables in this diagram and look for the best cut in this order. We do not know of anybody using this approach.

   Yet another approach is to create some special decision diagram and apply the above technique. This is the approach applied by both Massud Pedram et al and Tsutomu Sasao.

2. Similarly, the methods that have been created for networks of multiplexers are equally applicable. For instance, Schaefer and Perkowski use the method developed by Almaini et al to create Kronecker Decision Diagrams. This approach has powerful advantage from the point of view of Pattern Theory, since it is well-suited to the strongly unspecified functions. Analogously, this approach can be applied to AC decomposition as well.

3. There is recently much research about new kinds of decision diagrams that have a decreased node count [590, 591, 592, 561, 338, 475, 326, 598, 55] and have been created for strongly unspecified functions. Some of them [338, 475] have been created for machine learning applications, this research can be again used to find a good variable sequence for AC decomposition.

4. Because decision diagrams are just factorized decision trees, the good variable ordering for a tree is usually a good order for a corresponding decision diagram. Therefore the results of the research on finding the variable order in programs such as C4.5 may be useful as well for finding good variables sequence.

5. There exist variable ordering algorithms [490] which use a combination of many methods to find a good order. These methods can be used to combine the best heuristics of the above methods in

the case of tie break of other methods (this method in a very simplified variant was also used by Craig Files [215]).

6. An interesting point in variable partitioning is that both the variant with increasing step-by-step the free set of variables, and the variant of increasing the bound set of variables have been tested in an older versions of Flash. These two algorithms correspond to two variable ordering methods applied in the decision diagrams. While most researchers find the order of the variables by building the diagram from the root to leaves [590], some other build the graph from the leaves to the root. This is an approach applied by both H. Wu and M. Perkowski and R. Kohavi. There is one more approach, the variable sifting algorithm, which gives currently the best results for BDDs but is the most complex from the programming point of view. This approach has been created by Richard Rudell in his BDD package [558] and by Rolf Drechsler in his KDD package [179]. The disadvantage of using this approach directly, i.e. on a completed diagram, might be obtaining poor quality results in case of the completely specified diagrams (one with terminal nodes: 0, and 1) were originally created. A solution to this problem would be to create from scratch an incompletely specified diagram of a kind described by Massud Pedram. Such diagram has three types of terminal nodes: 0, 1 and DC). However, as far as we know, there are no algorithms in the literature yet to implement the sifting algorithm of Rudell, or another order modifying method to this kind of diagrams.

The advantage of these methods is that they are all in the exact mainstream of logic synthesis research and industrial development, so many people work on them and there is much progress in this area recently. Re-using their results in the less popular and "crowded" area of decomposition seems to me like a practical and good idea, of not "reinventing a wheel".

33 Search Strategies.

Several search strategies have been used in the literature to find the best partitions for both the disjoint and non-disjoint cases. For the disjoint case, the following methods are known:

1. Depth First search (we do not recall any particular papers, but this is the most natural method).
2. Breadth First search (one variant in Flash, designed by Mark Axtell).
3. Best Bound (Haomin Wu/Perkowski), and Limited Best Bound search (Haomin Wu, Craig Files).
4. A* algorithm of Artificial Intelligence (Haomin, Perkowski). This can be seen as the generalization to the Best Bound method.
5. Genetic Search (Mike Noviskey, Prof. Frenzel from Idaho [226]).
6. Search through all subsets of a set of variables, (Shen/Kellar, Varma/Trachtenberg).
7. predetermined order based on a heuristic of evaluating pairs of variables (cube based approach in presented by Wei Wan and Perkowski [678]).
8. Add and subtract variables from and to bound and free sets - Steinbach.
9. Select a single variable, based on complex block separation criteria based on partition calculus heuristics (Luba).
10. Other BDD variable selection methods (Friedman).
11. Other decision tree like variable selection methods (C4.5 - Quinlan).

34 Approaches to Graph Coloring in Literature

As explained in the introduction, our general plan is to investigate relations between graph coloring and Column Minimization. With this plan in mind, we have started our investigations in algorithms for the Graph Coloring and Maximum Clique problems. It is interesting, that although the number of papers on these topics is very large, there exist actually very few papers (and all of them relatively new) that deal with practical exact and high quality heuristic algorithms. In this section we will report those results from the point of view of using them in the research outlined.

Given is graph G with set of nodes N. Let us denote $card(N) = nn$. The *(Proper) Graph Coloring* of graph G is to find a color for each node of G such that every two adjacent (i.e. linked by an edge) nodes will have different colors. The set of all colors is denoted by C. The minimum cardinality of colors to color graph G is denoted by $\kappa(G)$, and called its *chromatic number*.

Graph Coloring has been proven to be an NP-complete problem Why is the coloring problem so hard? Is it "harder" than some other NP-hard problems used in CAD?

Although for some NP-hard problems good approximate algorithms exist, coloring is different: none of the algorithms found in literature does a good job for coloring large (n ¿ 1000) graphs [410, 411] See also the cited above theoretical result of Johnson.

We will review the exact algorithms first. The algorithms for finding chromatic number in an exact way are given in Hammer and Rudeanu [272], Christofides [139, 138], Persin [507], Wang [680], and Brelaz [100]. They were applied to graphs with not more than 100 (check it) nodes. Some algorithms, such as Maghoute's (cited after [691], use the method of finding all maximum independent sets and next covering nodes with independent sets. This method is totally analogous to the classical Quine-McCluskey algorithm for SOP minimization, so using this method to find exact graph colorings could take advantage of high-quality modern Boolean minimizers. (such as Espresso-Signature).

We are interested in other Graph Coloring methods based on backtracking, which may be will be more efficient than our graph-coloring algoritms.

Most of the approximate graph coloring algorithms belong to the greedy category. Greedy algorithm colors nodes one by one in some prespecified order not coloring neighbors with the same color.

All other algorithms use one of the following heuristics [410, 411]:

A. A node of high degree is harder to color than a node of low degree.

B. Nodes adjacent to the same set of nodes are colored with the same color.

C. Try to color as many nodes as possible with every color already used.

D. Color with the smallest color available.

There are two kinds of Graph Coloring Algorithms: Sequential (Preset) and Recursive (Adaptive).

34.1 The Sequential (Preset) Algorithms

These algorithms decide the order of coloring of nodes before the coloring.

1. The LARGEST_FIRST algorithm of Welsh and Powell [683] orders nodes by descending degree. Then colors them greedily.

2. The SMALLEST_LAST algorithm of Matula, Marble and Isaacson [420] selects the node of the lowest degree to color last, removes it and repeats the process. Then colors greedily beginning with the last node removed.

3. Algorithm of Williams [688] orders nodes in descending degree by considering neighborhood degree sums, and so on, instead of the degrees.

4. Syslo (1989) comments on the relations between sequential coloring and the Welsh-Powell (1967) upper bound for the chromatic number of a graph. He suggests that new ordering may lead to improved results. His book (1983) [648] has section 4.1 devoted entirely to coloring algorithms and extensive bibliography.

34.2 The Recursive Sequential (Adaptive) Algorithms

1. The recursive algorithms can change order recursively during coloring. They use heuristics A,B,C,D dynamically.
2. The algorithm of Johri and Matula.
3. The algorithm by Tehrani [652].
4. The algorithm of Leighton [366].
5. The algorithm of Wood [694] defines a similarity measure on pairs of nodes and next examines nodes in pairs in order of decreasing similarity, sometimes coloring and sometines not, depending on degrees of the pair and whether or not one of the two is already colored.
6. The algorithm of Johnson [310] selects nodes of low degree and constructs large independent sets.
7. The algorithm by Brelaz, called DSATUR, uses heuristics A and B, [100]. The color-degree of node v is the number of colors used so far on vertices adjacent to v. DSATUR repeatedly chooses a node of largest color-degree, and assigns to it the smallest possible color. If two nodes have equal color-degrees, the one adjacent to most uncolored nodes is colored first. DSATUR colored graphs with 100 nodes and gave solutions worse than few nodes. By some authors it is treated as the best algorithm for small graphs [410, 411]
8. The really interesting Boolean functions produce graphs of 1000 nodes. According to [410] all those algorithms give poor results on graphs with 1000 nodes.
9. An interesting hypothesis is given in [411]. The authors suggest that the "failure in coloring large graphs may be due, at least partially, to the way in which κ is determined in large graphs".

35 The Maximum Clique Problem

The maximum clique of graph G is the maximum independent set of graph $\overline{G}$, the complement of graph G. Both of these problems can be reduced to linear integer programming [139].

There are several important papers on maximum cliques in our references. Several algorithms have been written, computer programs implemented, and even compared for this problem, all of them more efficient than the classical old approaches referenced by most authors. There are especially many good papers from former Soviet Union that include deep mathematical theory for Graph Coloring. Below we will provide bf only partial literature on the subject. (I have more positions but I have no time to find them all now.) Some algorithms are better for planar, triangle-free, bipartie, etc graphs. There are many mathematical results and timing/complexity comparisons. Since a maximum clique for graph G is a maximum independent set for the complement graph of G, then the results on the maximum independent sets should be utilized as well.

There exist the well known relations among maximum clique generation, clique partitioning, clique covering and coloring of a complement graph, that are used by several authors to find good solutions to this or very similar problems.

In our opinion the area of maximum clique generation is very competitive, even more than decomposition. Therefore the results of this research may prove very useful in creating efficient decomposers.

There are, basically, two groups of methods. Some methods start from small cliques and grow them, other methods start from large groups and remove nodes until cliques are found. It seems that it depends on the type of graph which method is better. Perhaps for graphs of Boolean functions it is better to start from small cliques and grow them instead of starting from large groups and shrinking them. In any case, which particular method is better, depends perhaps on the density, type and other properties of the graph. What is the experimental evidence of the properties of graphs that are created for decomposition? Do they have any particular properties that make this algorithm better than the well known algorithms?

Important algorithms for maximum cliques are in: [50, 51, 89, 188, 168, 169, 327, 262, 252, 252]. Paper [651] is a very important paper, also, chapter 9 in [248] is worth reading.

Many algorithms for proper graph coloring and related problems exist in the literature, [100, 105, 106, 138, 139, 143, 187], [193, 264, 258, 202, 238, 238, 240, 248, 310, 366], [683, 528, 507, 438, 420, 419, 411, 410, 652, 680, 688], [694, 691, 688].

36 Conclusions

Based on the surveyed literature, the following observations seem to be crucial:

1. The most important aspect of a successful Functional Decomposer is the representation of data. From all representations in the literature, the most successful programs use Binary Decision Diagrams and Edge-Valued Decision Diagrams [351], lists of disjoint ON and OFF cubes [638], and variable-partitions on ON and OFF minterms or cubes [397]. The last representation has very interesting properties. It allows to define, on incompletely specified functions, several operators, that have not been defined before. This leads to efficient variable partitioning and column compaction algorithms, as well as to formulating new kinds of decompositions, including a new method of "variable re-using". Additional advantage of this representation is its easy generalization for multiple-valued logic. On the other hand, the disadvantage of this approach is the use of large lists, that are necessary to represent the ON and OFF sets of minterms or cubes. Hence comes our idea of encoding these large sets as two BDDs: the ON BDD, and the OFF BDD. All such sets for all input and output variables, together with the newly created sub-functions, can be represented as a "Shared" BDD. Such representation is totally new in the literature, and has all advantages of the previous ones, and none of their respective disadvantages. A ready public domain software package can be used to represent "reduced, ordered, shared BDDs with negated edges", since the only operations that we use are the standard ones: intersection, union and complement.

2. The representation allows to treat all newly created sub-functions as additional inputs variables, by calculating the partitions done by these functions on ON and OFF cares. Then, all new variables can be treated in the same way as the original initial input variables - the analysis of vacuous variables and variable partitioning are applied to them. This allows for powerful function sharing and investigating a wider space of decompositions.

3. Another advantages of successful modern decomposers that are missing in Flash are: multi-output functions, and multiple-valued functions. A careful analysis of all operations used in the respective systems, suggests that the proposed above representation will allow easy generalization to multi-output and multiple-valued functions. In essence, it is based on approaches that have been already created or generalized for these cases. Moreover, our representation is well suited not only for AC decomposition, but for many other decomposition variants [638, 351], [397, 61, 70, 116], [175, 176], [332]. Especially, it allows to take into account the symmetry, as a most natural design bias. Moreover, this realization easily allows us to represent the "generalized don't cares", that we invented [486] and found useful in the decompositional approach to Machine Learning.

4. One more advantage of the representation is that it has been created to solve efficiently the two most important problem in decomposition: selecting the best bound sets, and minimizing the number of columns/encoding problem. The sets of minterms are kept separated in the decomposition; for each input variable, output variable and sub-function. This representation allows for convenient formulation of many partial problems associated with decomposition. Many heuristic or methodical criteria to select variables in order to achieve some separation goals can be easily defined. All information that is lost and repeatedly found in classic decomposition algorithms, is retained here for further use.

5. All NP-hard combinatorial problems have been reduced here to **a single one**: the weighted covering problem. Excellent algorithms exist [491, 397] to solve this problems, and many publications are devoted to it.

6. Several search strategies have been used in the literature to find the best partitions for both the disjoint and non-disjoint cases. A parameterized program can be created that would allow to easily create and compare several of them:

the depth-first search;

the depth-first-search-with-one-successor;

the breadth-first search (a variant in Flash by Mark Axtell);

the best-bound (Haomin Wu/Perkowski [697]);

the limited best-bound search (Haomin Wu, Craig Files [215]);

the A* algorithm of Artificial Intelligence (Haomin Wu, Perkowski);

the searching subsets of a subset of variables (Shen/Kellar [607, 608, 609], Varma/Trachtenberg [701]);

using the predetermined order based on a heuristic of evaluating pairs of variables (cube based approach in presented by Wei Wan and Perkowski [678]);

the search by "add and subtract" variables from and to bound and free sets - Steinbach [637, 638];

the search by selecting a single variable, based on complex block separation criteria of partition calculus heuristics (Luba [376, 382, 392, 397]);

other BDD variable selection and ordering methods [179, 591, 559];

and

other decision tree like variable selection methods (such as in the C4.5 program by Quinlan).

7. It would be very useful for future research to use a tool (from Mentor, Synopsys, or MIS II) to draw the structures created in decompositions. We would be able to find symmetries, linearities, etc. in some known functions of some known properties. This would perhaps help to find good heuristics for future decomposer versions.

8. One should investigate a relation between functional decompositions and iterative circuits. An iterative circuit can be often created by a class of AC decompositions (since the number of outputs from a block in an iterative circuit is often the same or larger than the number of inputs). The iterative circuit has, however, a high regularity, which can be used to increase the search efficiency of the decomposition.

9. Circuits that include blocks which have more outputs than inputs are quite common in practical design, so the respective new kinds of non-Curtis FD decompositions should also be interesting.

10. Currently, a symbol is encoded in binary. It would be useful to be able to use the variables from the symbol all together as a group, while performing a decomposition. The property of grouping variables to bond sets would be also useful for decomposition of fuzzy functions, where variables x and $\overline{x}$ should always come together.

11. Decision Diagram-based methods may provide better ways of sharing sub-functions during decomposition. It seems that the problem of variable sharing is important and it was not investigated in the literature.

12. We should investigate the complexity of connections - the "communication" complexity, versus the "computation" complexity calculated now, as related to various decomposition methods. Is it true that the communication complexity is close to DFC? Facts of this kind should be verified experimentally.

13. The synthesis methods based on correlations of input and output variables should be investigated, since their results can be used for AC decomposition. Such ideas appeared in papers of Oliviera and Zaki [740] and Mike Noviskey [469].

References

[1] H. Abelson, A. Ehrenfeucht, J. Fickett, J. Mycielski, "Compositional Complexity of Boolean Functions," *Discrete Applied Mathematics,* Vol. 4, No. 1, pp. 1-10, 1982.

[2] P. Abouzeid, et. al., "Multilevel Synthesis Minimizing the Routing Factor," *Proc. 27th ACM/IEEE Design Automation Conference,* pp. 365-368, June 1990.

[3] P. Abouzeid, et. al., "Lexicographical Expression of Boolean Functions for Multilevel Synthesis of High Speed Circuits," *VLSI Logic Synthesis and Design,* pp. 31-39, 1990.

[4] P. Abouzeid, et. al., "Lexicographical Factorisation Minimizing the Critical Path and the Routing Factor for Multilevel Logic," *IFIP Workshop on Logic and Architecture Synthesis,* pp. 219-228, Grenoble 1990.

[5] P. Abouzeid, B. Babba, M. Crastes de Paulet, and G. Saucier, "Input-Driven Partitioning Methods and Application to Synthesis on Table-Lookup-Based FPGAs," *IEEE Tr. on CAD* Vol. 12, No. 7, pp. 913–925.

[6] Abtech Corporation, company materials, 1994.

[7] S.B. Abugharbieh and S.C.Lee, "A Fast Algorithm for the Disjunctive Decomposition of m-Valued Functions, Part I: The Decomposition Algorithm," *Proc. 23-rd ISMVL,* pp. 118-125, 1993.

[8] S.B. Abugharbieh and S.C.Lee, "A Fast Algorithm for the Disjunctive Decomposition of m-Valued Functions, Part II: Time Complexity Analysis," *Proc. 23-rd ISMVL,* pp. 126-131, 1993.

[9] Y.S. Abu-Mostafa, *"Complexity in Information Theory",* Springer-Verlag, New York, 150pp, ISBN 0-387-96600-5, 1988.

[10] G.P. Agibalov, "Problemy Sinteza Cifrovych Automatov," pp. 96-100, Moscow, 1967, (in Russian).

[11] M.O. Albertson, and K.L. Collins, "Duality and perfection for edges in cliques," *J. Comb. Theory.* Ser. B., Vol. 36, No. 3., pp. 298–309, June 1984.

[12] C. Aleksander, "Notes on the Synthesis of the Form," Cambridge, MA, *Harvard Univ. Press,* 1967.

[13] I. Aleksand., *Radio El. En.* 38,28, 1969. TK6540 .R668

[14] C.M. Allen, and D.D. Givone, "A Minimization Technique for Multiple-Valued Logic Systems," *IEEE TC*, Febr. 1968, pp. 182–184.

[15] S.B. Akers, "On a Theory of Boolean Functions," *Journal of the Society for Industrial and Applied Mathematics,* 7, 4, 1959.

[16] S.B. Akers, "A Diagrammatic Approach to Multilevel Logic Synthesis," *IEEE Trans. on Comput.,* Vol. 14, pp. 174-181, April 1965.

[17] S.B. Akers, "Decompositions of Logical Functions Using Majority Decision Elements," 1965.

[18] A. E. A. Almaini, M. E. Woodward, "An Approach to the Control Variable Selection Problem for Universal Logic Modules," *Digital Processes,* Vol. 3, pp. 189-206, 1977.

[19] A.E.A. Almaini, "Sequential Machine Implementations Using Universal Logic Modules," *IEEE Trans. on Comput.* Vol. 27, No. 10, pp. 951-960, 1978.

[20] A. Apostolico, and E. Hambrusch, "Finding maximum cliques on circular-arc graphs," *Inf. Proc. Letters*, Vol. 26., No. 4, December 1987, pp. 209–215.

[21] T. Araki, T. Kameda, "A Theory of Coteries. Mutual Exclusion in Distributed Systems," *IEEE Trans. on Parallel and Distributed Systems,* Vol. 4, No. 7, pp. 779-794, July 1993.

[22] H. Arango, J. Santos, A. Chacur, "Ternary Cyclo Decompositions," *IEEE Trans. Comput.*, Vol. C-17, No 12, pp. 1175-1176.

[23] C. Arcelli, *Kybernetik,* Vol. 13, pp. 155-?, 1973.

[24] W.W. Armstrong, and G. Godbout, "Properties of Binary Trees of Flexible Elements Useful in Pattern Recognition," *Proc. IEEE 1975 Int. Conf. Cybern. and Society*, San Francisco, pp. 447-450.

[25] W.W. Armstrong, J. Gecset, "Adaptation Algorithms for Binary Tree Networks," *IEEE Trans. on Systems, Man and Cybern.*, Vol. 9, No. 5, pp. 276-285, 1979.

