Portland State University

PDXScholar

Dissertations and Theses

Dissertations and Theses

12-11-1995

Minimization of Sum-of-Conditional-Decoders Structures with Applications in Finite Machine EPLD Design and Machine Learning

Sanof Mohamedsadakathulla
Portland State University

Follow this and additional works at: https://pdxscholar.library.pdx.edu/open_access_etds

Part of the [Electrical and Computer Engineering Commons](#)

Let us know how access to this document benefits you.

Recommended Citation

Mohamedsadakathulla, Sanof, "Minimization of Sum-of-Conditional-Decoders Structures with Applications in Finite Machine EPLD Design and Machine Learning" (1995). *Dissertations and Theses*. Paper 5158.

<https://doi.org/10.15760/etd.7034>

This Thesis is brought to you for free and open access. It has been accepted for inclusion in Dissertations and Theses by an authorized administrator of PDXScholar. Please contact us if we can make this document more accessible: pdxscholar@pdx.edu.

THESIS APPROVAL

The abstract and thesis of Sanof Mohamedsadakathulla for the Master of Science in Electrical Engineering were presented December 11, 1995, and accepted by the thesis committee and the department.

COMMITTEE APPROVALS:

Marek A. Perkowski, Chair

Malgorzata E. Chrzanowska-Jeske

Maria Balogh
Representative of the Office of Graduate Studies

DEPARTMENT APPROVAL:

Rolf Schaumann, Chair
Department of Electrical Engineering

ACCEPTED FOR PORTLAND STATE UNIVERSITY BY THE LIBRARY

by

on 21 March 1996

ABSTRACT

An abstract of the thesis of Sanof Mohamedsadakathulla for the Master of Science in Electrical Engineering presented December 11, 1995.

Title: Minimization of SUM-OF-CONDITIONAL-DECODERS Structures with Applications in Finite State Machine EPLD Design and Machine Learning

In order to achieve superior speed in sequencer designs over competing PLD devices, Cypress brought to market an innovative architecture, CY7C361. This architecture introduced a new kind of universal logic gate, the CONDITION DECODER (CDEC). Because there are only 32 macrocells in the chip, saving only one CDEC gate can be quite important (the well-known "fit/no-fit problem"). A problem that is related to the fitting problem of the Cypress CY7C361 chip is the SOC Minimization. Due to the limited low number of macrocells in CY7C361, a high quality logic minimization to reduce the number of macrocells is very important. The goal of this thesis is, however, more general, since we believe that CDEC can be used as a general-purpose gate for standard cell structures with few levels, and also for new PLD structures. We depart, therefore, from the Cypress chip as a sole motivation of our work, and we present a generic logic synthesis problem of SOC minimization.

In this thesis, we formulate the SOC minimization problem and present a new kind of approach using graph coloring to solve it. A Cube Splitting algorithm is also presented, whereby the input cubes are split in such a way, that the generated cubes are lower in number than the minterms, and when these cubes are used as nodes in graph

coloring algorithm, gives near exact solutions. The algorithms used in the SOC minimization program, SOCMIN, have been designed for Strongly Unspecified functions, defined by ON and OFF sets, and hence finds important applications for Machine Learning and Pattern theory, where there is a high percent of don't cares.

The approach to solving the covering problem, the Conditional Graph Coloring, can be used in other similar problems such as PLA minimization, or Column Minimization Problem in Curtis-like decomposition of Multi-valued Relations. We found also the Muller method very efficient for ON,OFF data representation: it can be used to extend any other single-output minimizer for incomplete functions to a multi-output one.

MINIMIZATION OF SUM-OF-CONDITIONAL-DECODERS STRUCTURES
WITH APPLICATIONS IN FINITE STATE MACHINE EPLD
DESIGN AND MACHINE LEARNING

by

SANOF MOHAMEDSADAKATHULLA

A thesis submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE
in
ELECTRICAL ENGINEERING

Portland State University
1996

ACKNOWLEDGMENT

I would like to thank Dr. Marek A. Perkowski, my advisor, who introduced me to the area of Design Automation, guided me, and encouraged me in my research. I would also like to thank Dr. Malgorzata E. Chrzanowska-Jeske and Dr. Maria Balogh for their valuable comments which helped to improve this thesis.

I would like to thank my dear parents for the overwhelming support and encouragement and all my friends for their help and support. Also, I would like to thank Ms. Shirley Clark and staff of the EE Department office for all the assistance.

TABLE OF CONTENTS

	PAGE
ACKNOWLEDGEMENTS	ii
LIST OF TABLES	vi
LIST OF FIGURES	vii
CHAPTER	
I MOTIVATION FOR THIS RESEARCH	1
II CUBE REPRESENTATION AND OPERATIONS	5
II.1 Cubical Representation	5
II.2 Cube Operations	9
III THE CYPRESS CY7C361 EPLD AND MACHINE LEARNING . . .	15
III.1 The Cypress CY7C361 EPLD	15
III.1.1 The Architecture of the CY7C361	15
III.1.2 Applications and advantages of Cypress CY7C361 . .	17
III.2 Machine Learning and KDD	18
III.2.1 Basic research idea of the PTG and FDG	19
III.2.2 Decomposed Function Cardinality	21
III.2.3 SOC minimization for Machine Learning	22
IV CDEC GATE AND CDEC IMPLICANTS	24
IV.1 The Condition Decoder gate	24
IV.2 CDEC Implicants	27
IV.2.1 Definitions	27

	iv
IV.2.2 CDEC Implicants	28
V FORMULATION OF THE SOC MINIMIZATION PROBLEM	31
VI THE CONDITIONAL GRAPH COLORING	36
VI.1 Supercube Operation	36
VI.2 Conditionally Compatible Cubes	37
VI.3 Conditional Graph Coloring	39
VI.3.1 Definitions	44
VI.3.2 The Conditional Graph Coloring Algorithm	48
VI.4 Conditional Graph Multi-Coloring Algorithm	49
VI.4.1 Algorithm	49
VII SPLITTING OF CUBES	60
VII.1 Initial-Split Cube Operation	62
VII.2 Definitions	63
VII.3 Algorithm to create Split Cubes	63
VIII MULTIPLE OUTPUT AND INCOMPLETELY SPECIFIED FUNC- TIONS	69
VIII.1 Multi-output Functions	69
VIII.2 Incompletely Specified Function	73
IX RESULTS AND COMPARISON	74
IX.1 Single Output MCNC benchmarks.	74
IX.2 Multiple Output MCNC benchmarks.	80
IX.3 Delay of CDEC	81
IX.4 Machine Learning examples	85
IX.4.1 DFC for SOC	85
IX.4.2 Results for Machine Learning examples	89

IX.4.3	Machine Learning examples with randomly gener-	
	ated data	90
X	CONCLUSION	95
	REFERENCES	97

LIST OF TABLES

TABLE		PAGE
I	Example of binary input binary output function	6
II	Literal Encoding	37
III	Machine Learning examples with data randomly removed	90
IV	Machine Learning examples with data randomly removed	91

LIST OF FIGURES

FIGURE	PAGE
1 ON and OFF array.	7
2 Karnaugh map for 3 variables.	8
3 Karnaugh map for binary input cubes.	9
4 Example for Sharp operation.	12
5 Cube operations.	13
6 Architecture of the <i>CY7C361</i>	17
7 CDEC gate	25
8 Boolean function f as a SOSC.	25
9 Realization of SOSC forms with the <i>CY7C361</i>	26
10 Karnaugh map for h implies f	28
11 Karnaugh map for prime implicants	29
12 Karnaugh map for prime implicants	29
13 CDEC implicant.	30
14 Approaches to the Minimization using <i>CDEC</i> Gates.	32
15 Supercube of two cubes C_i and C_j	37
16 Examples of (conditionally) compatible cubes.	40
17 <i>CDEC</i> implicants and graph GCCC.	42
18 Function to demonstrate that the minimum SOC cannot be obtained by factorization of primes	43
19 Color-Restriction-Vector of a node	44

20	Color-Restriction-Matrix for a node	45
21	Implicants and graph GCCC.	50
22	Entries made in each Pass of the algorithm	51
23	Restriction-Vector-Matrix for node 4 during Pass VI	52
24	CDEC Implicants for Conditional Graph Coloring and multi-coloring algorithm	59
25	Example for split	61
26	Example for Initial Split.	63
27	Initial Split for several possible combinations of occurrence of input cubes.	64
28	Example for Cube Split Algorithm.	65
29	Example of the Muller Transform.	70
30	Realization of the example.	72
31	CDEC implicants for Incompletely specified function	72
32	Single output MCNC benchmarks with lower terms than Espresso. . .	75
33	Single output MCNC benchmarks for 5x, f, bw.	76
34	Single output MCNC benchmarks for misex, sao.	77
35	Single output MCNC benchmarks for con, duke, majority, rd, vg. . .	78
36	Multi output MCNC benchmarks.	79
37	CDEC gate and its NAND representation.	81
38	Sum of CDEC in NAND representation.	82
39	AND OR structure and its NAND and NOR representation.	83
40	Example function in AND OR and CDEC OR structure.	84
41	Decomposition to calculate DFC.	86
42	DFC for a CDEC gate	86

43	Machine Learning examples	87
44	Machine Learning examples	88
45	Scatter plot of input terms versus time.	92
46	Scatter plot of output terms versus time.	93

CHAPTER I

MOTIVATION FOR THIS RESEARCH

In many traditional PLD architectures, the excitation functions of the state macrocell, are realized with primitive AND, OR, ... gates [1] or with multiplexors [3]. If a primitive gate is used, the architecture allows the realization of high speed conditions to state macrocells, but the condition itself is only a very limited Boolean function. Multiplexors with several data and data select inputs can realize much more complex Boolean functions. However, the speed decreases when multiplexors with several input variables are used. Therefore, companies like *Cypress* and *Quicklogic* [7] have developed architectures with completely new kinds of logic gates, called the Condition Decoder gates or the *CDEC* gates.

A Programmable Logic Device from *Cypress* Semiconductor, *CY7C361* [5], has been optimized for realization of very fast asynchronous (self-synchronized) sequential controllers by means of concurrency and partitioning of FSMs. Such approach leads to Tokenized State Machines (TSM), a new concept which generalizes several known notions of parallelism, and which was used to synthesize a number of practical controllers [19]. Controllers are designed from multiple deterministic and non-deterministic finite state machines, and mapped into this new device. WARP, the PLD synthesis system based on behavioral VHDL [8], encompasses a unique combination of high-level synthesis, logic synthesis, and fitting.

The input array of *CY7C361* consists of an array of logic gates feeding, in a one-for-one fashion, the state macrocell array. Each logic gate in the array, called

Condition Decoder (*CDEC*), is a product of a product term and the complement of a product term, ($CDEC = P \cdot \overline{Q}$, where P and Q are product terms). The inputs to each gate come from all external inputs to the device, and from a number of feedback connections from macrocells, arranged in a segmented array. Therefore, compared to the traditional PLA architectures where the excitation functions of the state macrocells are Sum of Product (SOP) expressions, the excitation functions of the CY7C361 macrocells are *Sum of CDEC (SOC)* expressions.

As it was found analyzing the structural behavior of many real-life state machines, the *CDEC* gate is optimal for a very fast realization of conditions to state macrocells [6, 19]. It is optimal for the commonly encountered simple branch and join conditions of concurrent state machines. A problem that is related to the fitting problem [17] of the *Cypress CY7C361* chip is the *SOC Minimization*. Since the *CY7C361* has only 32 macrocells, a high quality logic minimization to reduce the number of macrocells is very important.

The SOC minimization problem was solved as a part of WARP compiler of *Cypress* [8], and preliminary work was done by Stefan Goller [28], and in this thesis, we present a tool that is superior to one from [8, 22]. The goal of this thesis is, however, more general, since we believe that *CDEC* can be used as a general-purpose gate for standard cell structures with few levels, and also for new PLD structures [29]. We depart therefore below from the *Cypress* chip as a sole motivation of our work, and we present a *generic logic synthesis problem of SOC minimization*. Since we designed our algorithms to give good results for very strongly unspecified functions defined by ON and OFF sets, they are applicable in Machine Learning [23] as well.

In this thesis, we will introduce a new method to solve the covering problems without generating prime *CDEC*-implicants. Therefore, the new approach

presented here is based on a new kind of graph coloring algorithm, similar to the one in [9, 10, 28]. We reduce the covering problem to the graph coloring problem. Instead of solving the covering problem with prime *CDEC*-implicants, we solve the coloring problem for a graph whose nodes correspond to minterms or some product implicants of a new type. Graph-coloring problems can be solved using heuristic, non backtracking algorithms for an approximate minimization, or optimal backtracking algorithms for an exact solution with the minimum number of implicants. In graph theory literature, methods are presented that trade-off quality for speed [12, 13] and are applicable to large graphs.

Also, we present a *Cube Splitting* algorithm, whereby the input cubes are split in such a way, that the newly generated cubes are lower in number than the minterms, and when these cubes are used as nodes in graph coloring algorithm, gives near exact solutions.

The thesis is organized as follows. The basic concept of the cube representation and operations used for SOC minimization is introduced in Chapter II. Chapter III describes the architecture, advantages and application of the *Cypress CY7C361*, and the basic concepts of Machine Learning and Pattern Theory, which are the motivation for our work. The *CDEC* gate of the *CY7C361* and the *CDEC*-implicants are described in Chapter IV. Chapter V describes the formulation of the SOC minimization problem and the approaches to solve it. The conditionally compatible cubes and the *Conditional Graph Coloring algorithm* are presented in chapter VI. Chapter VII introduces the motivation for the splitting of cubes and the algorithm for the splitting of the input cubes is also presented. Extension of the program for multi-output function and the advantages of using SOCMIN for Incompletely Specified functions are presented in Chapter VIII. The results of SOCMIN for MCNC

benchmarks and Machine Learning examples are investigated in Chapter IX and finally, Chapter X presents the conclusion.

CHAPTER II

CUBE REPRESENTATION AND OPERATIONS

In this chapter, we give some basic definitions and discuss some basic concepts. These concepts and definitions are necessary for the following chapters. The cube representation and operations used for SOC Minimization are introduced.

II.1 Cubical Representation

A *binary vector* (vector for short) is a series of symbols, where each symbol is either a 0 or a 1. We call each 0 or 1 a bit. An m -valued switching function f of n variables $X_1, X_2, \dots, X_n$ can be represented by binary vectors. If the variable X_i is m -valued, the literal $X_i^{s_i}$ can be represented by a binary vector of m bits: $c_i^0 c_i^1 c_i^2 \dots c_i^{m-1}$ where $c_i^j = 0$ if $j \in S_i$ and $c_i^j = 1$ if $j \notin S_i$. For example, for a binary logic, each literal can be represented by a vector of two bits as follows:

- $X_i^\emptyset$ is represented as 00,
- X_i^1 is represented as 01,
- X_i^0 is represented as 10,
- $X_i^{0,1}$ is represented as 11.

A product term of n literals can be represented by n such vectors. A symbol "-" is used for separating each vector. For example, $X_1^1 X_2^0 X_3^0 X_4^1$ in cubical representation is represented by four vectors: [01-10-10-01], which is called a *cube*. A cube can represent:

TABLE I
Example of binary input binary output function

a	b	c	f
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

1. a product of literals,
2. a sum of literals,
3. an exclusive sum of literals.

There is no difference in the representation of these forms as a cube. For example, $X_1^1 X_2^0 X_3^0 X_4^1$ and $X_1^1 + X_2^0 + X_3^0 + X_4^1$ are both represented as [01-10-10-01]. When using a cube, we should specify which form the cube is used to represent. We can represent a Product-Of-Sums(POS), a Sum-Of-Products(SOP) or a Exclusive-Sum-Of-Products(ESOP) by an array of cubes. This way of representation is called *positional notation of cube calculus (cube notation for short)*. In this thesis, without other specification, we use a cube to represent a product term, and use an array of cubes to represent an SOP.

Example II. 1 Given a function f as specified by Table I. Table I is called a *truth table*. A truth table enumerates the values of functions for each of their possible combinations of the inputs. Each row of the truth table presents a possible

ON-array			OFF-array		
a	b	c	a	b	c
0	0	0	0	0	1
0	1	1	0	1	0
1	1	0	1	0	0
1	1	1	1	0	1

(a)
(b)

Figure 1. ON and OFF array.

input combination and an associated output value. The set of those input combinations, whose associated output values are 1, is referred to as an *ON-array* of function f . The set of those input combinations, whose associated output values are 0, is referred to as an *OFF-array* of function f . The ON-array and the OFF-array of the function f is shown in Figure 1a and 1b, respectively. In Table I, a , b and c are binary input variables, f is a binary output variable. Please note that either ON-array or OFF-array of the function f contain the same information as the truth table.

The function f can also be specified as shown by the following Boolean equation 1:

$$f = \bar{a}\bar{b}\bar{c} + \bar{a}bc + ab\bar{c} + abc \quad (1)$$

where $\bar{a}\bar{b}\bar{c} + \bar{a}bc + ab\bar{c} + abc$ is a SOP expression. Please note that a , b and c in Table I are variables, while in the SOP expression they are used as literals.

A map of an n -variable, p -valued input, two-valued output function consists of p^n cells. Cells that contain a 1 will be called *true minterms (1-cells)* while cells that contain a 0 will be called *false minterms (0-cells)*. For a binary input function,

ab \ c	0	1
00	1	0
01	0	1
11	1	1
10	0	0

Figure 2. Karnaugh map for 3 variables.

the map is a *Karnaugh map*.

In Example 1, the function has 3 binary variables. So, a corresponding Karnaugh map has $2^3 = 8$ cells. There are 4 true minterms and 4 false minterms as shown in Figure 2.

Example II. 2 Consider a SOP equation for binary logic given by the following equation 2:

$$f = \bar{a}b\bar{c} + ad + \bar{a}\bar{b}c \quad (2)$$

The SOP equation (2) can be represented in cube notation as the following 3 cubes:

$[10 - 01 - 10 - 11] [01 - 11 - 11 - 01] [10 - 10 - 01 - 11]$.

For binary logic, we can simplify the cube notation as follows:

- 00 is represented as ϵ ,
- 10 is represented as 0,
- 01 is represented as 1,

Figure 3. Karnaugh map for binary input cubes.

- 11 is represented as x.

So, the above example can also be represented as

010x, 1xx1, 001x.

Example II. 3 The function f of Example 1 can be represented by the following ON-array of cubes - 000, 011, 110, 111. We can draw cubes on maps. On the map, each circle indicates one cube, as shown in Figure 3a. We can also use maps to minimize the number of cubes. For instance, 4 cubes in Figure 3a are reduced to 3 cubes in Figure 3b. In other words, we can use the ON-array of 3 cubes to express the function f of Example 1.

000, 011, 11x.

II.2 Cube Operations

If A is a cube, then the mapping $A^n \rightarrow A$ is a cube operation. We will only

discuss unary and binary cube operations.

The *supercube operation* of cubes A and B is defined as follows:

$$A \uplus B = [A_1 \cup B_1 \cup, \dots, A_n \cup B_n]$$

where $A_i \cup B_i$ is a set union.

The *intersection operation* of cubes A and B is defined as follows:

$$A \cap B = \begin{cases} [A_1 \cap B_1 \cap, \dots, A_n \cap B_n] & \text{if there is no such } i \text{ that } A_i \cap B_i = \emptyset \\ \emptyset & \text{otherwise} \end{cases}$$

where $A_i \cap B_i$ is a set intersection and $\emptyset$ is an empty cube.

Since we use a cube to represent a term, the operation of terms can be represented as the operation of cubes. For example, $A = [A_1, \dots, A_n]$ and $B = [B_1, \dots, B_n]$ represent two terms $T_S = X_1^{S_1}, \dots, X_n^{S_n}$ and $T_R = X_1^{R_1}, \dots, X_n^{R_n}$ respectively, then the supercube operation of two cubes A and B

$$A \uplus B = [A_1 \cup B_1 \cup, \dots, A_{i-1} \cup B_{i-1}, A_i \cup B_i, A_{i+1} \cup B_{i+1}, \dots, A_n \cup B_n]$$

is equivalent to the supercube operation of two terms T_S and T_R .

$$T_S \uplus T_R = X_1^{S_1 \cup R_1}, \dots, X_{i-1}^{S_{i-1} \cup R_{i-1}}, X_i^{S_i \cup R_i}, X_{i+1}^{S_{i+1} \cup R_{i+1}}, \dots, X_n^{S_n \cup R_n}.$$

In cube notation, an operation between two variables is a local operation, and an operation between two cubes is a global operation. A local operation is a set operation, and a global operation is a cube operation. For example, the supercube operation $A \uplus B$ is a global operation, and $A_i \cup B_i$ is a local operation.