[26] A.R.A. Asaad, L.A.M. Bennett, "Synthesis of Tributary Switching Networks by Spectral Means," *IEE Proc.* Vol. 137, Pt.E, No. 5, September 1990.

[27] R.L. Ashenhurst, "The Decomposition of Switching Functions," *Bell Laboratories Report,* No. BL-1(11), pp. 541-602, 1952.

[28] R.L. Ashenhurst, "Non-disjoint Decomposition," *Bell Laboratories Report,* No. 4, pp. 1V-1 - 1V-12, 1953.

[29] R.L. Ashenhurst, "The decomposition of switching functions," *pp. 571-603 of Curtis'62.*, 1952.

[30] R.L. Ashenhurst, "The Decomposition of Switching Functions," *Bell Telephone Labs Report*, 16, pp. III1-III72, 1956.

[31] R.L. Ashenhurst, "The Decomposition of Switching Functions," *Proceedings of International Symposium of Theory of Switching,* 1957.

[32] R.L. Ashenhurst, "The Decomposition of Switching Fuctions," *Proceedings of an International Symposium on the Theory of Switching,* April 2-5, 1957, Ann. Computation Lab., Harvard Univ., Vol. 29, pp. 74-116, 1959.

[33] F.F. Atstonas, and K.I. Plukas, "A Method of Minimization of Microprograms for Electronic Digital Computers," *Avtomatika i Vycisl. Tehn,* Vol. 4., pp. 10–16, 1971.

[34] J.G. Auguston, and J. Minker, "An Analysis of Some Graph Theoretical Cluster Techniques," *JACM,* Vol. 17, No. 4, pp. 571–588, October 1970.

[35] M.L. Axtell, "Partition Selection Algorithms: Row/Column Ratio Experiment," *Technical Report,* Veda Inc., c/o WL/AART-2, W/P AFB, OH 45433-7408, April 1993.

[36] M.L. Axtell, T.D. Ross, M.J. Noviskey, "Performance Comparison Between Occam-Based, Back-Propagation, and Abduction Learning Networks," In *IEEE Int. Conf. on Neural Network Applications in Signal Processing,* 1993.

[37] L. Babai, P. Hajnal, E. Szemeredi, G. Turan, "A Lower Bound for Read-once-only Branching Programs," *Journal of Computer and System Sciences,* Vol.35, pp. 153-162, 1987, QA76.5 .J7.

[38] B. Babba, M. Crastes, "Automatic Synthesis on Table Lookup-Based PGAs," *Proc. Euro-ASIC*, 1992.

[39] U.G. Baitinger, H.M. Lipp, D.A. Mlynski, D.A. K. Reiss, "MEGA-Ein Modulares Entwurfssystem fuer Gate-Arrays," *Fortschritt-Berichte VDI,* Reihe 9, Nr. 65, VDI-Verlag, Duesseldorf 1987, (in German).

[40] E. Balas, and C. Yu, "Finding a maximum clique in an arbitrary graph," *SIAM, J. Comput.*, Vol. 15, No. 4., pp. 1036, November 1986.

[41] E. Balas, V. Chvatal, and Nesetrily, "On the maximum weight clique problem," *Math. Oper. Res.*, Vol. 12., No. 3, pp. 522-535, August 1987.

[42] R.B. Banerji, *Pattern Recognition,* 1, 63, 1968.

[43] J. Bankowski, "Difference Theorems on Decomposition of Boolean Functions," not published manuscript, *Warsaw Technical University.* 1971, (in Polish).

[44] S.I. Baranov, V.N. Siniev, *Avtomaty i Programmirujemyje Matricy,* Minsk 1980, (in Russian).

[45] D.F. Barnard and D.F. Holman, "The Use of Decomposition Algorithm in Multilevel Design of Circuit," *Computer J.*, Vol. 11, pp. 269-276, 1968.

[46] K.A. Bartlett, R.K. Brayton, G.D. Hachtel, R.M. Jacoby, C.R. Morrison, R.L. Rudell, A.L. Sangiovanni-Vincentelli, A.R. Wang, "Multilevel Logic Minimization Using Don't Cares," *IEEE Trans. on CAD,* Vol. 7, pp. 723-740, June 1988.

[47] A.R. Barron, and R.L. Barron, "Statistical Learning Networks: A Unifying View," *Symposium on the Interface: Statistics and Computing Science,* 1988.

[48] N.R. Bell, *IEEE Educat.* 11, pp. 173-?, 1968.

[49] L.A.M. Bennett, "The Application of Map-Entered Variables to the Use of Multiplexers in the Synthesis of Logic Functions," *Int. J. Electronics,* Vol. 45, No. 4, pp. 373-379, 1978.

[50] R.G. Bennetts, "An Improved Method of Prime C-Class Derivation in the State Reduction of Sequential Networks," *IEEE Trans. Computers,* Vol. C-20, pp. 229–231, 1971.

[51] R.G. Bennetts, J.L. Washington, and D.W. Lewin, "A Computer Algorithm for State-Table Reduction," *Radio Electron. Engn.* Vol. 42, pp. 513–520, 1972.

[52] E.A. Bender, and J. Butler, "Enumeration of functions realized by fanout-free networks of general multi-valued gates," *Proc. 1979 ISMVL*, pp. 94-103, May 1979.

[53] E.A. Bender, E.R. Canfield, "Fanout-Free Functions," *American Mathematical Monthly,* Vol. 88, No. 4, pp. 278-280, 1981.

[54] E.A. Bender, E.R. Canfield, "Fanout-Free Networks of Symmetric Gates: Synthesis and Canonical Forms," *preprint Dept. of Math.* University of California at San Diego, La Jolla, CA 94093.

[55] B. Becker, R. Drechsler, and R. Wechner, "On the Relation Between BDDs and FDDs," *Technical Report*, University of Frankfurt, 12, 1993.

[56] E.A. Bender, "Asymptotic Methods in Enumeration," *SIAM Rev.*, Vol. 16, pp. 485-515, Oct. 1974.

[57] C. Berge, "Theory of Graphs," Methuen, London, 1962.

[58] T. Besson, H. Bousouzou, M. Crastes, and G. Saucier, "Synthesis on Multiplexer-based Programmable Devices Using (Ordered) Binary Decision Diagrams," *Proc. EURO-ASIC,* pp. 8-13, Paris, France, June 1992.

[59] T. Besson, H. Bousouzou, M. Crastes, and G. Saucier "Synthesis on Multiplexer-based F.P.G.A. Using Binary Decision Diagrams," *Proc. of IEEE ICCD*, pp. 163-167, 1992.

[60] P.N. Bibilo, S.V. Yenin, "Vychislitelnaia Tekhnika v Mashinostroenii," ed. 1, pp. 100-108, Minsk 1979, (in Russian).

[61] P.N. Bibilo, S.V. Yenin, "Decomposition of a Boolean Function With Minimum Number of Significant Arguments of the Subfunctions," *Engineering Cybernetics,* Vol. 18, No. 3, pp. 75-81, 1980.

[62] P.N. Bibilo, S.V. Yenin, "Joint Decomposition of a System of Boolean Functions," *Engineering Cybernetics,* Vol. 18, No. 2, pp. 96-102, 1980.

[63] P.N. Bibilo, *Izv. A.N. BSSR Ser. fiz-mat nauk,* No. 4, pp.33-38, 1980, (in Russian).

[64] P.N. Bibilo, S.V. Yenin, "Avtomatizacjia Logiceskogo Proektirovanija Diskretnych Ustrojstv," ed. 2., pp. 33-46, Minsk, 1980, (in Russian).

[65] P.N. Bibilo, S.V. Yenin, *Izvestia Academy of Sciences of USSR,* Series Technical Cybernetics, No. 2., pp. 108-113, 1980, (in Russian).

[66] P.N. Bibilo, *Avtomatizacija tekhn. podgotovki proizvodstva,* pp. 86-93, Minsk 1982, (in Russian). XXXXXXX

[67] P.N. Bibilo, *Avtomatizacija Tekhniceskoj Podgotovki Proizvodstva,* pp. 40-49, Minsk 1983, (in Russian).

[68] P.N. Bibilo, *Avtomatika i Vycislitelnaya Tekhnika,* No. 1, pp. 91-92, 1984 (in Russian).

[69] P.N. Bibilo, *Proektirovanije Ustrojstv Logiceskogo Upravlenija,* pp. 106-126, Moscow 1984, (in Russian).

[70] P.N. Bibilo, S.V. Yenin, "Synthesis of Combinational Networks Using the Method of Functional Decomposition," ("Sintez Kombinacjionnych Schem Metodami Funkcijonalnoj Dekompozicji,") *Nauka i Tekhnika,* Minsk, 1987, (in Russian).

[71] A.W. Biermann, J.R.C. Fairfield, T.R. Beres, "Signature Table Systems and Learning," *IEEE Trans. on Syst. Man and Cybern.*, Vol. 12, No. 5, pp. 635-648, 1982.

[72] A.J. Billionnet, "An upper bound on the size of the largest clique in a graph," *Graph Theory*, Vol. 5., No. 2., pp. 165-169, Summer 1981.

[73] Z.W. Birnbaum, *JSIA MATH,* 13, 444, .

[74] N.N. Biswas, "Identification of Totally Symmetric Boolean Functions," *IEEE Trans. on Comput.*, Vol. C-70, pp. 645-648, July 1973. Correction IEEE TC, pp. 863-864, September 1973.

[75] R.E. Bixby, *J. Comb. Th. B* 18, pp. 59, 1975, QA164 .A1 J6 ser.B

[76] V.D. Bliznyuk, M.F. Kholodny, "Decomposition of Boolean Functions Applying the Apparatus of Boolean Derivatives," *Automation and Remote Control,* USSR, Vol. 45, No.5, pp. 636-643, 1984.

[77] A. Blumer, A. Ehrenfeucht, D. Haussler, and M.K. Warmuth, "Occam's Razor", *Information Processing Letters*, Oct. 1987, pp. 377-380.

[78] D. Bochmann, "Zerlegung von Blockschaltungen," *Messen Steuern Regeln*, Vol. 11, No. 3, 1968, (in German).

[79] D. Bochmann, "Einfuehrung in die Strukturelle Automatentheorie," *VEB Verlag Technik,* Berlin 1975; Hanser Verlag, Muenchen-Wien 1975, (in German).

[80] D. Bochmann, Ch. Posthoff, "Binaere Dynamische Systeme," *Akademie Verlag,* Berlin 1981; Oldenbourg-Verlag, Muenchen 1981, (in German).

[81] G.V. Bochmann, W.W. Armstrong, "Properties of Boolean Functions with a Tree Decomposition," *BIT*, Vol. 13, pp. 1-13, 1974.

[82] D. Bochmann, F. Dresig, B. Steinbach, "A New Decomposition Method for Multilevel Circuit Design," *Proc. of the European Conference on Design Automation (EDAC),* Amsterdam, 1991.

[83] D. Bochmann, B. Steinbach, "Logikentwurf mit XBOOLE," *Verlag Technik,* Berlin, 1991.

[84] D.Bochmann, F. Dresing, B. Steinbach, "New Decomposition Method for Multilevel Circuit Design," *Proc. EDAC'92*, 1992.

[85] R. Bolton, 'Digital Systems Design with Programmable Logic," *Addison-Wesley Publishing Company,* 1990.

[86] B. Bollobas, and E.J. Cockayne, "Graph-theoretic parameters concerning domination, independence and irredundance," *J. Graph Theory*, Vol. 3., No. 3., pp. 241–250, Fall 1979.

[87] J.A. Bondy, "Bounds for the Chromatic Number of a Graph," *J. of Combinatorial Theory,* 7, p. 96, 1969.

[88] G. Boole, "An Investigation of the Laws of Thought on Which are Founded the Mathematical Theories of Logic and Probabilities," *London*, 1854.

[89] A. Bouchet, "An Algebraic Method for Minimizing the Number of States in an Incomplete Sequential Machine," *IEEE Trans. Computers*, Vol. C-17, pp. 795–798, 1968.

[90] E. Boros, V. Gurvich, P.F. Hammer, T. Ibaraki, and A. Kogan, "Decomposition of Partially Defined Boolean Functions," *DIMACS Technical Report 94-9*, March 1994.

[91] R.K. Brayton, and C. Mc. Mullen, "The Decomposition and Factorization of Boolean Expressions," In *Proc. of the Intern. Symp. on Circuits and Systems,* pp. 49-54, May 1982.

[92] R.K. Brayton, G.D. Hachtel, C. McMullen, A. Sangiovanni-Vincentelli, "Logic Minimization Algorithms for VLSI Synthesis," *Kluwer Academic Publishers,* Boston, 1984.

[93] R.K. Brayton, R. Rudell, A. Sangiovanni-Vncentelli, A.R. Wang, "MIS - a Multiple-Level Logic Minimization System," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems,* Vol. 6. No. 6, pp. 1062-1081, 1987.

[94] R.K. Brayton, "Factoring Logic Functions," *IBM J. Res. Develop.,* Vol. 31, No. 2, March 1987.

[95] R. Brayton and F. Somenzi, "An Exact Minimizer for Boolean Relations," R. Brayton and F. Somenzi, "An Exact Minimizer for Boolean Relations," Proc. of ICCAD, pp. 316-320, 1989.

[96] R. K. Brayton, G. D. Hachtel, and A. L. Sangiovanni-Vincentelli, "Multilevel Logic Synthesis," *Proceedings of the IEEE*, Vol. 78, No. 2, pp. 264-300, February 1990.

[97] R.K. Brayton, P.C. McGeer, J.V. Sanghavi, A.L. Sangiovanni-Vincentelli, "A New Exact Minimizer for Two-Level Logic Synthesis," *in T. Sasao, (ed), "Logic Synthesis and Optimization,* Kluwer 1993, pp. 1 –31.

[98] M.A. Breen, T.D. Ross, M.J. Noviskey, M.L. Axtell, "Pattern Theoretic Image Restoration," *Proc. SPIE '93 Nonlinear Image Processing IV* 1902-12. January 1993.

[99] M. Breen, T.D. Ross, M.L. Axtell, "Computing Column Multiplicity in Function Decomposition," *report, WPAFB*, 1994.

[100] D. Brelaz, "New Methods to Color Vertices of a Graph," *Communications ACM,* Vol. 22, pp. 251–256, 1979.

[101] Breuer, M.A. (ed.), "Design Automation of Digital Systems," Vol. 1, Prentice Hall, Englewood Cliffs, New Jersey, 1972.

[102] R.C. Brigham, and R.D. Dutton, "Graphs which with their complements have certain covering numbers," *Discrete Math.*, Vol. 34, No. 1., pp. 1-7, April 1981.

[103] C. Bron, and J. Kerbosch, "Algorithm 457 - Finding all cliques of an undirected graph," *CACM,* Vol. 16., No. 9., pp. 575 – 1973.

[104] I.I. Brona, "On a Kind of Decomposition of Boolean Functions," *Kibernetika,* 1970, No.3.

[105] R.L. Brooks, "On Colouring the Nodes of a Network," *Proc. Cambridge Philosophical Soc.*, 37., p. 194, 1941.

[106] J.R. Brown, "Chromatic Scheduling and the Chromatic Number Problem," *Management Science,* Vol. 19, pp. 456-463, 1972.

[107] F.M. Brown, "Weighted Realizations of Switching Functions," *IEEE Trans. on Comput.* Vol. 24, No. 11, pp. 1217-1221, 1975.

[108] F.M. Brown, "Equational Realizations of Switching Functions," *IEEE Trans. on Comput,*, Vol. 24, No. 11, pp. 1054-1066, 1975.

[109] F.M. Brown, "Constrained-Input Problem," *IEEE Trans. on Comput.*, Vol. 24, No. 1, pp. 102-106, 1975.

[110] F.M. Brown, "Boolean Resoning. The Logic of Boolean Equations," *Kluwer Academic Publishers,* 1990.

[111] R.E. Bryant, "Graph-Based Algorithms for Boolean Function Manipulation," *Trans. on Comput.*, Vol. C-35, No. 8, pp. 667-691, 1986.

[112] R. Bryant, "Binary Moment Diagrams," *Carnagie Mellon University*, CMU-CS-94-160 Report, May 31, 1994.

[113] E. Buehl, "Beitrag zur Theorie der Dynamischen Fehler-erscheinungen in Kombinatorischen und Sequentiellen Schaltnetzwerken," *Dissertation,* T.H. Karl-Marx-Stadt (Chemnitz), 1975, (in German).

[114] J.T. Butler, "On the Number of Functions Realized by Cascades and Disjunctive Networks," *IEEE Trans. on Comput.*, Vol. 24, No. 7, pp. 681-690, July 1975.

[115] J.T. Butler, "Fanout-Free networks of multi-valued gates," *Proc. 1976 ISMVL*, pp. 39-46, May 1976.

[116] J.T. Butler, "Tandem Networks of Universal Cells," *IEEE Trans. on Comput.*, Vol. 27, No. 9, pp. 785-799, 1978.

[117] J.T. Butler, "Analysis and Design of Fanout-Free Networks of Positive Symmetric Gates," *Journal of the Association for Computing Machinery,* Vol. 25, No. 3, pp. 481-498, 1978.

[118] J. Butler, K.J. Breeding, "Some Characteristics of Universal Cell Nets," *IEEE Trans. on Comput.*, Vol. C-22, pp. 897-903, October 1973.

[119] J. Butler and J. Schueller, "Worst Case Number of Terms in Symmetric Multiple-Valued Functions," *Proc ISMVL 1991.*

[120] 23. A.A. Butov, *Avtomatika i telemekhanika,* No. 9, pp. 121-125, 1978, (in Russian).

[121] A.A. Butov, *Avtomatizacjia Log. Proektirovanija Diskret. Ustroystv,* pp. 72-77, Minsk, 1980, (in Russian).

[122] A.A. Butov, *Kibernetika,* No.3, pp. 63-70, 1980, (in Russian).

[123] R.W. Butterworth, "A Set Theoretic Treatment of Coherent Systems," *SIAM JAMA,* Vol. 22, No. 4, pp. 590–598, 1972. QA1 .S73.

[124] P. Calazans, Ph.D. Thesis, 1994

[125] J. Carroll, *Final Technical Report*, Project GEMMA, Wright Lab/WPAFB, 8/9/94, 1994.

[126] P.A. Catlin, "On the Hajnal-Szemeredi theorem on disjoint cliques," *Util. Math.*, Vol. 17, pp. 163-178, May 1980.

[127] E. Cerny, D. Mange, E. Sanches, "Synthesis of Minimal Binary Decision Trees," *IEEE Trans. on Comput.*, Vol. 28, No. 7, pp. 472-482, July 1979.

[128] E.M. Clarke, K. McMillan, X. Zhao, M. Fujita, and J. Yang, "Spectral transforms for large boolean functions with applications to technology mapping," *Proc. 30th DAC*, June 1993.

[129] M. Chabinyc, "Pattern Based Machine Learning: A Comparison of Ada Function Decomposer Versions," *Final Report,* AFOSR School Apprentice Program, August 1990.

[130] K. Chakrabarti, O. Kolp, "Fan-in in Constrained Tree Networks of Flexible Cells," *IEEE Trans. on Comput.*, Vol. C-23, pp. 1238-1249, December 1974.

[131] A. H. Chan, "Using decision trees to derive the complement of a binary function with multiple-valued inputs," *IEEE Trans. on Comp.,* Vol. C-36, pp. 212 - 214, Febr. 1987.

[132] , S.C. Chang, M. Marek-Sadowska "Technology Mapping via Transformations of Function Graph," *Proc. Intern. Conf. on Computer Design,* pp. 159-162, 1992.

[133] M. Chein, H. Habib, M.C. Maurer, "Partitive Hypergraphs," *Discr. Math,* Vol.37, pp.35-50, 1981. QA1 .D52

[134] K.C. Chen et al, "Graph Based FPGA Technology Mapping for Delay Optimization," *IEEE Design and Test of Computers,* pp. 7-20, Sept. 1992.