Sometimes, different local or global operations may be performed on two cubes depending on a relation between them. In cube notation, a relation between two vectors (corresponding to a variable) is a local relation and a relation between

two cubes (corresponding to two terms) is a global relation. For instance, given two cubes $A = [A_1, \dots, A_n]$ and $B = [B_1, \dots, B_n]$, then $A \subseteq B$ is a global relation, and $A_i \subseteq B_i$ is a local relation. A global relation is satisfied if all the local relations are satisfied.

The disjoint sharp operation on cubes A and B is defined as follows:

$$A \#_d B = \begin{cases} A & \text{when } A \cup B = \emptyset \\ \text{emptyset} & \text{when } A \subseteq B \\ A \#_{d_{\text{basic}}} B & \text{otherwise} \end{cases}$$

$$\text{where } A \#_{d_{\text{basic}}} B = \{A_1, \dots, A_{i-1}, \neg B_i \cap A_i, A_{i+1} \cap B_{i+1}, \dots, A_n \cap B_n\}$$

In this formula, $A \cap B$ and $A \subseteq B$ are global relations. If the relation $A \cap B = \emptyset$ is true (satisfied), the resultant cube is A . If the relation $A \subseteq B$ is true, an empty cube is generated. Otherwise the global operation $A \#_d B$ is performed. $A_i \not\subseteq B_i$ is a local relation, and both $A_i \cap B_i$ and $\neg B_i \cap A_i$ are local operations.

Example II. 4 Given two terms bc and acd . They can be expressed as two cubes. $A = [11 - 01 - 11 - 01]$ and $B = [01 - 11 - 01 - 01]$. These two cubes are shown in Figure 4a. Figure 4b shows that $A \cap B \neq \emptyset$ (none of the vectors in resultant cube are empty sets). Next we check if the relation $A \subseteq B$ is true. Note that $A \subseteq B$ is equivalent to $\neg A \cup B \neq \emptyset$. We can also check if the relation $A \subseteq B$ is false. Figure 4c converts B to $\neg B$. Here $\neg B = [\neg B_1, \dots, \neg B_n]$ is a global relation. Figure 4d intersects A and $\neg B$ and shows that two of the local relations are true ($\neg B_1 \cup A_1 \neq \emptyset$ and $\neg B_3 \cup A_3 \neq \emptyset$). These two local relations are indicated by two arrows. Since the relation $A \subseteq B$ is false, the operation $A \#_{d_{\text{basic}}} B$ should be performed. Two resultant cubes are generated in Figure 4e and Figure 4f, respectively. Figure 4g shows the final results in the Karnaugh map. Note that the disjoint sharp operation of cubes A and B generates an array of disjoint cubes,

Figure 4. Example for Sharp operation.

which cover the minterms in cube A but not in cube B .

The *symmetric consensus operation* of cubes A and B is defined as follows:

$$A \# c B = \begin{cases} A \cap B & \text{when } \text{distance}(A, B) = 0 \\ \emptyset & \text{when } \text{distance}(A, B) > 1 \\ A \# c_{\text{basic}} B & \text{otherwise} \end{cases}$$

where $A \# c_{\text{basic}} B$ is defined as

Intersection operation

Disjoint Sharp operation

Consensus operation

Figure 5. Cube operations.

$X_1^{A_1} \cap B_1, \dots, X_{i-1}^{A_{i-1}} \cap B_{i-1}, X_i^{A_i} \cap B_i, X_{i+1}^{A_{i+1}} \cap B_{i+1}, \dots, X_n^{A_n} \cap B_n$ for $i = 1, \dots, N$.

The *asymmetric consensus operation* is similar but operation $\neg(A_i \subseteq B_i)$ is additionally checked.

$$A \# a B = \{X_1^{S_1 \cup R_1}, \dots, X_{i-1}^{A_{i-1}} \cap B_{i-1}, X_i^{A_i} \cap B_i, X_{i+1}^{A_{i+1}} \cap B_{i+1}, \dots, X_n^{A_n} \cap B_n\}$$

for such $i = 1, \dots, N$, that $\neg(A_i \subseteq B_i)$.

Example II. 5 The Supercube, Intersection, Disjoint sharp and consensus operations are shown in Figure 5. Figure 5a shows the Supercube operation on the Karnaugh map. The supercube of two cubes X1011 and X110 is given by the cube X1XX. The Intersection of two cubes A and B is given by the cube I, as shown in Figure 5c and d respectively. The Disjoint sharp operation of the cubes A and B gives two resultant cubes 1 and 2, shown in Figure 5e and f. Finally, Figure 5g and h shows the Consensus operation for cubes A and B resulting in the cube C.

CHAPTER III

THE CYPRESS CY7C361 EPLD AND MACHINE LEARNING

III.1 The Cypress CY7C361 EPLD

Cypress Semiconductor introduced an EPLD, the *CY7C361* [6] to realize very fast asynchronous (self-synchronized) sequential controllers by means of concurrency and partitioning of FSMs. In the following section III.1.1, we present the architecture of the *CY7C361*.

III.1.1 The Architecture of the CY7C361

The *Cypress CY7C361* [6] represents a EPLD device with only 32 macrocells and highly limited connectivity between the macrocells. The *state macrocell array* is located between an *input* and an *output array*. The input array allows the realization of very fast conditions to state macrocells as a function of the inputs of the chip and the outputs of the state macrocells. The output array allows the realization of the outputs of the chip as a function of the state macrocells output only. The architecture of the chip is shown in Figure 6.

The input array consists of the Segmented Interconnection Array and an array of logic gates feeding, in a one-for-one fashion, the state macrocell array. Each logic gate in the array is the product of a product term and the complement of a product term. It is called *Condition Decoder (CDEC) gate* ($CDEC = P * \overline{Q}$, where P and Q are product terms). The inputs to each *CDEC* gate come from all external inputs to the device and from a number of feedback connections from the macrocells

outputs. As it can be seen in Figure 6, the feedback paths are realized as short wires being very close to the associated macrocell.

Contrary to other architectures from Xilinx [3], Actel [2] or Concurrent Logic, the macrocells of the *Cypress* chip do not correspond to the direct realizations of Boolean function (AND, OR, etc.) and storage elements (D-flip-flops, latches, etc.). They are designed for the immediate realization of the basic elements of Tokenized State Machines. The methodology of Tokenized State Machine design developed for this chip is described in [16]. The idea for this new architecture was created by analyzing the utilization of the behavioral PLA structure. Conventional two-level PLD architectures consist of an input plane and an array of state macrocells triggered by the outputs of the input plane. For PLAs/PALs the input plane consists of a programmable AND/OR array that allows the realization of Sum Of Product (SOP) terms as Boolean functions for the conditions to the state macrocells. Usually, there is only little feedback from the output of the macrocells back to the input plane.

When comparing the CY7C361 architecture to conventional two-level PLD architectures, there is much more feedback from the state output of the macrocell array back to the input array. The feedback paths of the *CY7C361* are realized with very short wires that allow a shorter cycle time and result in a higher speed of the device.

Compared to traditional PLA architectures where the excitation functions of the state macrocells are Sum of Product (SOP) terms, the excitation functions of the CY7C361 macrocells are Sum of *CDEC* gate expressions. As it was found by the analyzing the structural behavior of many real life state machines, this form of condition decoder is optimal for a very fast realization of conditions to state macrocells [16]. The *CDEC* gates require a new method to minimize Boolean functions to

Sum of *CDEC* gates expressions.

Figure 6. Architecture of the *CY7C361*.

III.1.2 Applications and advantages of Cypress CY7C361

System designer specifications like Ethernet, VMEBus, SCSI, Futurenet, FDDI, ISO, Multibus, etc., lead to parallel implementation of machines, because blocks of functionality described in these specifications are naturally expressed in terms of communicating state machines. Also, divide and conquer design methods allow for controlling the complexity of the design. The use of subroutine/coroutine-like paral-

lel components makes it easier for the user to describe, modify, debug and recombine separate pieces of the total concurrent description. Generally, an implementation with multiple communicating FSMs results in a more compact and faster circuit.

The State Machine Device (SMD), *CY7C361*, is targeted at applications requiring accurate sequential control with minimal time granularity (8ns). The SMD is most useful when an application breaks down into several concurrent state machines, or when the input test conditions are complex. When comparing the SMD architecture to conventional two-level PLD architectures, the registers present between the input array and the output array give rise to shorter cycle time. There is also much more feedback from the state macrocell array back to the input array. In addition, control codes can be implemented in the shift array of macrocells. The input array consists of an array of logic gates feeding, in a one-for-one fashion, the state macrocell array. The inputs to each logic gate, called the *CDEC* gate, come from all external inputs to the device, and from a number of feedback connections from macrocells, arranged in a segmented array. This form of condition decoder is optimal for the commonly encountered simple branch and join conditions of concurrent state machines.

The *CY7C361* from *Cypress* Semiconductor also permits for straightforward realization of Petri nets and other high-level parallel sequential circuit descriptions. Clock Generator chip can be used with this kind of devices for direct implementation of high-level specification of self-synchronized circuits.

III.2 Machine Learning and KDD

In the recent years, the methods developed in Design Automation area have been increasingly used by researchers in other fields. Till now, the techniques devel-

oped in Design Automation, have been concentrated mainly in the field of circuit design. But, as the methods and techniques researched are so powerful and universal that these are increasingly used by researchers outside the field of circuit design. For instance, they are now used in machine learning, automatic theorem proving, robotics, industrial operations scheduling, stock market prediction, genetics research, and many others [23]. It is quite probable, that the methods developed in design Automation will lead to advances and breakthroughs in these other areas.

Logic Synthesis is one of the sub-areas of Design automation which can find numerous external applications. In the past few years, they have found increased applications in machine learning, Knowledge discovery, Knowledge acquisition, database optimization, AI, Image coding, automatic theorem proving and verification of hardware and software [23]. In this chapter, we will investigate the uses of SUM-OF-*CDEC* (SOC) minimization in Machine Learning (ML) and Knowledge Discovery in Databases (KDD) applications.