[135] K-C. Chen and S. Muroga, "Input Assignment for Decoded PLA's with Multi-Input Decoders," *IEEE International Conference on Computer-Aided Design,* Digest of Technical Papers, pp. 474-477. YEAR*******.

[136] C.C. Cheung, "A Trimmed Tree Realization of Boolean Functions Using Universal Logic Modules," M.S. Thesis, *Dept Elec. and Comput. Eng. Univ Mass, Amherst,* 1973.

[137] C.C. Cheung, B.W. Ehrich, "Minimization of Tree-Type Universal Logic Circuits," *IEEE Trans. on Comput,*, Vol. 24, No. 11, pp. 1110-1113, 1975.

[138] N. Christofides, "An Algorithm for the Chromatic Number of a Graph," *Comp. J.,* Vol. 14, pp. 38-39, 1971.

[139] N. Christofides, "Graph Theory," *Academic Press*, New York, 1975.

[140] M. J. Ciesielski, S. Yang, and M. Perkowski, "Multiple-Valued Minimization Based on Graph Coloring," *Proc. ICCD'89,* pp. 262 - 265, October 1989.

[141] M. Ciesielski, S. Yang, "PLADE: A Two Stage PLA Decomposition," *IEEE Trans. on CAD,* Vol. 11, pp. 943-954, 1992.

[142] E.J. Cockayne, O. Favaron, C. Payan, and A.G. Thomason, "Contributions to the theory of domination, independence and irredundance in graphs," *Discrete Math.*, Vol. 33., No. 3., pp. 249–258, March 1981.

[143] T.F. Coleman, and J.J. More, "Estimation of Sparse Jacobian Matrices and Graph Coloring Problems," ANL-81-39, Argonne Nat'l. Lab., June 1981.

[144] Concurrent Logic Inc, "CLi6000 Series Field Programmable Gate Array," *Preliminary Information,* December 1991.

[145] J. Cong, A. Kahng, P. Trajmar, and Kuang-Chien Chen, "Graph Based FPGA Technology Mapping for Delay Optimization," *Proc. First Int'l ACM/SIGDA Workshop on Field Programmable Gate Arrays,* Berkeley, CA, February, 1992, pp. 77-82.

[146] S.C. Crist, "Synthesis of Combinational Logic Using Decomposition and Probability," *IEEE Trans. on Comput.,* Vol.C-29, No.11, pp.1013-1016, November 1980.

[147] L. Csanky, M. Perkowski, I. Schaefer, "Canonical Restricted Mixed-Polarity Exclusive Sums of Products," *Proc. of the IEEE ISCAS'92,* Intern. Symp. on Circ. and Systems, Sheratons on Harbor Island, San Diego, CA, May 10-13, 1992, pp. 17-20.

[148] L. Csanky, M. Perkowski, I. Schaefer, "Canonical Restricted Mixed-Polarity Exclusive-Or Sums of Products and the Efficient Algorithm for their Minimization," *IEE Proceedings, Pt.E,* Vol. 140, No. 1, January 1993.

[149] W.H. Cunnigham, "A Combinatorial Decomposition Theory," *Can. J. Math.,* Vol.32, pp.734-765, 1980. QA1 .C36

[150] H.A. Curtis, "Non-Disjunctive Decompositions," *Theory of Switching Bell Labs Report,* No. 19, Section II, 1, 49, 1958.

[151] H.A. Curtis, "A Functional Canonical Form," *J. ACM,* 6, pp. 245-258, 1959.

[152] H.A. Curtis, "A Generalized Tree Circuit," *J. ACM,* pp. 484-496, 1961.

[153] H.A. Curtis, "A New Approach to the Design of Switching Circuits," em Princeton, N.J., Van Nostrand, 1962.

[154] H.A. Curtis, "Generalized Tree Circuit - The Basic Building Block of an Extended Decomposition Theory," *J. Assn. Comput. Mach,* 1963, 10, pp. 562-581.

[155] H.A. Curtis, "Simplified Decomposition of Boolean Functions," *IEEE Trans. on Comput.*, Vol. 25, pp. 1033-1044, October 1976.

[156] S. Dao Trong, "Ein Entwurfsverfahren fuer Mehrstufige Schaltnetzwerke unter Gate-Array Randbedingungen," *Fortschritt-Berichte VDI,* Reihe 9, Nr. 46, VDI-Verlag, Duesseldorf 1984, (in German).

[157] J. Darringer et al, "Logic Synthesis Through Local Transformations," *IBM J. Res. Develop.,* pp. 272-280, July 1981.

[158] J.A. Darringer, D. Brand, W.H. Joyner, J.V. Gerbi, L. Trevillyan, "LSS: A System for Production Logic Synthesis," *IBM J. Res. Develop.* Vol. 28, July 1984.

[159] S.R. Das, C.L. Sheng, "On Detecting Total or Partial Symmetry of Switching Functions," *Trans. on Comput.* pp. 352-355, March 1971.

[160] S.R. Das, "On a new approach for finding all the modified cut-sets in an incompatibility graph," *IEEE Trans. Comput.*, Vol. C-22, pp. 187-193, February 1973.

[161] E.S. Davidson, "An Algorithm for NAND Decomposition Under Network Costraints," *IEEE Trans. on Comput.*, pp. 1098-1109, Dec. 1969.

[162] M. Davio, J. P. Deschamps, and A. Thayse, "Discrete and Switching Functions," *Mc.Graw-Hill,* 1978.

[163] D. De Caen, P. Erdos, N. Pullman, and N. Wormeld, "Extremal Clique Coverings of Complementary Graphs," *Combinatorica,* Vol. 6., No. 4., pp. 309–314, 1986.

[164] D. DeMicheli, M. Santomauro, "Topological Partitioning of Programmable Logic Arrays," *Proc. Intern. Conference on Computer Aided Design,* ICCAD'83, Santa Clara 1983.

[165] G. DeMicheli, R.K. Brayton and A. Sangiovanni–Vincentelli, "Optimal State Assignment for Finite State Machines," *IEEE Transactions on Computer–Aided Design,* Vol. CAD-4, No. 3, pp. 268–284. July 1985.

[166] G. DeMicheli, "Symbolic Design of Combinational and Sequential Logic Circuits Implemented by Two-Level Logic Macros," *IEEE Transactions on Computer–Aided Design,* Vol. CAD-5, No. 4, pp. 597–616, Oct. 1986.

[167] G. de Micheli, D. Ku, F. Mailhot, T. Truong, "The OLYMPUS Synthesis System," *IEEE Trans. on Design and Test of Computers,* Oct. 1990.

[168] S.C. de Sarkar, A.K. Basu, and A.K. Choudhury, "Simplification of Incompletely Specified Flow Tables With the Help of Prime Closed Sets," *IEEE Trans. Computers,* Vol. C-18, pp. 953–955, 1969.

[169] S.C. de Sarkar, A.K. Basu, and A.K. Choudhury, "On the Determination of Irredundant Prime Closed Sets," *IEEE Trans. Computers*, Vol C-20, pp. 933–938, 1971.

[170] P. Deschizeaux , "Synthese de Fonction Booleennes Generales," Dr.Ing. Thesis, *Inst. Math. Appl. Grenoble,* France, 1967.

[171] S. Devadas, A.R. Wang, A.R. Newton, A. Sangiovanni-Vincentelli, "Boolean Decomposition in Multi-Level Logic Optimization," *Proc. IEEE International Conf. on Computer-Aided Design,* pp. 290-293, 1988.

[172] S. Devadas, A.R. Wang, A.R. Newton, and A. Sangiovanni-Vincentelli, "Boolean Decomposition of Programmable Logic Arrays," *Proc. IEEE Custom Integrated Circuit Conference,* pp. 2.5.1-2.5.5, 1988.

[173] S. Devadas, A.R. Wang, A.R. Newton, and A. Sangiovanni-Vincentelli, "Boolean Decomposition in Multi-Level Logic Optimization," *IEEE International Conference on Computer-Aided Design,* Digest of Technical Papers, 1988, pp. 290-293.

[174] S. Devadas, A. Wang, A.R. Newton, and A. Sangiovanni-Vincentelli, "Boolean Decomposition in Multilevel Logic Optimization," *IEEE Journal of Solid-State Circuits,* Vol. 24, pp. 399-408, April 1989.

[175] D.L. Dietmeyer, P.R. Schneider, "Identification of Symmetry, Redundancy and Equivalence of Boolean Functions," *IEEE Trans. Electron. Comput.*, Vol. 16, pp. 804-817, December 1967.

[176] D.L. Dietmeyer, "Logic Design of Digital Systems," *Boston: Allyn and Bacon,* 1971.

[177] S. Dormino, M.A. Canto, "Generalization of the Voith Method for Minimization of Incompletely Specified Function With Universal Logic Modules," *Intern. J. Electr.*, Vol. 54, No. 2, pp. 271, 1983.

[178] D. Drake, "Embedding maximal cliques of sets in maximal cliques of bigger sets," *Discrete Math.,* Vol. 58., No. 3., pp. 229-242, March 1986.

[179] R. Drechsler, A. Sarabi, M. Theobald, B. Becker, M.A. Perkowski, "Efficient Representation and Manipulation of Switching Functions Based on Ordered Kronecker Functional Decision Diagrams," *Proceedings of DAC '94*, San Diego, CA, June 1994.

[180] F. Dresig, "Synthese Kombinatorischer Schaltnetzwerke mit Arbeitsplatzcomputern," *Dissertation*, T.U. Karl-Marx-Stadt (Chemnitz), 1988, (in German).

[181] F. Dresig, C. Zimmermann, "ALOE - Logiksynthese mittels XBOOLE," *27. Arbeitstagung des FA Schaltkreissysteme und CAD-Komponenten der WS Computer- und Mikroprozessortechnik der KdT,* Eisenach, 1989.

[182] F. Dresig, U. Landmann, B. Meyer, and C. Zimmermann, "Von DDL zu NBS - Beitraege zum Logikentwurf mittels XBOOLE," *Tagungsmaterialien der 4. Tagung Schaltkreisentwurf,* Dresden 1990, (in German).

[183] F. Dresig, O. Rettig, and U.G. Baitinger, "Logic Synthesis for Universal Logic Cells," *Intern. Workshop on Field Programmable Logic and Applications,* Oxford, in: Moore, W.R., Luk, W.: FPGAs. Abington EE and CS Books, Abington, England, 1991.

[184] F. Dresig, Ph. Lanches, O. Rettig, U.G. Baitinger, "Functional Decomposition for Universal Logic Cells Using Substitution," *Proc. of the European Conference on Design Automation (EDAC),* Brussels, 1992.

[185] T. Duempfert, "Synthese von Schwellwertfunktionen," *Diplomarheit,* TU Chemnitz, 1992, (in German).

[186] J.R. Duley, D. L. Dietmeyer, "A Digital System Design Language (DDL)," *IEEE Trans. on Computers,* 17 (1968), pp. 850-861.

[187] F.D.J. Dunstan, "Sequential Colorings of Graphs," *Cong. Num.,* Vol. 15, pp. 151-158, 1976.

[188] A. Dunworth, and H.V. Hartog, "An Efficient State Minimization Algorithm for Some Special Classes of Incompletely Specified Sequential Machines," *IEEE Trans. Comp.*, Vol. C-28, No. 7, pp. 531–535, July 1979.

[189] Y. Durand, "Logic Synthesis Based on Unification and Rewriting Rules," in: *Design Methodologies for VLSI and Computer Architecture,* D.A. Edwards (Editor), Elsevier Science Publishers B.V. (North-Holland), 1989.

[190] V.F. D'yachenko, "On an Algorithm for Determining the Maximum of a Boolean Function," *Problemy Peredachi Informatsii,* No. 1, 1966.

[191] C.R. Edwards, "The Application of the Rademacher-Walsh Transform to Boolean Function Classification and Threshold Logic Synthesis," *IEEE Trans. Comput.*, Vol. C-24, pp. 48-62, 1975.

[192] C.R. Edwards, S.L. Hurst, "Digital Synthesis Procedure Under Function Symmetries and Mapping Methods," *IEEE Trans. on Comput.*, Vol. 27, No. 11, pp. 985-997, 1978.

[193] C.S. Edwards, and C..H. Elphick, "Lower bounds for the clique and chromatic number of a graph," *Discrete Appl. Math.,* Vol. 5., No. 1., pp. 541-64, January 1983.

[194] O. Ehrsam, "EKSIS - ein Programm zum Entwurf Kombinatorischer Schaltungen durch Iterative Ausnutzung von Symmetrieeigenschaften," *Nachrichtentechnik Elektronik* 12, pp. 460-462, 1984, (in German).

[195] A.B. Ektare, D.P. Mital, "An Algorithm for Designing Multiplexer Circuits," *Intern. J. of Electronics,* Vol. 49, No. 2, 1980.

[196] M. Elmasry, "Logic Partition for Multi-Emitter 2-Level Structures," *IEEE Trans. on Circuits and Systems,* Vol. 21, No. 3, pp. 354-359, 1974.

[197] G. Epstein, "Synthesis of Electronic Circuits for Symmetric Functions," *IRE Trans. Elec. Comp.* Vol. EC-7, March 1958, pp. 57-60.

[198] G. Epstein, D.M. Miller, J.C. Muzio, "Some Preliminary Views on the General Synthesis of Electronic Circuits for Symmetric and Partially Symmetric Functions," *Proc. 7th Intern. Symp. on Multiple-Valued Logic*, pp. 29-34, May 1977.

[199] G. Epstein, D.M. Miller, J.C. Muzio, "Selecting Don't Care Sets for Symmetric n-Valued Functions: a Pictorial Approach Using Matrices," *Proc. 10th Intern. Symp on Multiple-Valued Logic,* May 1980, pp. 219-225.

[200] P. Erdos, A.M. Hobbs, and C. Payan, "Disjoint cliques and disjoint maximal independent sets of vertices in graphs," *Discrete Math.*, Vol. 42., No. 1., pp. 57–61, Nov. 1982.

[201] A.P. Ershov, "Vvedenije v Teoreticeskoye Programmirovanije," Moscow, 1977 (in Russian).

[202] S. Even, "Algorithmic Combinatorics", Macmillan, New York, 1973.

[203] I.L. Fadeev, *Tr. Sib fiz-tehn. in-ta*, 1966, Collection 48, pp. 133-137. (in Russian).

[204] I.L. Fadeev, "Logiceskij Jazyk Dlja Predstavlenija Algoritmov Sintieza Relejnych Ustroistv," pp. 184-193, Moscow, 1966, (in Russian).

[205] I.L. Fadeev, "Avtomatizacjia Logiceskogo Proektirovanija Cifrovych Ustrojstv," Kiev, 1974, pp. 48-61 (in Russian).

[206] I.L. Fadeev, *Algoritmy resenija zadac diskret. matematiki,* pp. 139-149, Tomsk 1979, (in Russian).

[207] B.J. Falkowski, and M. Perkowski, "Algorithm for the Generation of Disjoint Cubes for Completely and Incompletely Specified Boolean Functions," *Intern. J. of Electronics,* Vol. 70, No. 3, pp. 533 - 538, March 1991.

[208] B. Falkowski, I. Schaefer, M. Perkowski, "Computer Algorithms for the Calculation of Rademacher-Walsh Spectrum for Completely and Incompletely Specified Boolean Functions", *Proc. of the IEEE EURO-DAC '92, European Design Automation Conference,* Sept. 7-10, Hamburg, 1992.

[209] B. Falkowski, I. Schaefer, M. Perkowski, "Effective Computer Methods for the Calculation of Rademacher-Walsh Spectrum for Completely and Incompletely Specified Boolean Functions", *IEEE Trans. on Computer-Aided Design,* October 1992, pp. 1207 - 1226.

[210] B. Falkowski, I. Schaefer, and Ch-H. Chang, "An Effective Computer Algorithm for the Calculation of Disjoint Cube Representation of Boolean Functions", *Proc. of the 36-th Midwest Symposium on Circuits and Systems,* pp. 1308-1311, 1993.

[211] K.Y. Fang, A.S. Wojcik, "Modular Decomposition of Combinational Multiple-Valued Circuits," *IEEE Trans. on Comput.*, Vol. 37, No. 10, pp. 1293-1301, 1988.

[212] S. Fastowicz, "Independence, Clique size and maximum degree," *Combinatorica,* Vol. 4., No. 1., pp. 35–38, 1984.

[213] E. Fehlauer, D. Garte, C. Mehlhorn, S. Ruelke, "AZUR - Ein Werkzeug zur Synthese Mehrstufiger Logik," *E.I.S. Workshop,* Dresden, April 1991, (in German).

[214] B. Fei, Q. Hong, H. Wu, M.A. Perkowski, and N. Zhuang, "Efficient Computation for Ternary Reed-Muller Expansions under Fixed-Polarities", *Int. J. Electronics,* Vol. 75, No. 4, pp. 685 - 688, 1993.

[215] C. Files, "Using a Search Heuristic in an NP-Complete Problem in Ashebhurst-Curtis Decomposition," Final Report for Graduate Summer Research Program, Wright Laboratory, Sponsored by Air Force Office of Scientific Research, Bolling Air Force Base, DC and Wright Laboratory, August 1994.

[216] C. Files, "A New Approach to Paritition Selection in Ashenhurst-Curtis Functional Decomposition," *M.S. Thesis, College of Graduate Studies,* University of Idaho, August 1995.

[217] D. Filo, J. Yang, F. Mailhot, G. de Micheli, "Technology Mapping for a Two-Output RAM-Based Field Programmable Gate Array," *Proc. of the European Conf. on Design Automation (EDAC)*, Amsterdam, pp. 534-538, 1991.

[218] M. Findler, "Neural Networks and Machine Learning," *Graduate Student Research Program,* Final Report, AFOSR, August 1989.

[219] H. Fleisher, and L.I. Maissel, "An Introduction to Array Logic," *IBM J. R&D*, Vol. 19, pp. 98-109, March 1975.

[220] J.M. Francioni, "Imprecise Decision Tables and Their Optimization," *Master Thesis,* Department of Mathematics and Computer Science, Florida State University, Tallahassee, Florida, June 1979.

[221] J.M. Francioni, A. Kandel, "Decomposable Fuzzy-valued Switching Functions," *Fuzzy Sets and Systems,* Vol. 9, No. 1, pp. 41-68, 1983.

[222] R. J. Francis, J. Rose, K. Chung, "Chortle: A Technology Mapping Program for Lookup Table-Based Field Programmable Gate Array," *Proc. 27th ACM/IEEE Design Automation Conf.*, 1990, pp. 613-619.

[223] R. Francis, J. Rose, Z. Vranesic, "Chortle-crf: Fast Technology Mapping Look-up Table-Based FPGAs," *Proc. 28-th ACM/IEEE Design Automation Conf.,* pp. 227-233, 1991.

[224] R. Francis, J. Rose, Z. Vranesic, "Technology Mapping of Look-up Table-Based FPGAs for Performance", *Proc. IEEE Intern. Conf. on CAD,* pp. 568-571, 1991.

[225] R.J. Francis, J. Rose, Z. Vranesic, "Technology Mapping for Delay Optimization of Lookup Table-Based FPGAs," *MCNC Logic Synthesis Workshop,* 1991.

[226] J.F. Frenzel, "Application of Genetic Algorithms to Pattern Theory," *Final Report,* Summer Faculty Research Program, July 1993.

[227] J. Fricke, "Decomposition of Multiple-Valued Logic Functions," *Proc. 8th ISMVL,* pp. 208-212, 1978.

[228] P. Fricnovics, "Coding of the Internal States of Asynchronous Finite Automata by a p-code of Minimal Length," *Theory of Finite Automata and Its Applications*, Vol. 1, pp. 22-34, pp. 64–65, 1973.

[229] A.D. Friedman, P.R. Menon, "Theory and Design of Switching Circuits." 1975.

[230] M. Fujita, Y. Matsunaga, T. Kakuda, "On Variable Ordering of Binary Decision Diagrams for the Application of Multi-Level Logic Synthesis," *Proc. of 2nd EDAC,* 1990.

[231] M. Fujita, Y. Matsunga, and T. Kakuda, "On Variable Ordering of Binary Decision Diagrams for the Application of Multi-Level Synthesis," *European Conf. on Design Automation*, pp. 50-54, 1991.

[232] M. Fujita, Y. Kukimoto, and R. Brayton, "BDD Minimization by Truth Table Permutations," *Proc. IWLS '95.*

[233] T. Funke, "Vergleich der Ergebnisse unterschiedlicher Synthesewerkzeuge," *Groeer Beleg*, T.U. Chemnitz, Juni 1991, (in German).

[234] Z. Furedi, "The number of maximal independent sets in connected graphs," *J. Graph Theory,* Vol. 11, No. 4, pp. 463–470, 1980.

[235] F. Furtek, G. Stone, I. Jones, "Labyrinth: A Homogeneous Computational Medium," *IEEE Custom Integrated Circuit Conference,* Boston, 1990.

[236] Galois

[237] M.R. Garey, and D.S. Johnson, "The Complexity of Near Optimal Graph Coloring," *JACM,* Vol. 23, pp. 43-49, 1976.

[238] M.R. Garey, D.S. Johnson, and H.C. So, "An Application of Graph Coloring to Printed Circuit Testing," *IEEE Trans. on Circuits and Systems,* Vol. 23, 1976.

[239] M. Garey and D. Johnson, "Computers and Intractability: A Guide to the Theory of NP-Completeness," *W. H. Freeman and Co. Publ.*, San Francisco, 1979.

[240] F. Gavril, "Algorithms for maximum k-coloring and k-covering of transitive graphs," *Networks*, Vol. 17, No. 4, pp. 465–470, Winter 1987.

[241] M.A. Gavrilov, "Teorija Avtomatov," Moscow, 1976, pp. 34-71 , (in Russian).

[242] M.A. Gavrilov, "Teorija Relejnych Ustrojstv i Konecnych Avtomatov: Izbrannyje Trudy," Moscow, 1983, (in Russian).

[243] T.K. Gearhart, "Investigations of a Lower Bound on the Error in Learned Functions," *Final Report,* USAF-UES Summer Faculty Research Program, July 1990.

[244] L. Geehaeds, and W. Lindenberg, "Clique detection for nondirected graphs: two new algorithms," *Computing,* Vol. 21, No. 4., pp. 295–322, 1979.

[245] A.J. de Geus, "Logic Synthesis and Optimization Benchmarks for the 1986 Design Automation Conference," *Proc. of the 23rd DAC,* 1986.

[246] A.J. de Geus, W. Cohen, "A Rule-Based System for Optimizing Combinational Logic," *IEEE Design and Test of Computers,* Vol. 2, No. 4, August 1986.

[247] S. Ghosh, A.K. Choudhury, "Partition of Boolean Functions for Realization with Multi-threshold Threshold Logic Elements," *IEEE Trans. Comput.*, Vol. C-22, pp. 204-215, 1973.

[248] A. Gibbs, "Algorithmic Graph Theory," *Cambridge University Press*, 1985.

[249] K.G. Gilmanov, "Synthesis of Logical Circuits on the Basis of Universal Formal Neurons with a Small Number of Inputs", *Engineering Cybernetics*, Vol.9, No.1, pp.93-97, 1971, (in English). TJ212 .A413.

[250] Gimpel, J.F., "The Minimization of TANT Networks," IEEE TEC, Vol. EC-16, pp. 18-38, February 1967.

[251] S. Ginsburg, "A Synthesis Technique for Minimal State Sequential Machines," *IRE Trans. Electron. Computers,* Vol. EC-8, No. 1, pp. 13–24, March, 1959.

[252] S. Ginsburg, "On the Reduction of Superfluous States in a Sequential Machine," *J. Assoc. Computing Machinery,* Vol. 6, pp. 259–282, April, 1959.

[253] J.A. Goldman, "Pattern Theoretic Knowledge Discovery", *Proc. 6th IEEE International Conference on Tools with Artificial Intelligence,* IEEE, November 1994.

[254] J. A. Goldman, "Machine Learning: A Comparative Study of Pattern Theory and C4.5," Wright Laboratory, USAF, Technical Report, WL-TR-94-1102, WL/AART, WPAFB, OH 45433-6543, August 1994.

[255] J. A. Goldman and M. L. Axtell, "On Using Logic Synthesis for Supervised Classification Learning," *7th IEEE International Conference on Tools with Artificial Intelligence,* IEEE, November 1995.

[256] J.A. Goldman, T.D. Ross, and D.A. Gadd, "Pattern Theoretic Learning", *AAAI Spring Symposium Series on Systematic Methods of Scientific Discovery,* AAAI, March 1995.

[257] S.W. Golomb, "On the Classification of Boolean Functions," *IRE Trans. Circ. Theory,* Spectral suppplement, CT-6, pp. 176-186, 1959.

[258] =

[259] M. Gondran, M. Minoux, "Graphs and Algorithms," *John Wiley & Sons,* New York, 1984.

[260] L.M. Goodrich, "A Map Technique for Identifying Variables of Symmetry", *Bell Syst. T,* Vol.53, No.5, pp.801-826, 1974. TK1 .B425

[261] Y.B. Gershkov, *Autom. Remot.* R. 753, 1967, (in Russian), TJ212 .A9135.

[262] A. Grasselli, and F. Lucio, "A Method of Minimizing the Number of Internal States in Incompletely Specified Sequential Networks," *IEEE Trans. Electr. Comp.,* Vol. EC-14, No. 3, pp. 330–359, June 1965.

[263] C.R. Greer, R.A. Thompson, "Combinational Logic With Decoder," *IEEE Trans. on Comput.*, Vol. C-27, No. 9, 1978.

[264] G.R. Grimmett, and C.J.H. McDiarmid, "On Colouring Random Graphs," *Math. Proc. Comb. Phil. Soc.* Vol. 77, pp. 314–324, 1975.

[265] W. Grzymala-Busse, Z. Pawlak, "On Some Subset of the Partition Set," *Fundamenta Informaticae,* Vol. VII, No. 4, 1984. PWN, Warszawa, Poland.

[266] J.W. Grzymala-Busse, "On the Reduction of Instance Space in Learning from Examples," In *"Methodologies for Intelligent Systems,"* eds. Z.W. Ras, M. Zemankova, M.L. Emrich, Elsevier Sci. Publ., pp. 388-395, 1990.

[267] J.W. Grzymala-Busse, "LERS - A System to Learning from Examples Based on Rough Sets," In *"Intelligent Decision Support. Handbook of Application and Advances of the Rough Sets Theory",* ed. R. Slowinski, Kluwer Academic Publishers, Dordrecht 1992.

[268] S.C. Gupta, "Decompositions of 4-Variable Boolean Functions," *Computers and Electrical Engineering,* Vol. 8, No. 1, pp. 41-48, 1981.

[269] M. Habib, M.C. Maurer, "1-Intersecting Families", *Discr. Math.* Vol.53, pp.91-101, 1985. QA1 .D52

[270] F. Haentemann, "Anwendungsspezifischer Entwurf von DOMINO-CMOS-Schaltnetzen im Gate-Matrix-Entwurfsstil," *Fortschritt-Berichte VDI,* Reihe 9, Nr. 107, VDI-Verlag, Duesseldorf 1990, (in German).

[271] C. Halatsis, N. Gaitanis, "Irredundant Normal Forms and Minimal Dependence Sets of Boolean Function," *IEEE Trans. on Comput.*, Vol. C-27, Nov. 1978.

[272] P.L. Hammer, and S. Rudeanu, "Boolean Methods in Operations Research," *Springer-Verlag*, New York 1968.

[273] F. Harrary, "Graph Theory," *Addison Wesley*, Reading, Mass., 1969.

[274] F. Harary, J.S. Maylee, "Graph and Applications," *Proc. First Colorado Symp. on Graph Theory,* John Wiley & Sons, New York, 1985.

[275] J. Hartmanis , R.E. Stearns, "Algebraic Structure Theory of Sequential Machines," Englewood Cliffs, N.Y., *Prentice Hall, Inc.,* 1966.

[276] K. Hauptmann, W. Schuefer, "Rapid Prototyping mit Programmierbaren Gate Arrays," *ITG-Fachbericht der ITG Fachtagung* vom 3.-5.10.1989, Stuttgart, VDE-Verlag GmhH, Berlin-Offenbach, in German.

[277] J.P. Hayes, "The Fanout Structure of Switching Functions," *JACM,* Vol. 22, No.4, pp. 551-571, Oct. 1975.

[278] J.P. Hayes, "Enumeration of Fanout-Free Boolean Functions," *JACM*, Vol. 23. pp. 700-709, Oct. 1976.

[279] B. Hedman, "The maximum number of cliques in dense graphs," *Discrete Math,* Vol. 54., No. 2., pp. 161–166, April 1985.

[280] B. Hedman, "Clique numbers of graphs," *Discrete Appl. Math.,* Vol. 15. No. 1, pp. 55–60, Sept. 1986.

[281] S.L. Height, "Complex Disjunctive Decomposition of Incompletely Specified Boolean Functions," *Ph.D. dissertation,* Univ. New Mexico, Albuquerque, July 1970.

[282] S.L. Height, "Minimal Input Solutions," *IEEE Trans. on Comput.*, pp. 923-925, August 1971.

[283] S.L. Height, "Complex Disjunctive Decomposition of Incompletely Specified Boolean Functions," *IEEE Trans. on Comput.*, Vol. C-22, No. 1, January 1973, pp.. 103-110.

[284] M. Helliwell, M. A. Perkowski, "A Fast Algorithm to Minimize Multi-Output Mixed-Polarity Generalized Reed-Muller Forms," *Proc. of the IEEE/ACM 25-th Design Automation Conference,* pp. 427 - 432, Anaheim, CA, June 12-15, 1988.

[285] Shusheng He and M. Torkelson, "Disjoint Decomposition With Partial Vertex Chart," In *Proc. of the Intern. Workshop on Logic Synthesis,* Lake Tahoe, CA, pp. P2a-1-P2a-5, May 1993.

[286] B. Heeb, "Logic Minimization for Novel Programmable Logic Architectures," *International Workshop on Field Programmable Logic and Applications,* Oxford, in: Moore, W.R., Luk, W.: FPGAs. Abington EE and CS Books, Abington, England, 1991.

[287] Ph. Ho, and M. A. Perkowski, "Minimization of Fine-Grain FPGAs Using Free Kronecker Decision Diagrams," *Report, Department of Electrical Engineering*, PSU, 1994.

[288] C. Hoede, "Semi-matchings and semi-stars, surroundings, decompositions and detachments. An analysis of the maximum clique problem," Memo 547, Dept. Applied Math., Twente Univ. of Techn., preprint, 1985.

[289] C. Hoede, "Hard Graphs for the Maximum Clique Problem," *Discrete Mathematics*, Vol. 72., pp. 175–179, 1988.

[290] S.J. Hong, R.G. Cain, and D.L. Ostapko, " MINI: A heuristic approach for logic minimization," *IBM J. Res. Develop.,* Vol. 18, pp. 443 – 458, Sept. 1974.

[291] G. Hopkins, and W. Staton, "Graphs with unique maximum independent sets," *Discrete Math.*, Vol. 57., No. 3, pp. 245–251. *Discrete Math.*, December 1985.

[292] E. Howorka, "On metric properties of certain clique graphs," *J. Comb. Theory* Series, Vol. B 27, No.1., pp. 67–74, Aug. 1979.

[293] S. J. Hong, "R-MINI: A Heuristic Algorithm for Generating Minimal Rules from Examples", *Pacific Rim International Conference on Artificial Intelligence,* PRICAI, 1994.

[294] B. Hruz, "Unateness Test of a Boolean Function and Two General Synthesis Methods Using Threshold Logic Elements," *IEEE Trans. on Comput.,* pp. 122-131, Febr. 1969.

[295] W.L. Hsu, and G.L. Nemhauser, "Algorithms for minimum covering by cliques and maximum clique in claw-free perfect graphs," *Discrete Math.*, Vol. 37., No. 2,3, pp. 181–191, December 1981.

[296] J.D. Huang, J.Y. Jou, and W.Z. Shen, "Compatible Class Encoding in Roth-Karp Decomposition for Two-Output LUT Architecture," *Proc 1995 ICCAD*, pp. 359–363, 1995.

[297] D.A. Huffmann, "The design and use of hazard-free switching networks," *Journal Ass. Comp. Mach.* Vol. 4, No. 1, 1957.

[298] R.B. Hurley, "Decision Tables in Software Engineering," *Van Nostrand Reinhold,* New York, 1983.

[299] S.L. Hurst, C.R. Edwards, "Preliminary Considerations of the Design of Combinatorial and Sequential Digital Systems Under Symmetry Methods," *Int. J. Electron.*, 1976, 40, pp. 499-507.

[300] S.L. Hurst, "Detection of Symmetries in Combinatorial Functions by Spectral Means," *IEE J. Electron. Circ. and Syst.*, Vol. 1, No. 5, pp. 173-180, 1977.

[301] S.L. Hurst, "Logical Processing of Digital Signals," *Crane-Russak,* New York, Edward Arnold, London, 1978.

[302] S.L. Hurst, D.M. Miller, J.C. Muzio, "Spectral Techniques in Digital Logic," *Academic Press,* London, 1985.

[303] T.T. Hwang et al, "Multi-Level Logic Synthesis Using Communication Complexity," *Proc. 26th DAC,* pp. 215-220, 1989.

[304] T.T. Hwang, R.M. Owens, M.J. Irwin, "Exploiting Communication Complexity for Multilevel Logic Synthesis," *IEEE Trans. on CAD,* Vol. 9, No. 10, pp. 1017-1027, Oct, 1990.

[305] T.T. Hwang, et al., "Efficiently Computing Communication Complexity for Multilevel Logic Synthesis," *IEEE Trans. on CAD,* Vol. 11., No 5, pp. 545-554, May 1992.

[306] Y. Ikuea, "Algorithms for vertex packing and clique covering problems on some unimodular graphs," Cornell Univ., Ithaca, N.Y., 1981, 158 pp.

[307] H. Imai, and T.S. Asano, "Finding the connected components and a maximum clique of an intersection graph of rectangles in the plane," *Algorithms,* Vol. 1, No. 4., pp. 310–323, Dec. 1983.

[308] K. Jasinski, T. Luba, J. Kalinowski, "CAD Tools for PLD Implementation of ASICs", *Proc. of Second Eurochip Workshop on VLSI Design Training,* pp. 225-230, Grenoble, 1991.

[309] G.H. John, R. Kohavi, and K. Pfleger, "Irrelevant Features and the Subset Selection Problem," In *Machine Learning: Proceedings of Eleventh International Conference,* 1994, July 1994.

[310] D.S. Johnson, "Worst Case Behavior of Graph Coloring Algorithms," *Proc. 5th S-E Conf. on Combinatorics, Graph Theory and Comput.,* Utilitas Mathematica, Winnipeg 1979, pp. 513–527.

[311] J.M. Joller, "Constructing Fault-Trees by Stepwise Refinement," *IEEE Trans. on Reliability,* Vol. 31, No. 4, pp. 333-338, 1982.

[312] L. Jozwiak, F. Volf, "An Efficient Method for Decomposition of Multiple-Output Boolean Functions and Assigned Sequential Machines", *Proc. European Conference on Design Automation,* pp. 114-122, 1992.

[313] L. Jozwiak, "General Decomposition and Its Use in Digital Circuit Synthesis," *Manuscript,* Eindhoven University of Technology, Faculty of Electrical Engineering, P.O. Box 513, 5600 MB Eindhoven, The Netherlands, 1994.

[314] A.Kandel, "Decomposition of Fuzzy Functions," *IEEE Trans. on Comput.*, Vol. 25, No. 11, pp. 1124-1130, 1976.

[315] A. Kandel, "Fuzzy Switching and Automata: Theory and Applications", *Crane-Russak,*, New York, 1979.

[316] M. Karnaugh, "The map method for synthesis of combinational logic circuits", *Trans. AIEEE 72,* pt. I, pp. 593-598, 1953.

[317] R.M. Karp, F.E. McFarlin, J.P. Roth, J.R. Wilts, "A Computer Program for the Synthesis of Combinational Switching Circuits," *In Proc. AIEE Annual Symposium on Switching Circuits Theory,* pp. 182-194, 1961.

[318] R.M. Karp, "Functional Decomposition and Switching Circuit Design," *J. Soc. Industr. Appl. Math.*, Vol. 11, No. 2, pp. 291-335, June 1963.

[319] R.M. Karp, "Sintez Relejnych Struktur," pp. 220-232, Moscow 1965, (in Russian),

[320] R.M. Karp, "Reducibility among Combinatorial Problems," Complexity of Computer Computations, (R.E. Miller and J.W. Thatcher eds), Plenum Press, New York (1972), pp. 85-104.

[321] K. Karp, "Using if-then-else DAGS to do Technology Mapping for Field-Progammable Gate Arrays," *Internal Report, No.: UCSC-CRL-90-43,* Univ. of California, Santa Cruz, Sept. 1990.

[322] K. Karplus, "XMAP: A Technology Mapper for Table-Lookup Field Programmable Gate Arrays," *Proc. 28th ACM/IEEE Design Automation Conf.,* pp.240-243, 1991.

[323] M.G. Karpovsky, "Finite Orthogonal Series in the Design of Digital Devices," *J. Wiley,* New York, 1976.

[324] M.G. Karpovsky, "Harmonic Analysis over Finite Commutative Groups in Linearization Problems for Systems of Logical Functions," *Inform. Control,* Vol. 33. No.2, pp. 142-165, Febr. 1977.

[325] V. Ye. Katsman, "Theory of Partition Models," *Soviet Journal of Computer and Systems Sciences,* (Tekhnicheskaya Kibernetika), Vol. 29, No. 1, pp. 64-73, Jan-Febr 1991.

[326] U. Kebschull, E. Schubert, and W. Rosenstiel, "Multilevel Logic Synthesis Based on Functional Decision Diagrams," *Proc. IEEE Euro-DAC*, pp. 43-47, 1992.

[327] J. Kella, "State Minimization of Incompletely Specified Sequential Machines," *IEEE Trans. Computers,* Vol. C-19, pp. 342–348, 1970.

[328] M.B. Kermani, M.H. Mickle, L.P. McNamee, "Identification of Disjunctively Decomposable Logic Functions Employing a Karnaugh Map," *IEEE Trans. on Comput.*, pp. 277-279, Vol. 18, March 1969.

[329] Kerntopf, P., Michalski, A., "Selected Problems in Synthesis of Combinatorial Logic Circuits," PWN, Warsaw, 1972 (in polish).

[330] B. Kick, "Logiksynthese - eine Uebersicht," in: *Simulation und Synthese logischer Schaltungen,* (Hrsg: H. Khakzar), Expert-Verlag, 1991, (in German).

[331] B.G. Kim, "Multilevel Logic Synthesis Using Extended Array Implementation," *Ph.D. Dissertation,* Univ. Wisconsin-Madison, 1989.

[332] B.G. Kim, D.L. Dietmeyer, "Multilevel Logic Synthesis of Symmetric Switching Functions," *IEEE Trans. on CAD,* Vol. 10, No. 4, pp. 436-446, April 1991.

[333] E. Kjelkrud, "A Computer Program for the Synthesis of Switching Circuits by Decomposition," WHERE?? June 1972.

[334] E. Kjelkrud, "On the Decomposition of General Switching Functions," *The Royal Institute of Technology, Division of Applied Electronics,* Stockholm, Sweden, 1971.

[335] U. Knoth, C. Zimmermann, "Funktionelle Verifikation synchroner, digitaler Systeme," *Dissertation,* TU Chemnitz, 1990, (in German).

[336] Z. Kohavi, "Switching Circuits and Finite Automata Theory," *McGraw Hill,* 1978.

[337] R. Kohavi, and B. Frasca, "Useful Feature Subsets and Rough Set Reducts", *Third International Workshop on Rough Sets and Soft Computing,* 1994.

[338] R. Kohavi, "Bottom-up Induction of Oblivious Read-Once Decision Diagrams," In *European Conference on Machine Learning,* 1994.