III.2.1 Basic research idea of the PTG and FDG

Pattern Theory group (PTG) at Avionics Laboratory of Wright Laboratories, develops new system-level concepts for military applications, mostly based on machine learning and image processing technologies. Machine Learning is any method of teaching a computer to recognize any kind of patterns. The approach of the PTG is based on logic synthesis methods: predominantly the so-called Curtis Decomposition of Boolean functions. The new approach of PTG will allow both automatic learning of any kind of images, and automatic creation of algorithms. Consequently, the programming system FLASH developed by this group [23], using the many decomposition ideas, is a Testbed for machine learning based on logic synthesis ap-

proach. The PTG, in conjunction with the Function Decomposition Group (FDG) at Portland State University, performs research on using function decompositions to Machine Learning applications.

In a nutshell, the main difference between the previous approaches to machine learning and the logic synthesis approach is that, in these previous approaches the recognizing network had some assumed structure which was next only "tuned" by the learning process (for instance, by decreasing and increasing the numerical values of some coefficients). Thus, creating a new structure is done by setting some coefficients to zero. All "new" structures are in a sense "hidden" in the assumed mother-structure. Also the type of an element, such as a formal neuron in Neural Nets, or a polynomial in Abductive Inference Mechanism is decided before the learning takes place (AIM¹). In contrast, in the Curtis Decomposition approach, there is no any assumption on the kind of structure of the learning network and the type of the elements. The elements are arbitrary discrete mappings (functions). Both the structure and the elements are **calculated** in the learning process, and this process is entirely based on finding patterns in data [23].

The central concept of Pattern Theory is a "pattern". In Pattern Recognition and Knowledge Discovery the problems with nice representations based on "features" belong to a more general class of problems with strong "patterns." Pattern finding is, therefore, a generalization and formalization of feature extraction. The goal of the Pattern Theory is to support "automating" the pattern finding process, and to construct a representation of a function based on examples; therefore, this is a method for constructive induction. Also, it constructs this representation by minimizing complexity, similar to Occam-based learning. The Ashenhurst

¹Trademark of AbTech Corp.

Function Decomposition (AFD) learns both the architecture of the combinational representation and the component functions from the examples and is implemented in FLASH [23].

III.2.2 Decomposed Function Cardinality

Pattern Theory [23] treats robust complexity determination as the problem of finding a pattern. Pattern theory uses Decomposed Function Cardinality (DFC), proposed by Y. S. Abu-Mostafa as a general measure of complexity [27, p.128]. DFC is based on the cardinality of a function. The Pattern theory is based on two central ideas. The first is that the functions that the investigator wishes to learn, have low Decomposed Function Cardinality. The second is functions with low DFC are learnable with a relatively small number of samples. In this thesis, we will show that some of the results generated by the SOC minimization, have a low DFC as compared to other decomposition methods and hence could find important application in machine Learning, and particularly, in Pattern Theory.

Decomposing a function involves breaking it up into smaller subfunctions. These smaller subfunctions are further broken down until all subfunctions will no longer decompose. For a given function, the number of ways to choose two sets of variables (the partition space) is exponential. The decomposition space is even larger, because there are several ways the subfunction can be combined and there are several levels of subfunction possible. The complexity measure that is used to determine the relative predictive power of different function decompositions is the Decomposed Function Cardinality.

III.2.3 SOC minimization for Machine Learning

The core idea of both the Curtis decomposition and the SUM-OF-*CDEC* minimization approach of SOCMIN, is to look for certain patterns in data. In Curtis decomposition these patterns are in columns of the map corresponding to the cofactors of the bound set [30]. In our SOC minimization approach in SOCMIN, the patterns are based on certain rules of Boolean Algebra.

Let us observe that both in Curtis decomposition and the SUM-OF-*CDEC* minimization approach of SOCMIN, the central idea is to minimize certain cost of the circuit. Traditionally, in any minimization one calculates the number of terms and the number of literals. In our case, we calculated additionally the total DFC as a sum of DFC of all non-decomposable blocks (this is an upper bound approximation of real DFC).

Now, let us discuss why SOC minimization has important applications to Machine Learning. The data used in Machine Learning problems have an extremely high percent of don't cares, which are combination of argument values for which the function value is not specified. The missing data can be represented as don't cares. Most of the minimizers like EXMIN or EXORCISM gives inferior results when there is an extremely high percentage of don't cares [23]. Espresso performs better than most other logic minimizers in those cases.

The most important advantage of SOCMIN is that it gives very high quality solutions for very strongly unspecified functions. The higher the percentage of don't cares in a function, the better is the quality of the solution provided by SOCMIN. And as the solution given out by SOCMIN is always better or equal to Espresso, this gives a better edge to SOCMIN over other minimizers for Machine Learning applications.

Another important property needed for machine learning is simultaneous classification of patterns to more than two categories (we want not only to distinguish a "friend from foe" airplane, but you want to learn its orientation, speed, etc.). This is nothing but the concurrent minimization of switching functions with many outputs in Logic Synthesis. Current FLASH operates on single-output functions, but SOCMIN works with multi-output functions. There are many decomposers that decompose multi-output functions, but all of them have been designed for circuit design. One needs a minimizer for strongly unspecified multi-output functions. SOCMIN satisfy the multi-output requirement and it can also be easily extended to handle the multi-valued input functions.

CHAPTER IV

CDEC GATE AND CDEC IMPLICANTS

The following chapter introduces the Condition Decoder gate or the *CDEC* gate of the Cypress *CY7C361* and the implicants formed by the *CDEC* gate called the Condition Decoder implicants or the *CDEC* implicants.

IV.1 The Condition Decoder gate

A single state macrocell of the *Cypress CY7C361* is triggered by a single *CDEC* gate. Therefore, the excitation function of the macrocell in Figure 8 has to be a *CDEC* implicant. In order to realize excitation functions f that are not *CDEC* implicants, a decomposition of f to a sum of *CDEC* implicants is necessary. Figure 8 shows the realization of a function f being the sum of several single *CDEC* implicants.

A Boolean function f that is the sum of n *CDEC* implicants can be realized on the *CY7C361* as shown in Figure 7. Each *CDEC* implicant $IM_1, \dots, IM_n$ is realized with a single *CDEC* gate and a state macrocell of the type START. An additional *CDEC* gate with a state macrocell $n+1$ becomes necessary and as a result, the function f is available at the output of the state macrocell $n+1$. Let us assume that the input of the state macrocells $1, \dots, n$ are available at the time interval t_i . The output of a START cell has a delay of one clock cycle compared to the output of the corresponding *CDEC* gate. Therefore, the outputs of the macrocells $1, \dots, n$ are available at the time interval t_{i+1} , and the output of macrocell $n+1$ at t_{i+2} . This

Figure 7. CDEC gate

Figure 8. Boolean function f as a SOSC.

additional delay times of one clock pulse has to be considered in the timing diagram of the state machine synthesis, if a sum of single *CDEC* functions has to be realized on the chip. The input array of the *CY7C361* is based on the Condition decoder *CDEC* gate, shown in Figure 7.

The Condition Decoder of the *CY7C361* forms a product of a product term and the complement of a product term, ($CDEC = P \cdot \overline{Q}$, where P and Q are product terms). In other words, it forms a product of the product and a sum over the input field. The sum term is obtained by inverting the inputs to Q . In a conventional PLA

Figure 9. Realization of SOSC forms with the *CY7C361*.

or PLD device, only P would be present in the first array and the output and the feedback would be encoded by a second programmable or fixed array. The speed of the state machines is limited mainly by the feedback.

Since there is immediate feedback information in the input field, multiway fork and join operations can be performed using this type of Condition Decoder. In other words, the Condition Decoder is used to control or gate the token being passed from macrocell to macrocell. In contrast, a traditional PLD or PLA requires more logic because the array is used to encode the gates.

Each Condition Decoder has a miser bit in its sum term path. If the term

is not used, the miser bit is automatically programmed. The miser bit completely disconnects the product term and replaces it with a logic HIGH. This results in power saving.

The input array has 41 Condition Decoders: one global reset decoder, 8 local reset decoders, and 32 macrocell decoders. The array has 44 true/complement input pairs, 88 inputs total.

IV.2 CDEC Implicants

The input array of the *CY7C961* is based on the Condition Decoder *CDEC* gate. The basic concept of the *CDEC* synthesis is that of the *CDEC* implicant. *CDEC* implicant is an implicant that can be realized with a single *CDEC* gate. We need to introduce some basic concepts and definitions to understand *CDEC* implicants.

IV.2.1 Definitions

The following basic concepts and definitions are introduced to grasp the concept of *CDEC* implicants.

Definition IV. 1: Let $f(x_1, x_2, \dots, x_n)$ be a switching function and $h(x_1, x_2, \dots, x_n)$ be a product of literals. If f covers h , then h is said to *imply* f , or h is said to be an *implicant* of f . The implication is often denoted $h \rightarrow f$.

Example IV. 1 If $f = ab + cd$ and $h = ab\bar{c}$, then f covers h and h implies f . This is shown in Figure 10.

Definition IV. 2: A prime implicant p of a function f is a product term which is covered by f , such that the deletion of any literal from p results in a new product

Figure 10. Karnaugh map for h implies f

which is not covered by f . Alternatively stated, p is a prime implicant if and only if p implies f , but does not imply any product with fewer literals which in turn also implies f . The set of all prime implicants of f will be denoted by P .

Example IV. 2 $\bar{a}\bar{b}$ is a prime implicant of $f = \bar{a}b + ac + \bar{b}\bar{c}$, since it is covered by f and neither $\bar{a}$ nor $\bar{b}$ alone implies f . This is shown in Figure 11.

Example IV. 3 Consider the Karnaugh map of Figure 12a where the function f is given by the minterms $\bar{a}\bar{b}\bar{c}\bar{d}$, $\bar{a}b\bar{c}\bar{d}$, $\bar{a}b\bar{c}d$, $\bar{a}bcd$, $ab\bar{c}\bar{d}$, $ab\bar{c}d$, $abcd$, $a\bar{b}\bar{c}\bar{d}$, $a\bar{b}\bar{c}d$.

The set of all prime implicants of f is $P = (\bar{a}\bar{c}d, \bar{a}b\bar{c}, bd, a\bar{c}d, a\bar{b}\bar{c}, \bar{b}\bar{c}\bar{d})$. This is shown in Figure 12b. Note that abd is not a prime implicant since it implies bd .

Definition IV. 3: A prime implicant p of a function f is said to be an essential prime implicant if it covers at least one minterm of f which is not covered by any other prime implicants. Since every minterm of f must be covered by an expression for f , all essential prime implicants must be contained in any irredundant expression for the function.