[339] K.L. Kodandapani, S.C. Seth, "Combinational Networks With Restricted Fan-Out," *IEEE Trans. on Comput.*, Vol. C-27, No. 4, pp. 309-318, 1978.

[340] M. Koegst, Ch. Posthoff, B. Steinbach, "Beraecksichtigung von kombinatorischen Bluecken beim Logischen Entwurf Digitaler Systeme," *Nachrichtentechn. Elektronik,* Berlin 34, 2, 1984, (in German).

[341] L.T. Kou, L.S. Stockmeyer, and C.K. Wong, "Covering edges by cliques with regard to keyword conflicts and intersection graphs," *CACM,* Vol. 21, No. 2., pp. 135–139, Febr. 1978.

[342] A. Kowalczyk, "Decomposition of Strongly Unspecified Boolean Functions," *Przeglad Telekomunikacyjny,* No. 1, 1983, (in Polish).

[343] J. Koza, *"Genetic Programming,"* MIT Press, 1992.

[344] H. Kreutz, "Entwurf von Schaltnetzen aus Zuordnerbausteinen durch Dekomposition von Baende In unvollstaendiger Schaltfunktionen," *Dissertation,* Universitaet Karlsruhe 1980, (in German).

[345] Y.S. Kuo, and C. Chau, " Generating essential primes for a boolean function with multiple-valued inputs," *Proceedings of DAC' 86,* 1986.

[346] Y.S. Kuo, "Generating Essential Prime Implicants for a Boolean Function with Multiple–valued Inputs", *IEEE Trans. on Computers,* Vol.36, pp. 356-359, Mar. 1987.

[347] A.V. Kuznetsov, "On Nonrepeating Contact Networks and Nonrepeating Superpositions of Boolean Functions," *Trans. Steklov Math. Inst. AN SSSR*, Izd. AN SSSR, Vol. 51, (in Russian), 1958.

[348] A.V. Kuznetsov, "A Property of Functions Realized by Nonplanar Nonrepeating Circuits," *Trans. Steklov. Math. Inst. AN SSSR,* Izd. AN SSSR, Vol 51, (in Russian).

[349] H.C. Lai, "Design of diagnosable MOS - Networks," *Dissertation,* Dep. of Computer Science, University of Illinois, December 1979.

[350] Y.T. Lai, M. Pedram, S. Vrudhula, "BDD-based Logic Decomposition: Theory." *Technical Report,* Dept. of EE. Systems, University of Southern California, 1992.

[351] Y.T. Lai, M. Pedram, S. Sastry, "BDD-based Decomposition of Logic Functions With Application to FPGA Synthesis," *Proc. of 30th DAC,* pp. 642-647, 1993.

[352] Y.T. Lai, K.R. Pan, M. Pedram, S. Vrudhula, "FGMap: A Technology Mapping Algorithm for Look-up Table Type FPGA Synthesis," *Proc. of IWLS,* pp. 642-647, 1993.

[353] Y.T. Lai, K.R. Pan, and M. Pedram, "FPGA Synthesis using Function Decomposition," 1994.

[354] U. Landmann, "Preprozessing-Methoden fuer die Synthese Kombinatorischer Schaltnetzwerke," *Dissertation,* T.U. Chemnitz, 1992, (in German).

[355] C. Lang, "Glitchpower in CMOS-Schaltungen. Praktikumsbericht," *IBM Laboratorien* Bueblingen, 1992, (in German).

[356] G.G. Langdon, "Decomposition Chart Technique to Aid in Realizations With Multiplexers," *IEEE Trans. on Comput.*, Vol. 27, No. 2, pp. 157-159, 1978.

[357] J. Langenderfer, "A Study of the Computational Complexities of Functions and their Inverses," Memorandum. *Wright Research and Development Center,* WL/AART Wright Patterson AFB, OH, Sept. 1989.

[358] F. Lapscher, "Decompositions Simples de Functions Booleans," *Automatisme*, No. 3, 14, 115, 1969.

[359] T. Q. Le, "Testbarkeit Kombinatorischer Schaltungen - Theorie und Entwurf," *Dissertation,* T.U. Karl-Marx-Stadt (Chemnitz), 1989, (in German).

[360] T.Q. Le, B. Steinbach, "Effiziente Methoden zum Logikentwurf Testbarer Kombinatorischer Schaltungen und deren Impliziten Testsatzberechnung," *3. ITG/GI-Workshop,* Blomberg, Muerz 1991, (in German).

[361] T.Q. Le, "Logikentwurf Digitaler Schaltungen," *Habilitation,* TU Chemnitz, 1992, (in German).

[362] R. Lechner and A. Moezzi, "The Synthesis of Encoded Programmable Logic Arrays in Spectral Techniques and Fault Detection," M. Karpovsky (Ed.) Orlando, FL, *Academic Press*, 1985.

[363] E.B. Lee and M.A. Perkowski, "A New Approach to Structural Synthesis of Automata," University of Minnesota, Dept. of Electrical Engineering, Technical Report, 1982.

[364] E.B. Lee and M.A. Perkowski, "Concurrent Minimization and State Assignment of Finite State Machines," *Proc. IEEE Int. Conference on Systems, Man and Cybernetics,* Oct. 1984. pp. 248–260.

[365] E. Lehman, Y. Watanabe, J. Grodstein, H. Harkness, "Logic Decomposition during Technology Mapping," *Proc. ICCAD'95*, pp. 264–271, 1995.

[366] F.T Leighton, "A Graph Coloring Algorithm for Large Scheduling Problems," *J. of Research. National Bureau of Standards,* Vol. 84, pp. 489-496, 1979.

[367] T.A. Leksich, S. Nikolich, "Computer-Systems and Automata Predictive Expansion of Switching Functions," *Engineering Cybernetics,* Vol. 18, No. 5, pp. 79-85, 1980.

[368] D. Lewin, *Computer BU,* Vol. 13, pp. 382, 1974. QA75.5 .C58

[369] D. Lewin, "Outstanding Problems in Logic Design," *Radio and Electronics Engineer,* Vol. 44, No. 1, pp.9-17, 1974.

[370] S.Y. Levy, R.O. Winder, T.H. Mott, "A Note on Tributary Switching Networks," *IEEE Trans. on Electr. Comput.*, Vol. EC-13, pp. 148-151, 1964.

[371] H.F. Li, "Variable Selection in Logic Synthesis Using Multiplexers," *Intern. J. of Electronics,* Vol. 49, No. 3, September 1980.

[372] M. Li and P. M. B. Vitányi, "Inductive Reasoning and Kolmogorov Complexity", *Journal of Computer and System Sciences,* Vol. 44, pp. 343-384, 1992.

[373] A. M. Lloyd, "Design of Multiplexer Universal-Logic-Module Networks Using Spectral Techniques," *IEE Proc. Pt. E.,* Vol. 127, pp. 31-36, January 1980.

[374] LogicBenchmarks, "Logic Benchmarks of International Workshop on Logic Synthesis," *MCNC*, North Carolina, May 1987.

[375] T. Luba, L. Wronski, "Decompositional Method of Realizing Systems of Switching Functions," *Review of Telecommunication,* No. 4, 1985, (in Polish).

[376] T. Luba, "Synthesis of Combinational Circuits Using Boolean Decomposition Method," *Publishers of Institute of Telecommunication,* Warsaw Technical University, No. 108, 1983, (in Polish).

[377] T. Luba, "New Approach to Boolean Function Decomposition for ROMs and PLAs," *Publishers of Institute of Telecommunication,* Warsaw Technical University, No. 123, 1985.

[378] T. Luba, "Synthesis of Combinational Circuits Using the Method of Boolean Decomposition," *Rozprawy Elektrotechniczne,* No. 2, 1985.

[379] T. Luba, "A Uniform Method of Boolean Function Decomposition," *Rozprawy Elektrotechniczne,* No. 4, pp. 1041-1054, 1986.

[380] T. Luba, L.Wronski, "Sequential Determination of Values of Boolean Expressions Using the Threshold Table Method," *Review of Telecommunication,* No.1, 1986, (in Polish).

[381] T. Luba, B. Zbierzchowski, "Topological Models of Boolean Functions and Their Application in the Synthesis of Complex Combinational Circuits," *Rozprawy Elektrotechniczne,* No.2, 1987.

[382] T. Luba, "Synthesis of Multi-Level Logic Circuits," *Prace Naukowe Politechniki Warszawskiej, Elektronika,* Vol. 79, 1988, Publishers of Warsaw Technical University, 00-665, Warszawa, Nowowiejska 24, Poland, (in Polish).

[383] T. Luba, J. Kalinowski, K. Jasinski, A. Krasniewski, "Combining Serial Decomposition with Topological Partitioning for Effective Multi-Level PLA Implementations", In *"Logic and Architecture Synthesis,"*, 1991. P. Michel and G. Saucier, (Editors), Elsevier Science Publisher B.V. (North Holland), pp. 243-252, 1991.

[384] T. Luba, J. Janowski, J. Rybnik, "Relations Between Multiple-Valued Logic and Decision Logic with Respect to Rough Set Theory Semantics," *Research Report, Institute for Telecommunications,* No. 139, 1991.

[385] T. Luba, J. Kalinowski, K. Jasinski, "PLATO - A Cad Tool for Logic Synthesis Based on Decomposition," *Proc. European Conference on Design Automation,* pp. 65-69, Amsterdam, February 1991.

[386] T. Luba, M.A. Markowski, B. Zbierzchowski, "Logic Decomposition for Programmable Gate Arrays," *Proc. of Euro-ASIC'92*, IEEE Computer Society Press, pp. 19-24, Paris 1992.

[387] T. Luba, J. Rybnik, "Rough Sets and Some Aspects in Logic Synthesis," In *Intelligent Decision Support - Handbook of Application and Advances of the Rough Sets Theory,* R.Slowinski (ed.), pp. 181-199, Kluwer Academic Publishers 1992.

[388] T. Luba, J. Rybnik, "Algorithm for Elimination of Attributes and Arguments Based on Unate Complement Concept," *Bull. Pol. Ac. , Tech.,* No. 40, pp. 313-322, 1992.

[389] T. Luba, J. Rybnik, "Rough Sets and Some Aspects in Logic Synthesis," in R. Slowinski (ed.), *Intelligent Decision Support - Handbook of Application and Advances of the Rough Sets Theory,* Kluwer Academic Publishers, 1992.

[390] T. Luba, J. Rybnik, "Algorithmic Approach to Discernibility Function with Respect to Attributes and Object Reduction", *Int. Workshop on Rough Sets: State of the Art and Perspectives,* Poznan 1992.

[391] T. Luba, K. Gorski, L.B. Wronski, "ROM-Based Finite State Machines with PLA Address Modifiers," *Proc. European Design Automation Conf.*, pp. 272-277, 1992.

[392] T. Luba, H. Selvaraj, A. Krasniewski, "A New Approach to FPGA-based Logic Synthesis," *Workshop on Design Methodologies for Microelectronics and Signal Processing,* pp. 135-142, Gliwice-Cracow 1993.

[393] T. Luba, R. Lasocki, J. Rybnik, "An Implementation of Decomposition Algorithm and its Application in Information Systems Analysis and Logic Synthesis," *International Workshop on Rough Sets and Knowledge Discovery,* pp. 487-498, Banff 1993,

[394] T. Luba, J. Rybnik, " Algorithmic Approach to Discernibility Function with Respect to Attributes and Objects Reduction," *Foundation of Computing and Decision Sciences,* Vol. 18, No. 3-4, pp. 241-258, 1993.

[395] T. Luba, H. Selvaraj, A. Krasniewski, "A New Approach to FPGA-based Logic Synthesis", *Workshop on Design Methodologies for Microelectronics and Signal Processing,* Gliwice-Cracow, 1993.

[396] T. Luba, M. Mochocki, J. Rybnik, "Decomposition of Information Systems Using Decision Tables," *Bulletin of the Polish Academy of Sciences, Technical Sciences,* Vol. 41, No.3, 1993.

[397] T. Luba, R. Lasocki, "Decomposition of Multiple-valued Boolean Functions," *Applied Mathematics and Computer Science,* Vol.4, No.1, pp. 125-138, 1994.

[398] T. Luba, "Decomposition of Multiple-Valued Functions," *Proc. 25th ISMVL*, pp. 256-261, 1995.

[399] T. Luba, "Multiple-Valued and Multiple-Level Decomposition and Its Applications in Information Systems Analysis and Logic Synthesis," *Report on Window-on-Science Project WOS-95-2155*, June 1995.

[400] T. Luba, M. Nowicka, "Balanced Multi-Level Decomposition and Its Applications in FPGA-based Synthesis," *Report*, Warsaw University of Technology, 1995.

[401] M. Luby, "A simple parallel algorithm for the maximal independent set problem," *SIAM J. Comput.*, Vol. 15, No. 4., pp. 1036–1055, Nov. 1986.

[402] G. Luger, W. Stubbefield, "Artificial Intelligence. Structures and Strategies for Complex Problem Solving," *Benjamin/Cummings*, 1993.

[403] O.B. Lupanov, "On the Synthesis of Certain Classes of Control Devices," *Collection: Problems of Cybernetics,* in Russian, No. 10, 1961.

[404] E. Macii, T. Wolf, "Multiple Stuck-At Fault Test Generation Techniques for Combinational Circuits Based on Network Decomposition," *Proc. 36th Midwest Symposium*, pp. 465-467, 1993.

[405] V. Madhavan, "Approximation algorithm for maximum independent set in planar triangle-free graphs," *Foundations of software technology and theoretical Computer Science*, Proc. of 4th conference, Bangalore, India, Dec. 13-15, 1984, Joseph, M., Shyamasundar, R. (eds), Lecture Notes in Computer Science, No. 181, Springer Verlag, N.Y., 1984.

[406] K.K. Maitra, "Cascaded Switching Networks of Two-Input Flexible Cells," *IRE Trans. Electr. Comput.*, Vol. EC-11, pp. 136-143, Apr. 1962.

[407] O. Maler, A.Pnueli, "Tight Bounds on the Complexity of Cascaded Decomposition of Automata," *Proc. of the 31t Annual Symposium on Foundation of Computer Science,* pp. 672-682., October 22-24 1990.

[408] G. Maley, J. Earle, "The Logic Design of Transistor Digital Computers," Englewood Cliffs, N.J. Prentice Hall, 1963.

[409] S. Malik, A. R. Wang, R. K. Brayton, and A. Sangiovanni-Vincentelli, "Graph-Based Algorithms for Boolean Function Manipulation," *IEEE Proc. of. Int. Conf. on CAD*, pp. 6-9, 1988.

[410] B. Manvel, "Coloring Large Graphs," *Proc. of the 1981 S.E. Conf. on Graph Theory, Combin. and Computer Science,* 1981.

[411] B. Manvel, "Extremely Greedy Coloring Algorithms," in "Graphs and Applications", *Proc. of the FIrst Colorado Symposium on Graph Theory*, F. Harray, and J. Maybee, Eds. John Wiley, 1985.

[412] M.P. Marcus, "Derivation of Maximal Compatibles Using Boolean Algebra," *IBM J. Res.Dev.*, Vol 8, pp. 537–538, 1964.

[413] P.N. Marinos, "Fuzzy logic and its application to switching systems", *IEEE Trans. on Comput.*, 18 (4), pp. 345-348, 1969.

[414] A. Maruoka, N. Honda, "Logical Networks of Flexible Cells," *IEEE Trans. Comput.*, Vol. C-22, pp. 347-358, Apr. 1973.

[415] A. Maruoka, N. Honda, "The Range of Flexibility of Tree Networks," *IEEE Trans. on Comput.*, Vol C-24, pp. 9-28, Jan. 1975.

[416] H.J. Mathony, "Zum Entwurf mehrstufiger Schaltnetze," *Interner Bericht* Bl-2.85, Univ. Karlsruhe, 1985, (in German).

[417] H. -J. Mathony, U.G. Baitinger, "CARLOS: An Automated Multilevel Logic Design System for CMOS Semi - Custom Integrated Circuits," *IEEE Transaction on Computer-Aided Design,* Vol. 7, No. 3, March 1988.

[418] H. -J. Mathony, "Algorithmische Entwurfsverfahren fuer Zwei- und Mehrstufige Schaltnetze," *Fortschritt-Berichte VDI,* Reihe 9, Nr. 79, VDI-Verlag, Duesseldorf 1988, (in German).

[419] D.W. Matula, "A Min-Max Theorem for Graphs with Application to Colouring," *J. of SIAM,* Review, 10., p.481, 1968.

[420] D.W. Matula, G. Marble, and J.D. Isaacson, "Graph Coloring Algorithms," *Graph Theory and Computing,* (R.C. Read, ed.), Academic Press, New York (1972), pp. 109-122.

[421] E. J. McCluskey, "Minimization of Boolean Functions," *Bell System Technical Journal,* Vol. 35, pp. 1417-1445, Nov. 1956.

[422] Ch. Meinel, J. Bern, and A. Slobodova, "Efficient OBDD-Based Boolean Manipulation in CAD Beyond Current Limits," *Proc. 32nd DAC,* San Francisco 1995.

[423] B. Melzer, "Dekompositorische Synthese Kombinatorischer Schaltungen mit Impliziter Testsatzberechnung," *Diplomarbeit,* TU Chemnitz, 1992, (in German).

[424] A.R. Meo, "Modular Tree Structures," *IEEE Trans. on Comput.*, Vol.17, 432-442, pp. 559-566, June 1968.

[425] R.D. Merril, "Symmetric Ternary Switching Functions: Their Detection and Realization with Threshold Logic," *IEEE Trans. on Electr. Comput.*, Vol. EC-16, No. 5, pp. 634-637,.

[426] . A. Michalski, Poland

[427] R. Michalski, J. Carbonell, and T.M. Mitchell, *"Machine Learning: An Artificial Intelligence Approach," Morgan Kaufmann,* vol. 1, 1983, vol. 2. 1986.

[428] R.E. Miller, "Switching Theory," Vol. 1 and 2, John Wiley, New York, 1965.

[429] D.M. Miller, "Decomposition in many-valued design," *Ph.D. Thesis*, Dept. of Computer Science, University of Manitoba, May 1976.

[430] G.J. Minty, "On maximal independent sets of vertices in claw-free graphs," *J. Combin. Theory,* B., Vol. 28., pp. 284–304, 1980.

[431] R. Miller, "Teorija Pereklucatelnych Skhem, Vol. 1. Kombinacjionnyje Schemy," Moscow 1970 (in Russian).

[432] D.M. Miller, J.C. Muzio, "Two-place Decoposition and the Synthesis of Many-Valued Switching Circuits," *Proc. 6th ISMVL,* Logan, UT, pp. 164-168, May 1976.

[433] D.M. Miller, "The Fanout-Free Realization of Multiple-Valued Functions," *Proc. 11th ISMVL,* 1981, pp. 246–255.

[434] D.M. Miller, "Spectral Symmetry Tests," Proc. 11th ISMVL, 1981, pp. 130-134.

[435] S. Minato, N. Ishiura, and S. Yajima, "Shared Binary Decision Diagram with Attributed Edges for Efficient Boolean Function Manipulation," *Proc. 27th ACM/IEEE DAC,* pp. 52-57, 1990.

[436] S. Minato, "Graph-Based Representations of Discrete Functions," *Proc. Reed-Muller'95 Workshop*, Chiba, Japan, August 1995, pp. 1-10.

[437] R.C. Minnick, "A System of Magnetic Bubble Logic," *IEEE Trans. Comput.*, Vol. C-24, pp. 217-218, Febr. 1975.

[438] J. Mitchem, "On Various Algorithms for Estimating the Chromatic Number,"

[439] G.J. Montgomery, and K.C. Drake, "Abductive Networks", *SPIE Applications of Neural Networks Conference,* April 1990,

[440] J.W. Moon, and L. Moser, "On Cliques in Graphs," *Israel J. of Mathematics*, Vol. 3, pp. 23–28, March 1965,

[441] W.R. Moore, W. Luk, "FPGAs," *Abington EE and CS Books,* Abington, England, 1991.

[442] C. Moraga, "Spectral Analysis of Co-Symmetries", *Proc of 3rd Intern. Workshop of Spectral Techniques*, pp. 127-139, ISSN 0933-6192, Dortmund 1988.

[443] Motorola MPA10XX Data Sheet, 1994.

[444] A. Muciek, "On Identification of Dimensional Function of Many Variables," *Proc. IEEE Instrumentation and Measurement Technology Conference,* pp. 209-212, 1991.

[445] K.D. Mueller, "Reduktion der Verlustleistung von CMOS-Schaltungen durch logisch-strukturelle Maenahmen," *Dissertation, Universitaet Karlsruhe,* 1977, (in German).

[446] A. Mueschwitzer, F. Rueler, "VLSI - Systeme," *VEB Verlag Technik,* Berlin 1988, (in German).

[447] A. Mukhophadyay, "Symmetric Ternary Switching Functions," *IEEE Trans. on Comp.* Vol. EC-15, pp. 731-739, Oct. 1966.

[448] A. Mukhopadhyay, "Unate Cellular Logic," *IEEE Trans. Comput.*, Vol. C-18, pp. 114-121, Febr. 1969.

[449] A. Mukhopadhyay and Stone, "Cellular Logic," *Academic Press*, New York, 1971.

[450] G.D. Muligan, and D.G. Corneil, "Corrections to Bierstone's Algorithm for Generating Cliques," *JACM,* Vol. 19, No. 2, pp. 244–247, April 1972.

[451] D. E. Muller, "Application of Boolean Algebra to Switching Circuit Design and to Error Detection", *IRE Trans. on Electr. Comp.*, vol. EC-3, pp. 6-12, Sept. 1954.

[452] R. Murgai, Y. Nishizaki, N. Shenoy, R.K. Brayton, A. Sangiovanni - Vincentelli, "Logic Synthesis for Programmable Gate Arrays," *Proc. 27th ACM/IEEE Design Automation Conf.,* pp. 620-625, 1990.

[453] R. Murgai, N. Shenoy, R.K. Brayton, A. Sangiovanni-Vincentelli, "Improved Logic Synthesis Algorithm for Table Look Up Architectures," *Proc. IEEE Intern. Conf. on Computer Aided Design,* pp. 564-567, 1991.

[454] R. Murgai, N. Shenoy, R.K. Brayton, A. Shangiovanni-Vincentelli, "Performance Directed Synthesis for Table Look Up Programmable Gate Arrays," *Proc. ICCAD 1991*, Santa Clara, CA, Nov. 1991, pp. 572-575.

[455] R. Murgai, R.K. Brayton, and A. Sangiovanni-Vincentelli, "Optimum Functional Decomposition Using Encoding," *Proc. 31st DAC*, pp. 408–414, June 1994.

[456] R. Murgai, M. Fujita, F. Hirose, "Logic Synthesis for a Single Large Look-up Table," *Proc. ICCD'95*, pp. 415-424, 1995.

[457] D. Muroga, H.C. Lai, "Minimization of Logic Networks under a Generalized Cost Function," *IEEE Trans. on Comput.*, pp. 893-??, 1976.

[458] S. Muroga, "Logic Design and Switching Theory," *John Wiley and Sons,* 1979.

[459] J.C. Muzio, S.L. Hurst, "The Computation of Complete and Reduced Sets of Orthogonal Spectral Coefficients for Logic Design and Pattern Recognition Purposes," *Computers Electr. Engng.*, Vol. 5, pp. 231-249, 1978.

[460] J.C. Muzio, D.M. Miller, G.Epstein, "The Simplification of Multiple-Valued Symmetric Functions," *Proc. of the 13th Intern. Symp. on Multiple-Valued Logic*, pp. 111-119, May 1983.

[461] J.C. Muzio, T.C. Wesselkamper, "Multiple-Valued Switching Theory," *Adam Hilger,* Boston, MA, 1986.

[462] J.C. Muzio, "Concerning the Maximum Size of the Terms in the Realization of Symmetric Functions," *Proc. 20th Intern. Symp. on Multiple-Valued Logic,* May 1990, pp. 292-299.

[463] J. Neuschwander, J. Beister, "Uher die Ursachen des Realisierungsaufwands von Schaltfunktionen," *Informationstechnik it,* 28. Jg. No. 2, 1986, (in German).

[464] L. Nguyen, M.A. Perkowski and N.B. Goldstein, "PALMINI – Fast Boolean Minimizer for Personal Computers," Proc. 24th. ACM/IEEE Design Automation Conference, 1987, pp. 615–621.

[465] M. Perkowski, N. Nguyen, "Minimization of Finite State Machines in System SuperPeg," *Proc. of the Midwest Symposium on Circuits and Systems,* pp. 139 - 147, Luisville, Kentucky, 22-24 August 1985.

[466] E.A. Nordhous, and J.W. Gaddum, "On Complementary Graphs", *American Math. Monthly*, 63, p.175, 1956.

[467] C.B. Novikov, C.B. Oleksin, "Riealizacija Sistiemy Slabaopriedielionnych Bulievych Funkcji Bolszovo Czisla Pieremiennych Setiju iz PLM," *YC i M,* No. 2, 1984, (in Russian).

[468] M.J. Noviskey, "Correlation Partition Selection Algorithm," *Technical Report,* WL/AART-2, W/P AFB, OH 45433-7408, August 1992.

[469] M.J. Noviskey, "Row Identification for Function Decommposition in Pattern Theory," *Technical Report,* WL/AART-2, W/P AFB, OH 45433-7408, May 1993.

[470] M. J. Noviskey, T.D. Ross, D.A. Gadd, M. Axtell, "Application of Genetic Algorithms to Function Decomposition in Pattern Theory," *report WL-TR-94-1015,* 1994.

[471] B.R. Nyers and R. Lin , "A Lower Bound on the Chromatic Number of a Graph," *Networks,* Vol. 1., p. 273., 1972.

[472] A.L. Oliveira, and A. Sangiovanni-Vincentelli, "Constructive Induction Using a Non-Greedy Strategy for Feature Selection," *Proceedings on the Ninth International Conference on Machine Learning,* pp. 355-360, 1992.

[473] A.L. Oliveira, and A. Sangiovanni-Vincentelli, "Learning Complex Boolean Functions: Algorithms and Applications", *Implementation and Simulation - VLSI*, preliminary copy 6/10/93.

[474] A.L. de Oliveira, *"Inductive Learning by Selection of Minimal Complexity Representations,"* Ph.D. Thesis, University of California at Berkeley, Dec. 1994.

[475] A. Oliveira, 1994.

[476] G. Pagallo, and D. Hausler, "Boolean Feature Discovery in Empirical Learning", *Machine Learning*, Vol. 5, pp. 71-99, 1990.

[477] A. Pal, "An Algorithm for Optimal Logic Design Using Multiplexers," *IEEE Trans. on Comput.*, Vol. C-35, No.8, August 1986.

[478] C.H. Papadimitriou, "The clique problem for planar graphs," *Inf. Proc. Letters,* Vol. 13, No. 4,5, pp. 138–141, December 1981.

[479] C.A. Papachristou, "Implementation of Discrete an Residue-Based Functions via Optimal Encoding: a PAL Approach," *IEEE Trans. on Comput.*, Vol. C-32, October 1983.

[480] C. Papachristou, "Characteristic Measures of Switching Functions," *Information Sciences,* Vol. 13, No. 1, pp.51-75, 1977.

[481] Z. Pawlak, "Rough Sets," *Theoretical Aspects of Reasoning about Data,* Kluwer Academic Publishers, Dordrecht, 1991.

[482] D. Patel, T. Luba, "Dependence Sets and Functional Decomposition of Boolean Functions," *Intern. J. of Electr.*, Vol. 75, No. 2, pp. 177-198, August 1993.

[483] D. Paulson, Y. Wand, "An Automated Approach to Information Systems Decomposition," *IEEE Trans. Soft. Engng.,* 18, pp. 174-189, 1992.

[484] J.Peck and M. Williams, "Examination Scheduling," *Communication of ACM,* Vol. 9, pp. 433–434, 1966.

[485] M. Perkowski, "Synthesis of Multioutput Three Level NAND Networks," *Proc. of the Seminar on Computer Aided Design,* IFAC - Intern. Federation of Control, pp. 238 - 265, Budapest, Hungary, 3-5 November 1976.

[486] M. Perkowski, H. Uong, "Generalized Decomposition of Incompletely Specified Multioutput, Multi-Valued Boolean Functions," *Unpublished manuscript, Department of Electrical Engineering,* PSU 1987.

[487] M. A. Perkowski, M. Helliwell, P. Wu, "Minimization of Multiple-Valued Input, Multi-Output Mixed-Radix Exclusive Sums of Products for Incompletely Specified Boolean Functions," *Proc. of the 19th ISMVL, International IEEE Symposium on Multiple-Valued Logic,* pp. 256 - 263, Guangzhou, People's Republic of China, May 1989.

[488] M. Perkowski, J. Brown, "A Unified Approach to Designs with Multiplexers and to the Decomposition of Boolean Functions," *Proc.ASEE Annual Conference,* pp.1610-1619, 1988.

[489] M.A. Perkowski, P. Wu, and K.A. Pirkl, "KUAI-EXACT: A New Approach for Multi-Valued Logic Minimization in VLSI Synthesis," *Proc. 1989 IEEE International Symposium on Circuits and Systems*, pp. 401–404, 1989.

[490] M.A. Perkowski, P. Dysko, and B.J. Falkowski, "Two Learning Methods for a Tree-Search Combinatorial Optimizer," *Proc. of IEEE Int. Conf. on Comput. and Comm.,* pp. 606-613, Scottsdale, Arizona, 1990.

[491] M.A. Perkowski, J. Liu, J.E. Brown, "Rapid Software Prototyping: CAD Design of Digital CAD Algorithms," In G. W. Zobrist (ed), *Progress in Computer-Aided VLSI Design,* Vol. 1, pp. 353-401, 1989.

[492] M.A. Perkowski, and P. D. Johnson, "Canonical Multi-Valued Input Reed-Muller Trees and Forms", *Proc. 3rd NASA Symposium on VLSI Design,* Moscow, Idaho, pp. 11.3.1 - 11.3.13, October 1991.

[493] M. A. Perkowski, "The Generalized Orthonormal Expansions of Functions with Multiple-Valued Inputs and Some of its Applications," *Proc. 22nd ISMVL,* pp. 442-450, Sendai, Japan, May 1992.

[494] M. Perkowski, L. Csanky, A. Sarabi, I. Schaefer, "Fast Minimization of Mixed-Polarity ADN/XOR Canonical Forms", *Proc. of the IEEE Intern. Conf. on Comp. Design,* ICCD'92, Boston, October 11-13, 1992, pp. 32-36.

[495] M. Perkowski, "The Generalized Orthonormal Expansion of Functions with Multiple-Valued Inputs and Some of its Applications", *Proc. of the ISMVL'92, the 21st IEEE Intern. Symp. on Multiple-Valued Logic,* Sendai, Japan, May 27-29, 1992, pp. 442-450.

[496] M. A. Perkowski, "A Fundamental Theorem for for EXOR Circuits," *Proc. of IFIP W.G. 10.5 Workshop on Applications of the Reed-Muller Expansion in Circuit Design",* Hamburg, Germany, September 16-17, 1993.

[497] M. A. Perkowski, A. Sarabi, and F. R. Beyl, "XOR Canonical Forms of Switching Functions," *Proc. of IFIP W.G. 10.5 Workshop on Applications of the Reed-Muller Expansion in Circuit Design",* Hamburg, Germany, September 16-17, 1993.

[498] M. A. Perkowski, I. Schaefer, A. Sarabi, and M. Chrzanowska-Jeske, "Multi-Level Logic Synthesis Based on Kronecker Decision Diagrams and Boolean Ternary Decision Diagrams for Incompletely Specified Functions," *accepted to Journal on VLSI Design,* April 1994.

[499] M.A. Perkowski, Ph. Ho, "Free Kronecker Decision Diagrams and their Application to Atmel 6000 Series FPGA Mapping," *Proc. Euro-DAC'94*, Grenoble, Sept. 19-23, 1994, France.

[500] M. Perkowski, T. Ross, D. Gadd, J. A. Goldman, and N. Song, "Application of ESOP Minimization in Machine Learning and Knowledge Discovery," *Proc. Reed-Muller'95 Workshop,* Chiba, Japan, August 1995, pp. 102-109.

[501] M. Perkowski, T. Luba, S. Grygiel, M. Kolsteren, R. Lisanke, N. Iliev, P. Burkey, M. Burns, R. Malvi, C. Stanley, Z. Wang, H. Wu, F. Yang, S. Zhou, and J. S. Zhang, "Unified Approach to Functional Decompositions of Switching Functions," *PSU Report*, unpublished, Version III, September 1995.

[502] M. Perkowski, "A New Representation of Strongly Unspecified Switching Functions and its Application to Multi-Level AND/OR/EXOR Synthesis," *Proc. Reed-Muller'95 Workshop,* Chiba, Japan, August 1995, pp. 143-151.

[503] M. Perkowski, M. Marek-Sadowska, T. Luba, S. Grygiel, P. Burkey, R. Malvi, Z. Wang, J.S. Zhang, and C. Stanley, "Cube Diagram Bundles: A New Representation of Multi-Valued Relations," *submitted to ISMVL'96.*

[504] M. Perkowski, M. Marek-Sadowska, T. Luba, S. Grygiel, P. Burkey, R. Malvi, Z. Wang, J.S. Zhang, and C. Stanley, "Fundamental Operations on Strongly Unspecified Multi-Valued Functions and Relations," *submitted to ISMVL '96.*

[505] M. Perkowski, M. Marek-Sadowska, T. Luba, S. Grygiel, P. Burkey, R. Malvi, Z. Wang, J.S. Zhang, and C. Stanley, "GUD-MV: Multi-Level Decomposition of Multi-Valued Functions and Relations," *submitted to ISMVL '96.*

[506] Perkowski et al, "Development of the Search Strategies for MULTIS," *report*, 1995.

[507] O. Persin, "An Algorithm for Determining the Minimum Coloring of a Finite Graph," *Engineering Cybernetics,* Vol. 11, pp. 980–985, 1973.

[508] E. Pichat, "Decomposition d'une Fonction de Variables a Nombres Finites de Valeurs," *Compt. Rend. H. Acad. Sci,* Vol. 268, ser. A, pp. 761-763, Nov. 18, 1969, (in French).

[509] E. Pichat, "Decomposition d'une Fonction de Plusiers Variables," *Compt. Rend. H. Acad. Sci,* Vol. 268, ser. A, pp. 1523-1526, June 23, 1969, (in French).

[510] E. Pierzchala, M.A. Perkowski, S. Grygiel, "A Field Programmable Analog Array for Continuous, Fuzzy, and Multi-Valued Logic Applications," *Proc. ISMVL '94*, pp. 148 - 155.

[511] D.A. Pospelov, "Analyse und Synthese von Schaltsystemen," *VEB Verlag Technik,* Berlin 1973, (in German).

[512] C. Posthoff, B. Steinbach, "Binaere Dynamische Systeme - Algorithmen und Programme," *Wissenschaftliche Schriftenreihe der T.H. Karl-Marx-Stadt,* (Chemnitz), 8, 1979, (in German).

[513] C. Posthoff, B. Steinbach, "Binaere Gleichungen - Algorithmen und Programme," *Wissenschaftliche Schriftenreihe der TH Karl-Marx-Stadt,* (Chemnitz) 1, 1979, (in German).

[514] Ch. Posthoff, D. Bochmann, K. Haubold, "Diskrete Mathematik," *Teubner-Verlag*, Leipzig, 1986, (in German).

[515] J. Poswig, "Disjoint decomposition," *Proc. of 3rd. International Workshop on Spectral Techniques,* Dept. Comp. Science, University of Dortmund, Dortmund 1988.

[516] J. Poswig, "Disjoint Decomposition of Boolean Functions," *IEE Proceedings,* Vol. 138, No. 1, pp. 48-56, January 1991.

[517] G.N. Povarov, "About Functional Separability of Boolean Functions", May 1954, (in Russian).

[518] G.N. Povarov, "On Functional Decomposability of Boolean Functions," *Doklady AN SSSR*, 94, No. 5, 1958.

[519] R.E. Prather, "Three Variable Multiple-output Tree Circuits," *IEEE Comput.*, Febr. 1965.

[520] R.E. Prather, "Introduction to Switching Theory: A Mathematical Approach," Boston, MA, *Allyn and Bacon*, 1967.

[521] F.P. Preparata, "Universal Logic Modules of a New Type," *IEEE Trans. Comput.*, Vol. C-21, pp. 585-588, June 1972.

[522] F.P. Preparata, "On the Design of Universal Boolean Functions," *IEEE Trans. on Comput.*, Vol. C-20, pp. 418-423, April 1971.

[523] E.I. Pupyrev, "Realization of Control Algorithms in a Rearrangeable Automata Basis," *Automation and Remote Control,* USSR, Vol. 50, No. 3, pp. 411-421, 1989.

[524] L. Pyber, "Clique covering of graphs," *Combinatorica*, Vol. 6, No.4, pp. 393–398, 1986.

[525] J. R. Quinlan, *"C4.5: Programs for Machine Learning",* Morgan Kaufmann, 1993, Palo Alto, Ca.

[526] R.C. Read, "Graph Theory and Computing," *Academic Press*, New York, 1972.

[527] I.S. Reed, "A Class of Multiple-Error-Correcting Codes and the Decoding Scheme", *IRE Transactions on Information Theory,* Vol. IT-4, pp. 38-49, September 1954.

[528] E.M. Reingold E.M., J. Nievergelt, and N. Deo, "Combinatorial Algorithms: Theory and Practice," *Prentice-Hall*, 1977.

[529] N.P Redkin, "Avtomatika i Telemekhanika," No. 8, pp. 84-88, (in Russian), 1970.

[530] D.C. Rine, "Computer Science and Multiple-Valued Logic," North Holland, 1977.

[531] T.K. Rosenfeld, V.N. Silayev, "Boolean Equations and Decomposition of Boolean Functions," *Engineering Cybernetics,* Vol. 17, No. 1, pp. 85-92, 1979.

[532] W. Rosenstiel (Ed.), *Proceedings of the IFIP WG 10.5 Workshop on Applications of the Reed-Muller Expansion in Circuit Design,* Hamburg, Germany, September 1993.

[533] T.D. Ross, "Pattern Representation and Recognition," *Research Prospectus,* Air Force Institute of Technology. 1988.

[534] T.D. Ross, "Elementary Theorems in Pattern Theory," *Ph.D. Thesis,* Air Force Institute of Technology. 1988.

[535] T.D. Ross and A.V. Lair, "On the Role of Patterns in Recognizer Design," In *Josef Kittler, ed. Pattern Recognition,* pp. 193-202, Springer-Verlag, New York, 1988.

[536] T.D. Ross and A.V. Lair, "Definition and Realization in Pattern Recognition System Design," In *Proc. of 1987 IEEE Int. Conf. on Systems, Man and Cybernetics,* pp. 744-748, 1987.

[537] T. D. Ross, M.J. Noviskey, T.N. Taylor, D.A. Gadd, "Pattern Theory: An Engineering Paradigm for Algorithm Design," *Final Technical Report WL-TR-91-1060,* Wright Laboratories, USAF, WL/AART/WPAFB, OH 45433-6543, August 1991.

[538] T.D. Ross, M.L. Axtell, M.J. Noviskey, M. Breen, "A Demonstration of a Robust Occam-Based Learner," In *Proc. IEEE Int. Symp. on Information Theory,* 1992.

[539] T.D. Ross, "Function Decomposition Strategy for the Function Learning and Synthesis Hotbed," *Technical Memorandum WL-TM-92-110,* Wright Laboratory, USAF, WL/AART, WPAFB, OH 45433-6543, August 1992.

[540] T.D. Ross, M.J. Noviskey, M.L. Axtell, D.A. Gadd, "Flash Software Description," *Technical report, Wright Laboratory,* USAF, WL/AART, WPAFB, OH 45433-6543, December 1993.

[541] T.D. Ross, M.L. Axtell, M.J. Noviskey, "Logic Minimization as a Robust Pattern Finder," *Intern. Workshop on Logic Synthesis,* Lake Tahoe, CA, May 23-26, 1993.