Figure 11. Karnaugh map for prime implicants

Figure 12. Karnaugh map for prime implicants

IV.2.2 CDEC Implicants

The concept of *CDEC* gate synthesis is based on *CDEC implicants*. A *CDEC* implicant is an implicant that can be realized with a single *CDEC* gate. It has an *AND-part* and a *NAND-part* sometimes called an *OR-part*, since $abcd = ab(\bar{c} + \bar{d})$. A *CDEC* implicant is represented in the computer as a pair of cubes: (AND-part, NAND-part). Analogously to prime and essential prime product implicants, prime

Figure 13. CDEC implicant.

and essential prime *CDEC* implicants can be defined.

Definition IV. 4: A *CDEC* implicant is an implicant of the form $P * \bar{Q}$ where P and Q are product terms.

Definition IV. 5: A **prime *CDEC* implicant** is a *CDEC* implicant that is not totally included in any other *CDEC* implicant.

Definition IV. 6: An **essential prime *CDEC* implicant** is a prime *CDEC* implicant that covers a minterm that is not covered by any other *CDEC* implicant.

A *CDEC*-implicant is the generalization of the concept of a product implicant. Product implicants and prime implicants are *CDEC*-implicants as well.

Example IV. 4 Let us illustrate a *CDEC*-implicant for one *CDEC* gate as shown in Figure 13. Let the function be given by minterms 0101, 0111 and 1111. We need two prime implicants to cover this function. The *CDEC* implicant is given by $f = bd * \bar{a}\bar{c}$ where the AND part is bd and the NAND part is $\bar{a}\bar{c}$. This can be put in one *CDEC* gate.

CHAPTER V

FORMULATION OF THE SOC MINIMIZATION PROBLEM

Standard minimizers for Boolean functions (such as Espresso from U.C. Berkeley) minimize a function f to a sum of product terms (SOP). The *CDEC* minimization results in a SUM-OF-*CDEC* implicants(SOC). In general, the representation as a sum of *CDEC* implicants consists of a smaller or equal number of *CDEC* terms compared to the number of product terms in the SOP representation. In other words, the exact SOC representation of function f has never more gates than a SOP of this function.

An approach to minimize Boolean functions f using *CDEC* gates is outlined below. The result is a representation of f as a sum of *CDEC* implicants.

1. Generate all prime implicants of function f (Espresso).
2. Factorize the prime implicants to prime *CDEC* implicants.
3. Solve the covering problem with true minterms as columns and prime *CDEC* implicants as rows, such that the number of *CDEC* implicants is minimal.

This approach, used in the *Cypress* Development System for the *CY7C361* [8] has several drawbacks.

1. First and most importantly, since not all prime cdec-implicants can be generated by factoring prime implicants, the minimum covering with prime cdec-implicants (called cdec covering from now) will be lost.

Figure 14. Approaches to the Minimization using *CDEC* Gates.

2. The generation of prime cdec-implicants based on standard factoring procedures is not efficient.
3. Since the number of prime cdec-implicants is much larger than that of prime implicants, the covering table of large size is created, which cannot be solved even for functions for which exact minimal prime covering would be feasible.
4. Using minterms in the table additionally limits the size of the problems that can be solved.

Our first observation is that in order to generate all prime cdec-implicants one has to start from some product implicants that are not necessarily prime implicants. Let us illustrate this fact on some examples.

Example V. 1 An example where the approach based on prime implicants can not find the minimum *CDEC* cover is given in Figure 14a. The function f can be represented as the sum of three prime implicants according to Equation 3. Based on

these three prime implicants, f can not be further factorized to *CDEC* implicants. The *CDEC* realization consists of 3 *CDEC* gates and the *CDEC* implicants are identical to the prime implicants.

$$f = \bar{c}d + \bar{a}b\bar{c} + abd \quad (3)$$

However, the function f can be realized with only two *CDEC* gates. The minimal solution can be obtained by the factorization of implicants that are not primes, as shown in Figure 14b and Equation 4. The two *CDEC* implicants resulting from the minimization are $\bar{a}\bar{c}(\bar{b}\bar{d})$ and $ad(\bar{c}\bar{b})$:

$$f = \bar{a}b\bar{c} + \bar{a}\bar{c}d + a\bar{c}d + abd = \bar{a}\bar{c}(b + d) + ad(\bar{c} + b) = \bar{a}\bar{c}(\bar{b}\bar{d}) + ad(\bar{c}\bar{b}) \quad (4)$$

Two important principles can be learned from this example:

1. Prime cdec-implicants must be created not only from prime implicants, but also from some product implicants included in prime implicants.
2. Essential prime implicants are useful to create essential prime cdec-implicants, that are next included in the exact minimum cdec cover.

Observation 1 can be used in an algorithm to create all cdec-implicants. Observation 2 can be used in both exact and approximate algorithms to find cdec coverings more efficiently by finding essential and secondary essential cdec-implicants first, and thus reducing the coloring graph used for combinatorial selection. Our approach is based on graph coloring, rather than on covering.

The basic exact minimization algorithm for a single output network with OR gate in the first level and *CDEC* gates in the second level may be created similarly

to the classical two-level AND-OR network minimization algorithm:

1. Find all prime cdec-implicants.
2. Find the covering of all true minterms with prime cdec-implicants (found in step 1) that minimizes the number of cdec-implicants (or the total cost of literals in prime cdec-implicants).

This kind of exact approach can be used only for functions with a small number of input variables. The number of prime cdec-implicants increases rapidly with the number of minterms, especially for functions with many don't cares. It is well known that the set of prime cdec-implicants can become too large to enumerate even if it is possible to find the exact minimum cover [26]. Similar property can be shown for prime *CDEC*-implicants. Results of this kind limits the application of algorithms based on generating all primes or prime cdec-implicants. Also, removing primes included in prime *CDEC*-implicants can lead to a loss of *CDEC* cover with minimum literal cost. In addition, the covering problem is NP-hard. Some problems that lead to extremely hard to solve covering problems have been constructed. It results then, that there are two reasons why the current approaches to exact minimization will meet limited success.

In this chapter, we will introduce a new method to solve the covering problems without generating prime cdec-implicants. Therefore, the new approach presented here is based on a new kind of graph coloring algorithm, similar to the one in [9], [10] and [11]. We reduce the covering problem to the graph coloring problem. Instead of solving the covering problem with prime cdec-implicants, we solve the coloring problem for a graph whose nodes correspond to minterms or some implicants of a new type. Therefore, we solve one NP-hard problem (graph coloring)

instead of two NP-hard problems (the generation of prime cdec-implicants and the covering).

Graph-coloring problems can be solved using heuristic, non backtracking algorithms for an approximate minimization, or optimal backtracking algorithms for an exact solution with the minimum number of implicants. In graph theory literature, methods are presented that trade-off quality for speed [12] [13] and are applicable for large graphs. In Chapter VI, a new heuristic algorithm is used to solve the *Graph CDEC-Compatible Coloring Problem*. This algorithm is used to minimize single-output Boolean functions in a sum of *CDEC* form.

The graph for coloring is created with any cubes from the ON-set of the function f as vertices. These can be either minterms M , (disjoint) product implicants, minimal product implicants (product of all prime implicants covering M), or disjoint minimum implicants PI [4] [9] [10]. The advantage of using minterms is the warranty of the optimum solution, the disadvantage is the size of the graph for functions with a high number of inputs. If the number of inputs is N , the number of minterms is of the order of 2^{N-1} , and the graph will be too big to construct. The advantage of using arbitrary disjoint cubes is the improved execution speed, but therefore, the minimum *CDEC* cover is lost. The algorithm presented in Chapter VI, starts from minterms, disjoint cubes, minimum implicants or arbitrary cubes in order to construct the graph.

The *CDEC* implicants from the sum of *CDEC*s solution can be: prime *CDEC* implicants (to minimize power or/and increase routability), or arbitrary *CDEC* implicants. The fast algorithm presented here returns the quasi-minimum cover of prime *CDEC* implicants.

CHAPTER VI

THE CONDITIONAL GRAPH COLORING

Boolean functions f are implemented as arrays (sets) of cubes. The following notation will be used: $\text{ON}(f)$ is the set of ON-cubes of f , $\text{OFF}(f)$ is the set of OFF-cubes, and $\text{DC}(f)$ is the don't care set of f . A Cube C_i is a string of 0's, 1's, X's, and ϵ 's; it represents a product of literals of the function f . An implicant of a function is an arbitrary subset of its ON-cubes and DC-cubes. A product implicant is an implicant being a cube. A prime implicant is a product implicant which is not included in any other other implicant of that function [4]. Standard notions of literals, sum of products, essential and secondary essential prime implicants, consensus, sharp, disjoint sharp, intersection and Hamming distance of cubes will be used. The cubes C_i below can be of any kind, if not mentioned otherwise.

VI.1 Supercube Operation

The main cube operator used in the minimization algorithm is the *supercube* [4]. The supercube of two cubes C_i and C_j is denoted by $C_i \uplus C_j$. The literals of the Cube C_i and C_j are encoded as shown in Table II. When the positional cube notation is used, the supercube operator corresponds to the component-wise Boolean OR of the two cubes. This is the smallest cube that includes both C_i and C_j . In other words, the supercube $C_i \uplus C_j$ is obtained by an OR operation of the single literals of C_i and C_j . Figure 15 shows an example of the supercube of the cube C_i and C_j .

TABLE II
Literal Encoding

literal	encoding	meaning
0	10	logic "0"
1	01	logic "1"
X	11	don't care
ϵ	00	is not

Figure 15. Supercube of two cubes C_i and C_j .

Example: $C_i = X101$, $C_j = X110$, $C_i \uplus C_j = X1XX$

VI.2 Conditionally Compatible Cubes

In order to formulate the *CDEC* compatible graph coloring problem the following definitions are presented. The cubes C can be minterms, (disjoint) product implicants, minimal product implicants, or disjoint minimum implicants.

Definition VI. 1: Two cubes $C_i \in ON(f)$ and $C_j \in ON(f)$ are said to be of

distance-1 if there is exactly one variable $C_k \in C_i$ and C_j which has different sets of truth values.

Definition VI. 2: Two cubes $C_i \in ON(f)$ and $C_j \in ON(f)$ are **compatible** if their supercube $C_i \sqcup C_j$ does not overlap with any Cube $C_k \in OFF(f)$ (i. e. does not include a false minterm).

$$(C_i \sqcup C_j) \cap OFF(f) = \emptyset \Rightarrow C_i \text{ and } C_j \text{ are compatible} \quad (5)$$

These cubes can be combined to one *CDEC* implicant with their supercube $SU_{ij} = C_i \sqcup C_j$ as the AND-part (the NAND-part is 0).