[542] T.D. Ross, M.J. Noviskey, M.L. Axtell, D.A. Gadd, "Flash user's guide," *Technical report, Wright Laboratory,* USAF, WL/AART, WPAFB, OH 45433-6543, December 1993.

[543] T.D. Ross, M.L. Axtell, M.J. Noviskey, D.A. Gadd, "Pattern Theory Paradigm for System Design," In *Proc. 36th Midwest Symposium on Circuits and Systems,"* 1993.

[544] T.D. Ross, J.A. Goldman, M.J. Noviskey, M.L. Axtell, D.A. Gadd, "Graph Coloring for Column Multiplicity: A Survey," WL WPAFB report, April 21, 1994.

[545] T.D. Ross, M.J. Noviskey, M.L. Axtell, D.A. Gadd, J.A. Goldmann, "Pattern Theoretic Feature Extraction and Constructive Induction," *Technical report,* Wright Laboratory, USAF, WL/AART, WPAFB, OH 45433-6543, April 22, 1994.

[546] T.D. Ross, M.J. Noviskey, J.A. Goldmann, D.A. Gadd, "On the Decomposition of Real-Valued Functions," *In Third Intern. Workshop on Post-Binary VLSI Systems,* 1994.

[547] T.D. Ross, "Variable Partition Search for Function Decomposition," *Technical report, Wright Laboratory,* USAF, WL/AARA-3, WPAFB, OH 45433-6543, November 1994.

[548] J.P. Roth, "Minimization over Boolean Trees," *IBM Journal.* 4, 5, pp. 543-555, 1960.

[549] J.P. Roth, R.M. Karp, "Minimization Over Boolean Graphs," *IBM Journal Res. and Develop.* No.4, pp. 227-238 (543-558?), April 1962.

[550] J.P. Roth and E.G. Wagner, "Algebraic Topological Methods for the Synthesis of Switching Systems. Part III: Minimization of Non-Singular Boolean Trees," *IBM J. Res. Develop.* Vol. 3, Oct. 1962.

[551] D. Rottem, and J. Urrutia, "Finding maximum cliques in circle graphs," *Networks II*, Vol. 3., pp. 269–278, Fall 1981.

[552] R.L. Rudell, "Espresso IIC User's Manual," CAD Toolbox User's Manual, UC Berkeley, Jan. 1984.

[553] R.L. Rudell, and A. Sangiovanni–Vincentelli, "Espresso MV: Algorithms for Multiple-Valued Logic Minimization," *Proc. IEEE Custom Integr. Circ. Conf.,* 1985, pp. 230–234.

[554] R.L. Rudell and A.L. Sangiovanni-Vincentelli, "Exact Minimization of Multiple–Valued Functions for PLA Optimization," Digest of Technical Papers, *IEEE Int. Conf. on Computer-Aided Design,* pp. 352-355, 1986.

[555] R.L. Rudell, *Multiple-Valued Logic Minimization for PLA Synthesis,* Masters Report, University of California, Berkeley, 1986.

[556] R.L. Rudell, and A.L. Sangiovanni-Vincentelli, "Multiple-valued minimization for PLA optimization," *Proc. Intern. Symp. on Multiple-Valued Logic,* pp. 198-208, May 26-28, Boston, MA, 1987.

[557] R.L. Rudell and A. Sangiovanni–Vincentelli, "Multiple-Valued Minimization for PLA Optimization," *IEEE Transactions on Computer–Aided Design,* Vol. CAD-6, No. 5, Sept. 1987, pp. 727–750.

[558] R. Rudell, "Dynamic Variable Ordering for Ordered Binary Decision Diagrams," *Intern. Workshop on Logic Synthesis,* pp. 3a 1-12, 1993.

[559] R. Rudell, "Dynamic Variable Ordering for Ordered Binary Decision Diagrams," *Proc. Int. Conf. on Computer Aided Design*, pp. 42-47, 1993.

[560] J. Rybnik, "Minimization of Partially Defined Switching Functions Using Rough Sets Theory," *Manuscript, Institute of Telecommunication*, Warsaw Technical University, 1990, (in Polish).

[561] M. Marek-Sadowska, "Detecting Symmetric Variables in Boolean Functions using Generalized Reed-Muller Forms," *Proc. ISCAS'94*, pp. 287-290, 1994.

[562] Sakawa, Masatosh, "Fuzzy Sets and interactive multiobjective optimization," *Plenum Press,* 233 Spring Street, New York, NY 10013, 1993.

[563] A. Sangiovanni-Vincentelli, A. Gamal, J. Rose, "Synthesis Methods for Field Programmable Gate Arrays," *Proc. IEEE,* Vol.81, No.7, pp. 1057-1083, 1993.

[564] E.S. Santos, "Maxmin Automata," *Information and Control,* pp. 363-377, 1968.

[565] J. Santos, H. Arango, M. Pascual, G. Roing, "A Cyclic Algebra for the Synthesis of Ternary Digital Systems," *IEEE Trans. on Comput.*, Vol. C-19, pp. 651-653, July 1970.

[566] A.A. Sapozenko, L.M. Karahanijan, *Avtomatika i Vycislitelnaya Tekhnika,* No. 3, pp. 28-35, 1981, (in Russian).

[567] A. Sarabi, M.A. Perkowski: "Fast Exact and Quasi-Minimal Minimization of Highly Testable Fixed-Polarity AND/XOR Canonical Networks", *Proc. of the IEEE/ACM Design Automation Conference,* Anaheim, CA, June 8-12, 1992, pp. 30-35.

[568] A. Sarabi, P.F. Ho, K. Iravani, W.R. Daasch, and M. Perkowski, "Minimal Multi-level Realization of Switching Functions based on Kronecker Functional Decision Diagrams," *Proc. IWLS '93, Intern. Workshop on Logic Synthesis*, 1993.

[569] A. Sarabi, M.A. Perkowski, "Cube Based Method for Optimal and Quasi-Optimal Minimization of Consistent Generalized Reed-Muller Expansions", *submitted to IEEE Transactions on Computers,* 1994.

[570] T. Sasao, "An application of multiple-valued logic to a design of Programmable Logic Arrays," *Proc. 8th Intern. Symp. on Multiple-Valued Logic (ISMVL),* 1978.

[571] T. Sasao, "Multiple-Valued Decomposition of Generalized Boolean Functions and the Complexity of Programmable Logic Arrays," *IEEE Trans. on Comput.*, Vol. 30, No. 9, pp. 635-643, 1981.

[572] T. Sasao, "Input Variable Assignment and Output Phase Optimization of PLAs," *IEEE Trans. on Comput.*, Vol. C-33, No. 10, pp. 879-894, October 1984.

[573] T. Sasao, "An Algorithm to Derive the Complement of a Binary Function with Multiple-Valued Inputs," *IEEE Transactions on Computers,* Vol. C-34, No. 2, Feb. 1985, pp. 131–140.

[574] T. Sasao, "MACDAS: Multi-Level AND-OR Circuit Synthesis Using Two-Variable Function Generators," *Proc. 23rd Design Automation Conference*, 1986, pp. 86-93.

[575] T.Sasao, "Functional Decomposition of PLAs", *Proc. of the Intern. Workshop on Logic Synthesis,* Research Triangle Park, North Carolina, May 12-15, 1987.

[576] T. Sasao, "Bounds on the average number of products in the minimal Sum-of-Products expressions for multiple-valued input two-valued output functions," *Proc. Intern. Symp. on Multiple-Valued Logic,* pp. 260-267, May 26-28, Boston, MA, 1987.

[577] T.Sasao, "Aplication of Multiple-Valued Logic to a Serial Decomposition of PLAs", *Proc. of the Intern. Symp. on Multiple-Valued Logic,* Zangzou, China, pp. 264-271, May 1989.

[578] T. Sasao, and Ph. Besslich, "On the Complexity of MOD-2 Sum PLAs," *IEEE Trans. on Comput.*, Vol. 39, No. 2, pp. 262-266, 1990.

[579] T. Sasao (ed.), "Logic Synthesis and Optimization," *Kluwer Academic Publishers*, 1993.

[580] T. Sasao, "FPGA Design by Generalized Functional Decomposition," in "Logic Synthesis and Optimization, " T. Sasao. (Ed), Kluwer Academic Publishers, pp. 233-258, 1993.

[581] T. Sasao, "Exmin2: A simplification algorithm for exclusive-OR sum of products expressions for multiple-valued-input two-valued output functions," *IEEE Tr. on CAD*, Vol. 12, No. 5, pp. 621–632, 1993.

[582] D. Sass, H. Warmers, E.-H. Horneber, "Waveform Estimation and Hazard Detection in Digital MOS Circuits." *Proc. of the Ist Int. Conf. on Microelectro, Opto, Mechanic Systems and Components,* Berlin, Sept. 1990.

[583] G. Saucier, P. Sicard, L. Bouchet, "Multi-Level Synthesis on PALs," *Proc. of the European Conference on Design Automation,* Glasgow, pp. 542-546, March, 1990.

[584] H. Sawada, T. Suyama, and A. Nagoya, "Logic Synthesis for Look-Up Table Based FPGAs Using Functional Decomposition and Support Minimization," *Proc. 1995 ICCAD*, pp. 353–358. 1995.

[585] P. Sawkar, D. Thomas, "Area and Delay Mapping for Table-Lookup Based Field Programmable Gate Arrays," *Proc. of 29th ACM/IEEE Design Automation Conf.,* pp. 368-373, 1992.

[586] I. Schaefer Ph.D. Thesis, Portland State University, 1992.

[587] I. Schaefer, M.A. Perkowski, "Multiple-Valued Input Generalized Reed-Muller Expansions", *IEE Proceedings, Pt.E,* Vol. 139, No. 6, pp. 519-527, Nov. 1992.

[588] I. Schaefer, B.J. Falkowski, M.A. Perkowski, "Generation of Adding and Arithmetic Multi-Polarity Transforms for Incompletely Specified Boolean Functions", *Intern. J. of Electr.*, Vol. 73, No. 2, pp. 321-331, 1992.

[589] I. Schaefer, M. Perkowski, "Extended Spectral Techniques for Logic Synthesis", Submitted book chapter, *"Formal Methods in Spectral Analysis",* prof. Radomir Stankovic, Editor, 1992.

[590] I. Schaefer, M. A. Perkowski, and H. Wu, "Multilevel Logic Synthesis for Cellular FPGAs Based on Orthogonal Expansions," *Proceedings of the IFIP WG 10.5 Workshop on Applications of the Reed-Muller Expansion in Circuit Design,* Hamburg, Germany, September 1993.

[591] I. Schaefer, M.A. Perkowski, "Synthesis of Multi-Level Multiplexer Circuits for Incompletely Specified Multi-Output Boolean Functions with Mapping Multiplexer Based FPGAs", *IEEE Trans. on Computer Aided Design,* Vol. 12, No. 11, Nov. 1993, pp. 1655 - 1664.

[592] I. Schaefer, M. Perkowski, "On the Minimal Multiple-Valued Input Kronecker Reed-Muller Form for Incompletely Specified Multiple-Valued Input Functions", *Submitted to IEEE Trans. on Comput.* , 1994.

[593] R. Scheuring, H. Wehlan, "On the Design of Discrete Event Dynamic Systems by Means of the Boolean Differential Calculus," *Prepr. of the IFAC Symposium: Design Methods of Control Systems,* Zuerich 1991, (in German).

[594] U. Schlichtmann, "Boolean Matching and Disjoint Decomposition for FPGA Technology Mapping, *IFIP Workshop on Logic and Architecture Synthesis,* pp. 83–102, 1993.

[595] Schmidt, D., C., Druffel, L.E., "An Extension of the Clause-Table Approach to Multi-Output Combinatorial Switching Networks," *IEEE TC,* Vol. C-23, April 1974.

[596] P.R. Schneider, D.L. Dietmeyer, "An Algorithm for Synthesis of Multiple-Output Combinational Logic," *IEE Trans. on Comput.*, pp. 117-128, Febr. 1968.

[597]) Schneider, new 1994 paper about improving power consumption by decompositional methods.

[598] E. Schubert, U. Kebschull, and W. Rosenstiel, "FDD Based Technology Mapping for FPGA," *Proc. EURO-ASIC*, pp. 14-18, Paris, France, June 1992.

[599] G.W. Schwede, and A. Kandel, "Fuzzy maps", *IEEE Trans. Syst. Man Cybernet.,* Vol. 7, No. 9, pp. 699-674, 1977.

[600] S.M. Selkow, "New bounds for the clique number of a graph," *Inf. Proc. Letters,* Vol. 7., No. 4., pp. 170–172, June 1978.

[601] H. Selvaraj, A. Czerczak, A. Krasniewski, T. Luba, "A Generalized Decomposition of Boolean Functions and its Application in FPGA-Based Synthesis," *IFIP Workshop on Logic and Architecture Synthesis,* pp. 147-166, Grenoble 1993.

[602] H. Selvaraj, "FPGA-Based Logic Synthesis," *Ph.D. Dissertation,* Warsaw University of Technology, 1994.

[603] W.L. Semon, "Characteristic Numbers and Their Use in the Decomposition of Switching Functions," *Proc. ACM,* Vol. 17, pp. 273-280, May 1952. (In Curtis'62).

[604] C.E. Shannon, "A symbolic analysis of relay and switching circuits," *Trans. AIEE 57,* pp. 713-723, 1938.

[605] C.E. Shannon, "The Synthesis of Two-Terminal Switching Circuits," *The Bell System Technical Journal,* 28, pp. 59-98, 1949.

[606] G. Piatesky-Shapiro, Ch. Matheus, P. Smyth, and R. Uthurusamy, "KDD-93: Progress and Challenges in Knowledge Discovery in Databases," *Artificial Intelligence Magazine,* Vol. 15, No. 3, pp. 77-82, Fall 1994.

[607] V.Y. Shen, "On Simple Disjunctive Decompositions of Switching Functions," Ph.D. thesis, *Dept. Electr. Engng., Princeton University,* N.J., January 1969.

[608] V.Yun-Shen, and A. C. McKellar, "An Algorithm for the Disjunctive Decomposition of Switching Functions," *IEEE Trans. on Comput.*, Vol. C-19, pp. 239-248, March 1970.

[609] V.Y. Shen, A. C. McKellar, and P. Weiner, "An Fast Algorithm for the Disjunctive Decomposition of Switching Functions," *IEEE Trans. on Campput.*, Vol. C-20, No. 3, pp. 304-309, March 1971.

[610] V.Z. Shen, J.D. Huang, and S.M. Chao, "Lambda Set Selection in Roth-Karp Decomposition for LUT-Based FPGA Technology Mapping," *Proc. 32 DAC*, June 1995, pp. 65–69.

[611] A.A. Shneider, *Kibernetika,* No.4, pp. 15-22, 1984, (in Russian).

[612] B.N. Shneyder, "Algebra of Sets Applied to Solution of Graph Theory Problems," *Engineering Cybernetics*, Plenum Press p. 521, 1970.

[613] M.I. Shvarcman, *Avtomatika i Vycislitelnaya Tekhnika,* No. 6, pp. 12-17, 1981, (in Russian).

[614] P. Sicard, M. Crastes, K. Sakouti, G. Saucier, "Automatic Synthesis of Boolean Functions on Xilinx and Actel Programmable Devices," *Proc. Euro ASIC '91,* pp. 142-145, 1991.

[615] H.A. Simon, A. Ando, "Aggregation of Variables in Dynamic Systems," *Econometrica,* 29, pp. 111-138, 1961.

[616] H.A. Simon, "The Sciences of the Artificial," Cambridge, MA, *MIT Press,* 1981.

[617] T. Singer, "The Decomposition Chart as a Theoretical Aid", *Harvard Computational Lab.,* Cambridge, Mass., Rept. BL-4, 1953.

[618] P.K. Sinha Roy, and C.L. Sheng, "A Decomposition Method of Determining Maximum Compatibles," *IEEE Trans. Computers*, Vol. C-21, pp. 309–312, 1972.

[619] J. Sklansky, "General Synthesis of Tributary Switching Networks," *IEEE Trans. on Electr. Comput.* Vol. EC -12, pp. 464-469, 1963.

[620] A. Skowron, C. Rauszer, "The Discernibility Matrices and Functions in Information Systems," *Res. Rep. 1/91,* Inst. Comp. Sci., Warsaw, 1991.

[621] Slagle, J.R., C.L. Chang, R.C.T. Lee, "A New Algorithm for generating Prime Implicants," IEEE TEC, Vol.EC-19, pp.304-311 April 70.

[622] N. Song and M. A. Perkowski, "EXORCISM-MV-2: Minimization of Exclusive Sum of Products Expressions for Multiple-Valued Input Incompletely Specified Functions," *Proc. ISMVL '93,* pp. 132-137, Sacramento, CA, May 24-27 1993.

[623] N. Song, and M.A. Perkowski, "Minimization of Exclusive Sum of Products Expressions for Multi-Output Multiple-Valued Input Switching Functions," *accepted to IEEE Trans. on CAD.*

[624] B.L. Sorokin, *Avtomatika i Vycislitelnaya Tehnika,*, No. 4, pp. 50-55, 1982, (in Russian).

[625] M. Stankovic, Z. Tosic, S. Nikolic, "Synthesis of Maitra Cascades by Means of Spectral Coefficients," *IEE Proc.* Vol. 130, Pt. E, No. 4, pp. 101-108, July 1983.

[626] R.S. Stankovic, "Matrix Relations for Symmetry Properties of Boolean Function Detection," *Proc. of XXXIII Yugoslav Conference on ETAN*, Novi Sad, (in Serbian), 1989.

[627] R.S. Stankovic, C. Moraga, M. Stankovic, K. Rangelov, "Simple Disjoint Decomposition of Multiple-Valued Functions," Research Reports in Applied Mathematics, Series: Multiple-Valued Logic, YU Report No. 4.

[628] R.S. Stankovic, N. Denic, "An Approach to the Detection of Symmetry and Co-Symmetry Properties of Switching Functions," Research Reports in Applied Mathematics, Series: Multiple-Valued Logic, Report No. 5.

[629] R.S. Stankovic, and C. Moraga, "Methods for Detection of Some Properties of Multiple-Valued Functions," *IEE Proceedings E,* Vol. 139, No. 5, Sept. 1992, pp. 421-429.

[630] S.M. Starobinets, "On an Algorithm for Finding the Greatest Internally Stable Sets of a Graph," *Engineering Cybernetics '73.*, Plenum Press, p. 873., 1973.

[631] B. Steinbach, "Theorie, Algorithmen und Programme fuer den Rechnergestaetzten Logischen Entwurf Digitaler Systeme," *Habilitation,* T.H. Karl-Marx-Stadt/Chemnitz 1983, (in German).

[632] B. Steinbach, T.Q. Le, "Tools fuer den Logikentwurf," *Wiss. Schriftenreihe der T.U. Karl-Marx-Stadt (Chemnitz)*, No. 9, 1988, (in German).

[633] B. Steinbach, R. Hilbert, "Schnelle Testdatengenerierung' Gestuetzt auf den Booleschen Differentialkalkuel," in: *Bochmann, D.; Uhar, R.: Fehler in Automaten. Verlag Technik,* Berlin, 1989, (in German).

[634] B. Steinbach, J. Fehmel, "Ausnutzung von Funktionseigenschaften beim Dekompositorischen Logikentwurf," *Proc. of 3rd "Tagung Schaltkreisentwurf"*, Tagungsmaterialien, Dresden, 1989, (in German)

[635] B. Steinbach, T.Q. Le, "Entwurf testbarer Schaltungen," *Wiss. Schriftenreihe der TU Chemnitz,* No. 12, 1990.

[636] B. Steinbach, "XBOOLE - A Toolbox for Modeling, Simulation and Analysis of Large Digital Systems," *System Analysis and Modelling, Gordon and Breach Science Publishers,* Vol. 9, No. 4, pp. 297-312, 1992.