Definition VI. 3: Given are two cubes $C_i \in ON(f)$ and $C_j \in ON(f)$ and their supercube $SU_{ij} = C_i \sqcup C_j$. If the supercube SU_{ij} overlaps with $OFF(f)$, the intersection is called the **OFF-part** $OFFP(SU_{ij})$ of the supercube SU_{ij} .

$$OFFP(SU_{ij}) = OFF \cap SU_{ij} \quad (6)$$

The supercube $\bigsqcup_{C_l \in OFFP(SU_{ij})} C_l$ of all cubes of the OFF-part $OFFP(SU_{ij})$ is denoted by $SOFFP(SU_{ij})$.

Definition VI. 4: Two cubes C_i and C_j are called **conditionally compatible** if $SOFFP(SU_{ij})$ does not intersect C_i nor C_j . Such cubes can be combined to one *CDEC* implicant $(SU_{ij} \cdot \overline{SOFFP(SU_{ij})})$, with cube SU_{ij} as the AND-part and cube $SOFFP(SU_{ij})$ as the NAND-part. The cube $SOFFP(SU_{ij})$ is called the **condition cube** under which C_i and C_j are conditionally compatible.

Definition VI. 5: Two cubes are **incompatible** if they are not compatible nor conditionally compatible. Such cubes cannot be combined to a single *CDEC* implicant.

Definition VI. 6: The set Π of cubes is called a **set of compatible cubes** if each pair $C_i, C_j \in \Pi$, is either compatible or conditionally compatible with respect to the same condition cube $SOFFP(\biguplus_{C_i \in \Pi} C_i)$. The set Π can be described by a single *CDEC* implicant:

$$(AND \text{ part}, NAND \text{ part}) = (\biguplus_{C_i \in \Pi} C_i, SOFFP(\biguplus_{C_i \in \Pi} C_i)) \quad (7)$$

Example VI. 1 Let us illustrate the definitions with an example shown in Figure 16. The Boolean function f is represented by the cubes

$$f = \bar{a}\bar{b}\bar{c}\bar{d} + \bar{a}\bar{c}d + a\bar{c}d + abcd \quad (8)$$

Let $C_1 = \bar{a}\bar{b}\bar{c}\bar{d}$, $C_2 = \bar{a}\bar{c}d$, $C_3 = a\bar{c}d$, $C_4 = abcd$.

We will show the example for a pair of compatible cubes, a pair of conditional compatible cubes, a pair of incompatible cubes and a pair of distance-1 cubes. The cubes C_2 and C_3 are compatible, because their supercube $C_2 \uplus C_3 = \bar{c}d$ does not intersect the OFF-set of f . The cubes C_1 and C_2 are conditionally compatible under the condition cube $SOFFP(\bar{a}\bar{c}) = \bar{a}\bar{b}\bar{c}\bar{d}$. Their supercube $C_1 \uplus C_2 = \bar{a}\bar{c}$ overlaps with the OFF-set of f . Therefore the OFF-part of $C_1 \uplus C_2$ is given by $OFFP(C_1 \uplus C_2) = \bar{a}\bar{b}\bar{c}\bar{d}$. However, $SOFFP(\bar{a}\bar{c})$ does not overlap C_1 nor C_2 . The cubes C_1 and C_4 are not compatible, because their supercube $C_1 \uplus C_4 = b$ overlaps with the OFF-set of f and the supercube of the OFF-part of $C_1 \uplus C_4$ overlaps with the cubes C_1 and C_4 . Sets of conditionally compatible cubes are: $\{C_1, C_2\}$ and $\{C_3, C_4\}$. The cubes C_1 and C_2 are of (Hamming) distance-1, as there is exactly one variable d that has different sets of truth values while the C_1 and C_4 are not of distance-1, as there are three variables a , c and d that have different sets of truth values (Hamming distance = 3).

Figure 16. Examples of (conditionally) compatible cubes.

VI.3 Conditional Graph Coloring

The input of the *Conditional Graph Coloring* algorithm is the non-ordered graph $GCCC = (SPI, SNE, SCE)$, where SPI is the set of nodes corresponding to product implicants (in particular, minterms) of function f . SNE is the set of non-directed *normal edges* and SCE is a set of non-directed *conditional edges* between the nodes. Two nodes $PI_i \in SPI$ and $PI_j \in SPI$ are connected by a *normal edge* if PI_i and PI_j are incompatible. Such cubes must be colored with various colors. If there is no edge between nodes, these nodes can be colored with the same color. PI_i and PI_j are connected by a *conditional edge* if PI_i and PI_j are compatible under the condition indicated by the label of the edge. The label $l(SC_i)$ of the conditional edge connecting the nodes PI_i and PI_j represents the $SOFFP(SU_{ij})$. If PI_i and PI_j have no other conditional edges, the two nodes are compatible without any condition, that means, the label of the conditional edge is not important and the nodes can be colored with the same color. The labels of the set $\{SC_i, \dots, SC_{i+k}\}$ ($k \geq 2$) of conditional edges $SC_i \in SCE$ have to be taken into account only if the set of nodes $\{PI_j, \dots, PI_{j+l}\}$ ($l \geq 3$) connected by $\{SC_i, \dots, SC_{i+k}\}$ is

selected to be colored with the same color. This is only possible if the supercube of all the labels $l(SC_i, \dots, SC_{i+k})$ does not intersect any of the nodes from the set $\{PI_j, \dots, PI_{j+l}\}$.

Example VI. 2 Figure 17a illustrates a minimum cover with *CDEC*-implicants in a Karnaugh map. These *CDEC*-implicants were created by *CDEC*-compatible coloring of the nodes of the graph from Figure 17b. Color A describes a *CDEC*-implicant $abc\bar{d}$. Color B corresponds to a *CDEC*-implicant $\bar{c}da\bar{b}$. Color C corresponds to a *CDEC*-implicant $bca\bar{d}$. Color D corresponds to a *CDEC*-implicant $\bar{a}bcd$. The nodes of the graph correspond to the minterms in the Karnaugh-Map. The interrupted edges labeled with a cube are the conditional edges with their label given by the cube associated to the edge, and the continuous edges are the normal (unconditional) edges. An unconditional edge means that the respective nodes it links must be colored with different colors. A set of nodes that has the same color describes a *CDEC*-implicant. For instance, the nodes given by the minterms 0111, 0110 and 1111, respectively, are colored with a single color, color C. The supercube of all those nodes is $0111 \cup 0110 \cup 1111 = X11X$. The only cube in $OFF \sqcap X11X$ is 1110, which does not overlap with minterms 0111, 0110, and 1111. In this example the number of primes from the minimum cover equals the number of *CDEC*-implicants from the minimum coloring. However, in a similar example with excluded minterms 1100 and 1011, the minimum prime cover includes three implicants, while the minimum *CDEC* cover includes only two (those described by colors B and C above). As we see, the solution found in exact coloring has the minimum number of terms, but not necessarily the minimum number of literals. Other coloring would find the minimum literal cost solution $\bar{a}\bar{c}d + \bar{a}bc + ab\bar{c} + bcd$.

Figure 17. *CDEC* implicants and graph GCCC.

Example VI. 3 Using this method to function from Figure 18c will lead to coloring nodes C_1 and C_2 with color A and nodes C_3 and C_4 with color B, which leads to exact *CDEC* cover from Figure 18b.

Figure 18. Function to demonstrate that the minimum SOC cannot be obtained by factorization of primes

Example VI. 4 Results with better literal cost can be obtained when a node is allowed to be colored with several colors, which is called *multi-coloring*. Multi-coloring of nodes for the function from Figure 17a would create, among others, a solution $\bar{c}da\bar{\bar{b}} + abc\bar{\bar{d}} + cd\bar{a}\bar{\bar{b}} + a\bar{b}\bar{c}\bar{\bar{d}}$. This method creates larger *CDEC* implicants, therefore, the above solution has no hazard in ones. One can also observe that the minimum literal cost solution for the function from Figure 17a could not be found using a covering algorithm with prime *CDEC* implicants, since all the primes from this solution are included in some prime *CDEC* implicants. **This gives one more argument for our graph-coloring approach.**

While proving the exactness of minimum literal cost solution by the Compatible Multi-Coloring Algorithm is more involved, the exact minimization of the number of *CDEC* implicants is based on the following Theorem 1.

Theorem 1. If graph GCCC is created with minterms and the conditional coloring of GCCC has the minimum number of colors (is exact), then the *CDEC* covering created from this coloring has the exact minimum number of *CDEC*-implicants.

Figure 19. Color-Restriction-Vector of a node

The basic idea of GCCC coloring is to color successively the most restricted nodes with the most *normal edges* by choosing the color that results in the smallest number of Color-Restrictions for all other nodes.

VI.3.1 Definitions

The following definitions are presented for the Conditional Graph Coloring algorithm

Definition VI. 7: A **Color-Restriction** of a node PI_i is a labeling of PI_i with a complement of a color that PI_i is not allowed to be colored with. The Color-Restriction of a node PI_i results from a coloring of node PI_j ($i \neq j$) with c .

Definition VI. 8: A node PI_i of the Graph GCCC is associated with a **Color-Restriction-Vector** that includes all the Color-Restrictions of node PI_i . The Color-Restrictions result from the coloring of other nodes of the Graph and are the colors that node PI_i is not allowed to be colored with.

Definition VI. 9: A **Z-Color-Restriction** of a node PI_i is the labeling of PI_i with a Color-Restriction-Vector having Z elements.

Figure 19 shows an example of a node PI_i with a 3-Color-Restriction. Node PI_1 cannot be colored with colors A, C, and G. When node PI_i is colored in the

Figure 20. Color-Restriction-Matrix for a node

algorithm, the Color-Restriction-Vector is determined or updated for every adjacent node PI_j , $i \neq j$.

Definition VI. 10: The **Color-Restriction-Matrix** of a node PI_i is a matrix that includes as columns all the Z-Color-Restrictions of nodes PI_j , $i \neq j$ resulting from a coloring of PI_i , $i \neq j$ with the color c_i representing the rows of the matrix.

If the index of the column is denoted with k and the index of the row is denoted with c an element a_{ck} of the Color-Restriction-Matrix indicates how many k -Color-Restrictions result from the coloring of the node PI_i , $i \neq j$ with the color c .

Example VI. 5 Figure 20 gives an example of a Color-Restriction-Matrix of the node PI_1 . It assumed that node PI_1 can be colored with A, B or C. The edges in Figure 20 are *normal edges* and the Color-Restriction-Vectors of the nodes 2...6 are shown besides the nodes. If node 1 is colored with A, node 2 would be re-labeled to $\bar{A} \bar{D}$ (2-Color-Restriction), node 3 would be re-labeled to $\bar{A} \bar{C} \bar{D}$ (3-Color-Restriction), node 4 would be re-labeled to $\bar{A} \bar{B} \bar{D}$ (3-Color-Restriction),

node 5 would be re-labeled to $\overline{A} \overline{B} \overline{C} \overline{D}$ (4-Color-Restriction), and node 6 would be re-labeled to $\overline{A} \overline{C}$ (2-Color-Restriction). Thus new labeling would have one node with a 4-Color-Restriction, two nodes with a 3-Color-Restriction and two nodes with a 2-Color-Restriction - as in the first row of the matrix. Similarly, if node 1 is colored with C, node 2 would be re-labeled to $\overline{C} \overline{D}$, and node 4 would be re-labeled to $\overline{B} \overline{C} \overline{D}$. As it can be seen in Figure 20 the best coloring of node PI_1 is color C, because in this case the Color-Restriction-Matrix is minimal. The coloring of PI_1 with A results in a 4-Color-Restriction whereas the highest Color-Restriction for the coloring with B or C are 3-Color-Restrictions. The coloring with C is better than the coloring with B because C has in the matrix the 2-Color-Restriction of 1 while B has the 2-Color-Restriction of 2.

Definition VI. 11: The maximal Z-Color-Restriction of all colors c in the Color-Restriction-Matrix of a node PI_i is called the **Max-Z-Color-Restriction**.

Definition VI. 12: A Color-Set is the set of all colors that have been colored to the nodes of the graph GCCC in the algorithm.

Definition VI. 13: The CDEC-Compatible-Set is the set of all nodes or implicants that can form one CDEC-implicant.

Definition VI. 14: Each color c_j , being assigned to the nodes, is associated with a Color-Compatible-CDEC-Set which comprises of all the nodes of the graph GCCC, which are assigned that same color c_j . This means that, the supercube of the OFF-part of all cubes, SOFFP, does not intersect the ON cubes. The nodes in the Color-Compatible-CDEC-Set form one CDEC-implicant.

Definition VI. 15: The Probable-Color-Set comprises of all the allowed colors

that a node can be colored with. These are the colors that are present in the Color-Set and not in the Color-Restriction-Vector. the compliment of the intersection of the Color-Set and the Color-Restriction-Vector gives the Probable-Color-Set.

Definition VI. 16: A **Candidate Node** PI_i is a (non-colored) node of the graph GCCC that is a candidate for the next coloring.

The evaluation whether PI_i is a good candidate for the next coloring is based on the criteria:

- (1) the maximal Z-Color-Restriction of the candidate,
- (2) the degree of normal edges,
- (3) the Color-Restriction-Matrix,
- (4) the degree of conditional edges.

Definition VI. 17: A **Candidate Color** is a color c_i with which the Candidate Node PI_i can be colored. That means, c_i is not included in the Color-Restriction-Vector of PI_i . The Candidate Color is chosen from the Probable-Color-Set according to the following Candidate-Color-Criteria.

(1) The Candidate node, if colored with the chosen Candidate color, has to form a CDEC-Compatible-Set with all the other nodes in the Color-Compatible-CDEC-Set.

(2) If more than one color satisfies condition 1, then Choose the color which gives the minimal Color-Restriction-Matrix.

(3) If more than one color, then choose the lowest color.

In the graph coloring algorithm the selection of nodes to be colored next is done according to the minimal Max-Z-Color-Restriction. That means, that this is the first criterion to choose the node and the color. If there is an equality in

the Max-Z-Color-Restriction, then all the other Z-Color-Restrictions of the Color-Restriction-Matrix are taken into account as it is shown in the above example.

VI.3.2 The Conditional Graph Coloring Algorithm

step 1:

- Choose the first node according to the following criteria:
 - Select the first node to be colored with the first color by taking a node with the highest degree of normal edges.
 - If more than one node with the same degree in this set, then choose the node with the highest degree of conditional edges from this set.
 - If more than one node in this set, then choose the smallest node from this set.
- Assign the first color to the first node.

step 2:

- Make entries of the complement of the chosen color in the Color-Restriction-Vector of the nodes which are connected by a normal edge to the selected node.
- Make entry of the chosen node in the Color-Compatible-CDEC-Set of the chosen color.
- Remove the colored node from graph GCCC.

step 3:

- Choose the Next Candidate node according to the Candidate-Node-criteria.
- Determine the Probable-Color-Set for the Candidate Node.

step 4:

- Choose the next Candidate color from the Probable-Color-Set according to the Candidate-Color-Criteria and assign it to the candidate node. If all the colors in the Probable-Color-Set are exhausted, choose a new color.
- Enter the colored Candidate Node to the Color-Compatible-CDEC-Set for the Candidate color.
- Enter the Candidate Color to the Color-Set.

step 5:

- If there are still non-colored nodes in the graph GCCC, go to step 2.

VI.4 Conditional Graph Multi-Coloring Algorithm

The following definition is needed for the Multi-Coloring algorithm.

Definition VI. 18: Each node is assigned a Multi-Color-Vector, which comprises of all the allowed colors, that the node can be colored with. This means that the node has to form a CDEC-Compatible-Set with all the other nodes in the Color-Compatible-CDEC-Set of all the allowed colors.

VI.4.1 Algorithm

Only the step 4 from the previous algorithm is changed. Instead of just choosing

Figure 21. Implicants and graph GCCC.

one Candidate color, choose all candidate colors from the Probable-Color-Set, which when assigned to the candidate node, forms a CDEC-Compatible-Set with all the other nodes in the Color-Compatible-CDEC-Set of all colors. The colors are chosen from the Probable-Color-Set in the order of increasing minimal Color-Restriction-Matrix.

Form the Multi-Color-Vector for each Candidate node. The Multi-Color-Vector gives all the colors a node can be colored with.

Example VI. 6 Let us illustrate all the steps in each pass of the Conditional Graph Coloring algorithm with an example. The function is represented by the cubes shown

Color	Multi-Color-Vector	Nodes	Pass of the Algorithm								
			I	II	III	IV	V	VI	VII	VIII	IX
B	B,D	1	3,3 A	2,3 A	2,2 A,C	1,2 A,C	0,2 A,C	0,1 A,C	0,0 A,C	0,0 A,C	0,0 A,C
D	D	2	5,3 A	4,3 A,B	3,3 A,B,C	3,3 A,B,C					
B	B,D	3	2,5 A	1,5 A	1,4 A	1,3 A	0,2 A	0,2 A	0,1 A	0,0 A,C	0,0 A,C
D	C,D	4	4,5 A	3,5 A,B	2,5 A,B	2,4 A,B	1,3 A,B	1,2 A,B	1,2 A,B		
C	C	5	5,2 A	4,2 A,B	3,2 A,B	2,2 A,B,D	1,2 A,B,D	1,2 A,B,D			
C	C	6	6,2 A	5,2 A,B	4,2 A,B						
A	A	7	7,0	6,0 B	5,0 B,C	5,0 B,C,D	4,0 B,C,D				
A	A	8	7,1								
B	B	9	6,3	5,3 A							
B	B	10	3,4	3,3 A	2,2 A,C	2,1 A,C	2,1 A,C	1,1 A,C	0,1 A,C	0,1 A,C	
Nodes Colored			8,9	6	2	7	5	4	10	1	3
Color-Set			A,B	A,B,C	A,B,C,D	A,B,C,D	A,B,C,D	A,B,C,D	A,B,C,D	A,B,C,D	A,B,C,D
Probable-Color-Set			empty	empty	empty	A	C	C,D	B,D	B,D	B,D
Color-Compatible-CDEC-Set	A	8	8	8	8	8,7	8,7	8,7	8,7	8,7	8,7
	B	9	9	9	9	9	9	9	9,10	9,10,19	10,1,3
	C		6	6	6	6	6,5	6,5,4	6,5,4	6,5,4	6,5,4
	D			2	2	2	2	2	2	2	2

Figure 22. Entries made in each Pass of the algorithm

in the Karnaugh map in Figure 21a. Let the cubes 0x000, 1x000, 0x001, 11001, 0x011, 1x011, 0x110, 1x110, 01101, 1x100 be represented by 1,2,...,10 respectively. The graph GCCC with the normal edges represented by solid line and the conditional

Figure 23. Restriction-Vector-Matrix for node 4 during Pass VI

edges represented by dotted line is shown in Figure 21b and c respectively.

The table shown in Figure 22 shows the entries for the respective parameters for each node in each pass of the algorithm. Under each pass for each node, there are two rows, the first one represents the number of normal edges and the number of conditional edges in graph GCCC. For example the entry 4,5 in Pass I for Node 4 describes that, for node 4, the degree of normal and conditional edges in graph GCCC are 4 and 5 respectively. The second row represents the colors in the Color-Restriction-Vector. These are the colors that the node cannot be colored with. For example the entry A,B,D in Pass IV for node 5 describes that these are the colors that node 5 cannot be colored with.

Pass I:

1.
 - Node 8 is chosen as the first node out of nodes 7 and 8, as both have the same degree, and node 8 has a higher degree of conditional edges.
 - Assign new color A to the first selected node 8.
2.
 - Enter the complement of color A, $\bar{A}$, to the Color-Restriction-Vector of

- nodes 1,2,3,4,5,6 and 9, which are connected by a normal edge to node 8.
- Enter the chosen node 8 in the Color-Compatible-CDEC-set for color A.
 - Remove node 8 from graph GCCC.
3.
 - Nodes 1,2,3,4,5,6,9 all have same Z-Color-Restriction-Vector($\bar{A}$), and out of this set of nodes, node 6 and 9 have same degree of normal edges(6), and out of this set, node 9 has higher degree of conditional edges. Hence node 9 is chosen as the candidate node.
 - The Probable-Color-Set is empty for node 9 as all the colors in Color-Set(A) are in the Color-Restriction-Vector($\bar{A}$).
 4.
 - As the Probable-Color-Set is empty, a new color is chosen and assigned to the candidate node 9.
 - Enter the candidate node 9 to the Color-Compatible-CDEC-set for the Candidate Color B.
 - Enter the Candidate Color to the Color-Set.
 5. As there are still non-colored nodes in graph GCCC, go to step 2 and continue with next pass of the algorithm.