[637] B. Steinbach, F. Schumann, M. Stoeckert, "Functional Decommposition of Speed Optimized Circuits," In *Auvergne, D., Hartenstein, R. (ed) "Power and Timing Modeling for Performance of Integrated Circuits,"* Proc. of the Third Intern, Workshop on Power and Timing Modeling and Optimization, IT Press Verlag, Bruchsal, 1993.

[638] B. Steinbach, M. Stoeckert, "Design of Fully Testable Circuits by Functional Decomposition and Implicit Test Pattern Generation," *Proc. 1994 IEEE Test Conference,* 1994.

[639] B. Steinbach, and A. Wereszczynski, "Synthesis of Multi-Level Circuits Using EXOR-Gates," *Proc. Reed-Muller'95 Workshop,* Chiba, Japan, August 1995, pp. 161-168.

[640] K.E. Stoffers, "Partitioning of Separating Edges - New Approach to Combinational Logic Design," *IEEE Trans. on Comput.*, Vol. 26, No. 8, pp. 833-836, 1977.

[641] , I. Stojmenovic, M. Miyakawa, and R. Tosic, "On Spectra of Many-Valued Logic Symmetric Functions," *Proc. 18th ISMVL*, Palma De Mallorca, pp 285-292, May 24-26, 1988.

[642] S.Y.H. Su, and P.T. Cheung, "Computer Minimization of Multivalued Switching Functions," *IEEE TC*, Vol. 21, No. 9, pp. 995-1003, Sept 1972.

[643] B.A. Subbotovskaya, "Comparison of Bases in the Realization of Boolean Functions by Formulas," DAN SSSR, 149, No. 4, 1963.

[644] A. Suzuki, *Electr. Co. J.,* 52, pp. 121-?, 1969.

[645] A. Suzuki, *Electr. Co. J.,* 52, pp. 132-?, 1969.

[646] Svoboda, A., "The Concept of Term Exclusiveness and Its Effect on the Theory of Boolean Functionss," Journ. of the Assoc. for Comp. Mach., Vol. 22, No. 3, July 1975, pp. 425-440.

[647] *Synopsys Design Compiler V 1.3h.* Manual 1991.

[648] M.M. Syslo, N.Deo, and J.S. Kowalik, "Discrete Optimization Algorithms with Pascal Programs," *Prentice Hall,* Englewood Cliffs, N.J., 1983.

[649] P. Szolgay, "On Algorithms in the Parallel design of Logic and Layout of Circuits with Functional Blocks", *Intern. J. of Circuit Theory and Applic.*, Vol.20, No.4, pp.411-429, 1992. TK454 .I58

[650] T.F. Tabloski, F. J. Mowle, "A Numerical Expansion Technique and Its Application to Minimal Multiplexer Logic Circuits," *IEEE Trans. on Comput.,* Vol. 25., No. 7, pp. 684-702, 1976.

[651] R.E. Tarjan, and A.E. Trojanowski, "Finding a maximum independent set," SIAM J. Comput., Vol. 6, pp. 537–46, 1977.

[652] A. Tehrani, "Un Algorithme de Coloration," *Cahiers Centre Etudes Recherche Oper.,* Vol. 17, pp. 395-398, 1975.

[653] A.K. Teslenko, *Vestnik Kijev politehniceskogo Instituta, seria avtomatiki i elektropriborostroenija,* coll. 12, pp. 59-61, 1975, (in Russian).

[654] A. Thayse, *Phil. Res. R.* Vol. 26, pp. 229, 1971.

[655] A. Thayse, "A Fast Algorithm for the Proper Decomposition of Boolean Functions," *Philips Res. Reports,* No. 27, 1972.

[656] A. Thayse, and M. Davio, "Boolean Differential Calculus and its Application to Switching Theory," *IEEE Trans. on Comput.*, Vol. C-22, No. 4, pp. 409-420, 1973.

[657] A. Thayse, "Boolean Differential Calculus and its Application to Switching Theory," *IEEE Trans. on Comput.*, Vol. C-22, April 1974.

[658] A. Thayse, "Boolean Differential Calculus," *Springer-Verlag,* Heidelberg, 1981.

[659] S. Thelliez, "Sur la Decomposition Disjonctive Complexe Arborescent d'une Fonction Ternaire en vue de son Application a la Synthese des Structures Combinatoires Ternaires," *Compt. Rend. H. Acad. Sci.,* Vol. 264, ser. A, pp. 419-421, February 1967, (in French).

[660] S. Thelliez, "Introduction to the Study of Ternary Switching Structures," *Gordon and Breach Science,* N.Y. 1973.

[661] R. Thiele, "Synthese von Verknuepfungsnetzen aus Universellen Logikbausteinen," *Dissertation,* Darmstadt 1978, (in German).

[662] P. Tison, "Generalization of Consensus Theory and Application to the Minimization of Boolean Functions," *IEEE TC*, Vol. EC-16, Aug. 1967, pp. 446–456.

[663] S. B. Thrun and et. al., "The Monk's Problems - A Performance Comparison of Different Learning Algorithms", Carnegie Mellon University, Dec. 1991.

[664] V.H. Tokmen, "Disjoint Decomposibility of Multi-Valued Functions by Spectral Means," *Proc. IEEE 10th International Symp. on Multiple Valued Logic,* pp. 88-93, 1980.

[665] I. Tomescu, "The maximum number of cliques and recoveries by cliques of complete chromatic hypergraphs," *Discrete Mathematics*, Vol. 37, No. 2,3, pp. 263–277, Dec. 1981.

[666] I. Tomescu, "Some properties of irreducible coverings by cliques of complete multipartite graphs," *J. Comb. Theory,* Sec. B 27, No. 2., pp. 164–167, April 1980.

[667] A.J. Tosser, D. Aoulad-Syad, "Cascade Networks of Logic Functions Built in Multiplexer Units," *IEE Proc. Pt. E*, Vol. 127, No. 2, pp. 64-68, March 1980.

[668] L. Trevillyan, "An Overview of Logic Synthesis Systems," *Proc. of the 24th ACM/IEEE Design Automation Conference (DAC),* 1987.

[669] A.C. Tucker, and L. Bodin, "A Model for Municipal Street Sweeping Operations," *Case Studies of Applied Mathematics,* Math. Assoc. of America, Washington, pp. 251–295, 1976.

[670] Chia-Jeng Tseng, and D.P. Siewiorek, "Facet: A Procedure for the Automated Synthesis of Digital Systems," *Proc. 20-th Design Automation Conference,* p.566.

[671] S. Tsukijama, M. Ide, H. Ariyoshi, and I. Shirakawa, "A New Algorithm for Generating All the Maximal Sets," *SIAM J. of Computing*, Vol. 6, No. 3., pp. 505–517, September 1977.

[672] G.B., Tumanyan, *Engng. Cybern.* 156, 525, 1964, (in Russian), TJ212 .A413

[673] G.B. Tumanyan, *Engng. Cybern.* 74, 1965, (in Russian). TJ212 .A413 5

[674] Tutte, W.T., (B. Descartes) "Solution of Advanced Problem No. 4526", American Math. Monthly, 61., p.352.

[675] P. Uehlau, "Eine Dekompositionsstrategie fuer den Logikentwurf auf der Basis Funktionstypischer Eigenschaften," *Dissertation,* T.U. Karl-Marx-Stadt (Chemnitz), 1987, (in German).

[676] K. Walczak, "Decompositional Method of Combinational Circuits Synthesis Free from Static Hazard for Adjacent Changes," *Archiwum Automatyki i Telemechaniki,* No. 1-2, 1977.

[677] K.M. Walliuzzaman, Z.G. Vranesic, "On Decomposition of Multiple-Valued Switching Functions," *Computer Journal,* Vol. 13, pp. 359-362, 1970.

[678] W. Wan, M.A. Perkowski, "A New Approach to the Decomposition of Incompletely Specified Functions based on Graph-Coloring and Local Transformations and Its Application to FPGA Mapping", *Proc. of the IEEE EURO-DAC '92, European Design Automation Conference,* Sept. 7-10, Hamburg, 1992, pp. 230 - 235.

[679] F.L. Wang, "Isomorphic Notation of Switching Circuits," IEEE Comp., 5., Dec. 1965.

[680] C.C. Wang, "An Algorithm for the Chromatic Number of a Graph," *JACM,* Vol. 21, pp. 385-391, 1974.

[681] Y. Watanabe, and R.K. Brayton, "Heuristic Minimization of Multiple-Valued Relations," *IEEE Trans. on CAD.*, Vol. 12, No. 10, pp. 1458-1472, October 1993.

[682] T. Wecker, R. Kumar, W. Rosenstiel, H. Kraemer, M. Neher, "CALLAS - ein System zur Automatischen Synthese Digitaler Schaltungen," *Informatik - Forschung und Entwicklung,* April 1989, (in German).

[683] D.J.A. Welsh, and M.B Powell, "An Upper Bound to the Chromatic Number of a Graph and its Application to Time-Table Problems," *Comput. J.* Vol. 10, pp. 85–86, 1967.

[684] C.D. Weiss, "The Characterization and Properties of Cascade Realizable Switching Functions," *IEEE Trans. on Comput.* pp. 624-633, July 1969.

[685] C.D. Weiss, "Optimal Synthesis of Arbitrary Switching Functions with Regular Arrays of 2-input 1-output Switching Elements", *IEEE Trans. on Comput.*, pp. 839-856, September, 1969.

[686] S.H. Whitesides, "An algorithm for finding clique cut-sets," *Inf. Process. Letters*, Vol. 12., No. 1., pp. 31–32, Febr. 1981.

[687] H.J. Wilf, "Spectral bounds for the cliques and independence numbers of graphs," *Comb. Theory Series B,* Vol. 40, No. 1, pp. 113-117, Febr. 1986.

[688] M.R. Williams, "The Coloring of Very Large Graphs," *Combinatorial Structures and Their Applications,* Gordon and Breach, New York, pp. 477–478, 1970.

[689] J. Wnek, and R.S. Michalski, "Hypothesis-driven Construction Induction in AQ17-HCI," *Machine Learning and Inference Laboratory, Center for Artificial Intelligence, School of Information Science and Technology, George Mason University,* Jan. 1992, Technical Report.

[690] W.S. Wojciechowski, A.S. Wojcik, "Automated Design of Multiple-Valued Logic Circuits by Automatic Theorem-Proving Techniques," *IEEE Trans. on Comput.*, Vol. 32, No. 9, pp. 785-798, 1983.

[691] Wojutynski, J., "Graph Coloring Algorithms," M.Sc. Thesis, Institute of Automatics, Warsaw Technical University, 1979 (in Polish).

[692] Nam-Sung Woo, "A Study on the Structure of the Intermediate Network in an FPGA Technology Mapping", *Proc. Field Programmable Logic and Applications,* Oxford, 1991.

[693] Nam-Sung Woo, "A Heuristic Method for FPGA Technology Mapping Based on the Edge Visibility," *Proc. 28th ACM/IEEE Design Automation Conf.* San Francisco, CA, June 1991, pp. 248-251.

[694] D.C. Wood, "A Technique for Coloring a Graph Applicable to Large-Scale Time-Tabling Problems," *Comp. J.,* Vol. 12, pp. 317-319, 1969.

[695] L-F. Wu, M.A. Perkowski, "Minimization of Permuted Reed-Muller Trees for Cellular Logic Programmable Gate Arrays", *Proc. of the 2nd Intern. Workshop on Field-Programmable Logic and Applications, FPL'92,* Vienna, Austria, August 31-September 2, 1992, pp. 7/4.1-7/4.4.

[696] L-F, Wu, and M. Perkowski, "Minimization of Permuted Reed-Muller Trees for Cellular Logic Programmable Gate Arrays," *invited chapter to a book edited by Prof. Reiner Hartenstein,* 1992, *Springer Verlag,* No. 705, pp. 78 - 87, 1993.

[697] H. Wu, and M. A. Perkowski, "Synthesis for Reed-Muller Directed-Acyclic-Graph networks with applications to Binary Decision Diagrams and Fine Grain FPGA Mapping", *Proc. of IWLS '93,* Tahoe City, CA, May 1993.

[698] H. Wu, N. Zhuang, and M. A. Perkowski, "Synthesis for Reed-Muller Directed-Acyclic-Graph network," accepted to *IEE Proceedings, Pt. E.,* in June 1993.

[699] H. Wu, M. A. Perkowski, and N. Zhuang, "A New Easily Testable Canonical AND/XOR Circuit Form and its Minimization," submitted to *IEEE Trans. on Comput.*, June 1994.

[700] P. Vanoostende, "Power estimation," *Internal Report,* IMEC Leuven, April 1991.

[701] D. Varma, E.A. Trachtenberg, "Design Automation Tools for Efficient Implementation of Logic Functions by Decomposition," *IEEE Trans. on CAD,* Vol. 8, No. 8, pp. 901-917, August 1989.

[702] D. Varma, E.A. Trachtenberg, "On the Estimation of Logic Complexity for Design Automation Applications," *1990 Conference of IEEE*, pp. 368-371, 1991.

[703] V.B. Varshavsky, *Engng. Cybern.* R 58, 1965, (in Russian). TJ212 .A413 5

[704] V.B. Varshavsky, Ovsievich, "Networks Composed of Ternary Majority Elements," October, *IEEE Comput.* 14, 730, 1965.

[705] V.B. Varshavsky, *Engng. Cybern.* R 39, 1965, (in Russian).

[706] Vashchenko, "A Method of Synthesizing a Class of Schemes on the Basis of Functional Decomposition," *Kibernetika,* No. 1, 1969.

[707] V.P. Vascenko, *Tr. Mosk. Energ. Instituta,* vol. 247, pp. 5-10, 1975, (in Russian)

[708] V.P. Vascenko, *Dokl. AN SSSR,* Vol. 234, No. 3, pp. 509-512, 1977, (in Russian).

[709] V.P. Vascenko, *Dokl. AN SSSR,* Vol. 238, No. 1, pp. 18-21, 1978, (in Russian).

[710] V.P. Vascenko, "On the Computation of all Nontrivial Simple Decompositions", Vol.20, No.4, pp.629-632, 1979, (in English). V.P. Vascenko, *Dokl. AN SSSR,* Vol. 247, No. 1, pp 15-18, 1979, (in Russian). QA1 .S78 Mathematics

[711] J. Vasudevamurthy, J. Rajski, "A Method for Concurrent Decomposition and Factorization of Boolean Expressions," *Proc. IEEE ICCAD,* Vol. 7, No. 12, December 1988, pp. 1290-1300.

[712] E.W. Veitch, "A chart method for simplifying truth functions", *Proc. ACM, Pittsburg, PA*, pp. 127-133, May 1952.

[713] Voigt

[714] E. Voigt, "Anwendung einer Ternaervektorlisten-Algorithmensprache auf die Struktursynthese von GATE-ARRAY-Schaltkreisen," *Dissertation,* T.H. Karl-Marx-Stadt (Chemnitz), 1985, (in German).

[715] Voith,R.P. "ULM Implicants for Minimization of Universal Logic Module Circuits," *IEEE Trans. on Comput.*, Vol. C-26, WHEN?????

[716] R.P. Voith, "Minimum Universal Logic Module Sequential Circuits with Decoders," *IEEE Trans. on Comput.*, Vol. C-26, No. 10, 1977.

[717] Xilinx Inc., "The Programmable Gate Array Data Book", "Xilinx Programmable Gate Array User's Guide," 1991.

[718] S.W. Yablonskii, "On Algorithmic Obstacles to the Synthesis of Minimal Contact Networks," *Problemy Kibernitiki,* No. 2, pp. 75-121, 1959, (in Russian).

[719] K. Yamamoto, "Design of Irredundant MOS - Networks: A Program Manual for the Design Algorithm DIMN," *Dept. of Computer Science,* University of Illinois, February 1976.

[720] Chao-Chih Yang, "On the Equivalence of Two Algorithms for Finding All Maximal Compatibles," *IEEE Trans. Comp.*, pp. 977–979, Oct. 1975.

[721] S. Yang, M. Ciesielski, "A Generalized PLA Decomposition With Programmable Encoders," In the *Proc. of the Intern. Workshop on Logic Synthesis,* May 1989, pp. 1-13.

[722] S. Yang, M. Ciesielski, "PLA Decomposition with Generalized Decoders," *Proc. of the ICCAD,* pp. 312-315, 1989.

[723] S. Yang and M.J. Ciesielski, "On The Relationship Between Input Encoding and Logic Minimization," *Proc. 23rd Hawaii International Conference on System Sciences*, pp. 377-386, Jan. 1990.

[724] S. Yang, M. Ciesielski, "Optimum and Suboptimum Algorithms for Input Encoding and its Relationship to Logic Minimization," *IEEE Transactions on Computer Aided Design,* Vol. 10, No. 1, January 1991.

[725] S.S. Yau, C.K. Tang, "Universal Logic Circuits and Their Modular Realizations," *1966 Spring Joint Comp. Conf.*, AFIPS Conf. Proc. Vol. 28, Washington DC, pp. 297-305, 1966.

[726] S.S. Yau, and M. Orsic, "Synthesis of Universal Logic Modules," *Proc. 3rd Annual Princeton Conf. Inform. Sci. and Syst.*, pp. 498-502, March, 1969.

[727] S.S. Yau, C.K. Tang, "Universal Logic Modules and Their Applications," *IEEE Trans. on Comput.*, Vol. C-19, No. 2, Febr. 1970.

[728] S.S. Yau, Y.S. Tang, "On Identification of Redundancy and Symmetry of Switching Functions," *IEEE Trans. on Comput.*, Vol. C-20, pp. 1609-1613, Dec., 1971.

[729] S.V. Yenin, P.N. Bibilo, *Avtomatizacjia Proektirovanija EVM,* pp. 55-65, Kiev 1979, (in Russian).

[730] S.V. Yenin, P.N. Bibilo, *Avtomatika i vycislitelnaya tekhnika,* No. 1, pp. 16-22, 1979, (in Russian).

[731] A.A. Ytkin, *Avtomatizacjia Logiceskogo Proektirovanija,* pp. 41-58, Minsk 1982, (in Russian)

[732] L.A. Zadeh, "A Fuzzy-Set-Theoretic Interpretation of Liguistic Hedges", *Journal of Cybernetics,* Vol.2, No.4, pp.4-34, 1972.

[733] A.D. Zakrevskij, "An Algorithm for Decomposition of Boolean Functions," *Trudy Sib. Fiz-tekhn In-ta*, Collection 44, pp. 5-16, 1964, (in Russian)

[734] 39. A.D. Zakrevskij, "Algoritmy Sinteza Diskretnych Avtomatov", Moscow 1971, (in Russian)

[735] A.D. Zakrevskij, "Logicheskije yravnenija," Minsk, 1975, (in Russian)

[736] A. D. Zakrevskij, *Dokl. A.N. BSSR,* Vol. 21, No. 11, pp. 991-994, 1977, (in Russian).

[737] A.D. Zakrevskij, "Logiceskij Sintez Kaskadnych Schem," Moscow, 1981, (in Russian).

[738] A.D. Zakrevskij, P.N. Bibilo, A. A. Dudkin, A.A. Shneider, "Upravlajuscije Sistemy i Masiny, No.3, pp. 115-118, 1984, (in Russian).

[739] A.D. Zakrevskij, P.N. Bibilo, A. A. Dudkin, *Upravlajuscije sistemy i masiny,* 1985, No. 1, pp. 27-29, (in Russian).

[740] Zaky, Oliviera,

[741] H.J. Zander, "Logischer Entwurf binaerer Systeme," *VEB Verlag Technik,* Berlin 1989, (in German).

[742] I.I. Zhegalkin, "O Tekhnyke Vychyslenyi Predlozhenyi v Symbolytscheskoi Logykye," *Mat. Sb.,* Vol. 34, pp. 9-28, 1927 (in Russian).

[743] K. Zibert, "Entwurfstechnik fuer IS - Versuch einer Bestandsaufnahme," ITG-Fachbericht der ITG-Fachtagung vom 3.-5.10.1989, Stuttgart, *VDE-Verlag GmbH,* Berlin Offenbach, (in German).

[744] A.A. Zykov, "On Some Properties of Linear Complexes," *Math. Sbornik,* 24/26, p. 163., Amer. Math, Soc. Translation No. 79.