Pass II:

1. Skip step 1 and go to step 2.
2.
 - Enter the compliment of color B, $\bar{B}\bar{A}$, to the Color-Restriction-Vector of nodes 2,4,5,6, and 7, which are connected by a normal edge to node 9.
 - Enter the chosen node 9 in the Color-Compatible-CDEC-set for color B.

- Remove node 9 from graph GCCC.
3.
 - Nodes 2,4,5 and 6 all have same number of high Z-Color-Restriction-Vector(2), and out of this set, node 6 is chosen as the candidate node as it has a higher degree of normal edges(5).
 - The Probable Color-Set is still empty
 4.
 - As the Probable-Color-Set is empty, a new color C is chosen and assigned to the candidate node 6.
 - Enter the candidate node 6 to the Color-Compatible-CDEC-set for the Candidate Color C.
 - Enter the Candidate Color C to the Color-Set.
 5. As there are still non-colored nodes in graph GCCC, go to step 2 and continue with next pass of the algorithm.

Pass III:

1. Skip step 1 and go to step 2.
2.
 - Enter the compliment of color C $\bar{C}$, to the Color-Restriction-Vector of nodes 1,2,7 and 10, which are connected by a normal edge to node 6.
 - Enter the chosen node 6 in the Color-Compatible-CDEC-set for color C.
 - Remove node 6 from graph GCCC.
3.
 - Node 2 is chosen as the Candidate node as it has the highest Z-Color-Restriction-Vector(3).
 - The Probable Color-Set is still empty

4.
 - As the Probable-Color-Set is empty, a new color D is chosen and assigned to the candidate node 2.
 - Enter the candidate node 2 to the Color-Compatible-CDEC-set for the Candidate Color D.
 - Enter the Candidate Color D to the Color-Set.
5. As there are still non-colored nodes in graph GCCC, go to step 2 and continue with next pass of the algorithm.

Pass IV:

1. Skip step 1 and go to step 2.
2.
 - Enter the compliment of color D $\bar{D}$, to the Color-Restriction-Vector of nodes 5,6 and 7, which are connected by a normal edge to node 6.
 - Enter the chosen node 2 in the Color-Compatible-CDEC-set for color D.
 - Remove node 2 from graph GCCC.
3.
 - Node 7 is chosen as the Candidate node as it has the highest degree of normal edges.
 - The Probable Color-Set for node 7 has color A. These are the colors which are in the Color-Set and not in the Color-Restriction-Vector.
4.
 - As the color A from the Probable-Color-Set satisfies the Candidate-Color-Criteria, Candidate node 7 is colored with color A.
 - As the Color-Set already has Color A, this step is skipped.

5. As there are still non-colored nodes in graph GCCC, go to step 2 and continue with next pass of the algorithm.

Pass V:

1. Skip step 1 and go to step 2.
2.
 - Enter the compliment of color A, $\bar{A}$, to the Color-Restriction-Vector of nodes 1,3,4 and 5, which are connected by a normal edge to node 7.
 - Enter the chosen node 7 in the Color-Compatible-CDEC-set for color A.
 - Remove node 7 from graph GCCC.
3.
 - Node 5 is chosen as the Candidate node as it has the highest Z-Color-Restriction-Vector.
 - The Probable Color-Set for node 5 has color C.
4.
 - As the color C from the Probable-Color-Set satisfies the Candidate-Color-Criteria, Candidate node 5 is colored with color C.
 - As the Color-Set already has Color C, this step is skipped.
5. As there are still non-colored nodes in graph GCCC, go to step 2 and continue with next pass of the algorithm.

Pass VI

1. Skip step 1 and go to step 2.
2.
 - Make entry in the Color-Restriction-Vector
 - Make entry in the Color-Compatible-CDEC-set

- Remove node 5 from graph GCCC.
3.
 - Node 4 is chosen as the Next Candidate node.
 - The Probable Color-Set for node 4 has color B and D.
 4.
 - As the candidate node 4 forms a CDEC-Compatible-Set with the nodes in Color-Compatible-CDEC-set for both colors C and D, the Color Restriction Matrix is determined. The Color-Restriction-Matrix gives one 2-color-Restriction-Vector for color C and One 3-Color-Restriction-Vector for color D as shown in Figure 23. As color C gives the minimal Color-Restriction-Matrix, it is chosen and assigned to candidate node 4.
 - As the Color-Set already has Color C, this step is skipped.
 5. As there are still non-colored nodes in graph GCCC, go to step 2 and continue with next pass of the algorithm.

Pass VII

1. Skip step 1 and go to step 2.
2.
 - Make entry in the Color-Restriction-Vector
 - Make entry in the Color-Compatible-CDEC-set
 - Remove node 4 from graph GCCC.
3.
 - Node 10 is chosen as the Next Candidate node.
 - The Probable Color-Set for node 10 has color B and D.
4.
 - As the candidate node 10 forms a CDEC-Compatible-Set with the nodes in Color-Compatible-CDEC-set for both colors B and D, and as the same

minimal Color-Restriction-Matrix is obtained for both colors, the lowest color B is chosen and assigned to node 10.

- As the Color-Set already has Color C, this step is skipped.

5. As there are still non-colored nodes in graph GCCC, go to step 2 and continue with next pass of the algorithm.

In a similar fashion, nodes 1 and 3 are assigned color B in pass VIII and IX.

Finally, nodes 8 and 7 are assigned color A. Nodes 9, 10, 1 and 3 are assigned color B. Nodes 6, 5 and 4 are assigned color C and node 2 is assigned color D. The CDEC-implicants are illustrated in figure 24b.

Example VI. 7 Let us illustrate the multi-coloring algorithm with the same previous example. From Pass I to Pass V, whatever Candidate color is assigned to the Candidate node is entered in the Multi-Color-Vector, as there is either only one color in the Probable-Color-Set or no color. In Pass VI, there are 2 colors C and D in the Probable-Color-Set for the Candidate node 4. As both the colors satisfy the Candidate-Color-Criteria for node 4, both C and D are entered in the Multi-Color-Vector for node 4. Hence node 4 can be colored with both colors C and D. In Pass VII, there are 2 colors B and D in the Probable-Color-Set for node 10. But as node 10 does not form a CDEC-Compatible-Set with the nodes in the Color-Compatible-CDEC-Set for color D, after being colored with B, only color B is entered in the Multi-Color-Vector. In a similar fashion to Pass VI, nodes 1 and 3 are assigned colors B and D in the Multi-Color-Vector in Pass VIII and IX respectively.

Finally, nodes 8 and 7 are assigned color A. Nodes 9, 10, 1 and 3 are assigned color B. Nodes 6, 5 and 4 are assigned color C and nodes 1, 2, 3 and 4 are assigned

Figure 24. CDEC Implicants for Conditional Graph Coloring and
multi-coloring algorithm

color D. The CDEC-implicants formed as a result of multi-coloring is illustrated in figure 24c.

CHAPTER VII

SPLITTING OF CUBES

The graph coloring gives exact solution using minterms, and can give good solutions with arbitrary cubes, but, as we have seen in Example 1, primes should be split in order to create *CDEC* implicants for the exact minimum solution. The advantage of using minterms is the warranty of the optimum solution, the disadvantage is the size of the graph for functions with a high number of inputs. If the number of inputs is N , the number of minterms is of the order of 2^{N-1} , and the graph will be too big to construct. The advantage of using arbitrary disjoint cubes is the improved execution speed, but therefore, the minimum *CDEC* cover is lost. The question is, how to generate good (but larger than minterms) initial cubes f .

Example VII. 1 Function from Figure 25 has four primes, given by equation $f = \bar{a}be + \bar{c}de + ab\bar{c}d + bc\bar{d}e$. The cubes $\bar{a}be$, $\bar{c}de$, $ab\bar{c}d$, $bc\bar{d}e$ are represented in Figure 25a by cubes 1, 2, 3 and 4 respectively. None of them can be factored to *CDEC*-implicants. If prime implicants were used to generate prime *CDEC*-implicants the solution with four prime *CDEC*-implicants (also prime implicants in this case) would be generated. However, as seen in Figure 25b, the same function can be realized with only three *CDEC*-implicants A, B and C. One of those *CDEC* implicants (shown in Figure 25b); $A = \bar{a}\bar{c}e(b + d)$ is created by factorizing product implicants $\bar{a}b\bar{c}e$ and $\bar{a}\bar{c}de$ which are both not prime implicants. Two other prime *CDEC* implicants are created by factoring one prime implicant and one product implicant being not a prime. For

(a)

(b)

Figure 25. Example for split

instance, $CDEC$ implicant $C = bce(\bar{a} + \bar{d})$ is created from prime implicant 4, $bc\bar{d}e$ and product implicant $\bar{a}bce$ included in prime implicant 1, $\bar{a}be$. One can observe that essential prime implicants 3, $ab\bar{c}d$ and 4, $bc\bar{d}e$ are used to create prime $CDEC$ -implicants $B = a\bar{c}d(b + e)$ and $C = bce(\bar{a} + \bar{d})$, respectively. The solution with three $CDEC$ implicants is: $f = \bar{a}\bar{c}e(b + d) + a\bar{c}d(b + e) + bce(\bar{a} + \bar{d})$.

Example VII.1 suggest that cube-splitting operations have to be performed on the input functions before invoking the graph coloring routine, and that essential and secondary essential primes play a role in splitting. We present a heuristic algorithm which splits the given input cubes and generates a *set of split cubes*. This new set of cubes, when used as nodes in graph GCCC for the exact graph coloring algorithm gives the exact solution in all tested functions, but we have no proof of

optimality.

VII.1 Initial-Split Cube Operation

A new cube operation, **Initial-Split**, is defined, which is needed for the cube splitting algorithm. Initial-split operation splits the candidate node P with respect to r cubes in the r -d1-set.

cube $P = (p^1, p^2, \dots, p^n)$, and set $N = \{1, 2, \dots, n\}$ where $p^i, i \in N$ are the literals of P .

r -d1-set = $\{Q_1, Q_2, \dots, Q_r\}$, and set $R = \{1, 2, \dots, r\}$ where $Q_j, j \in R$ are the distance-1 nodes to the candidate node.

$Q_j = \{q_j^1, q_j^2, \dots, q_j^n\}$, where $j \in R$ where $q_j^i, i \in N, j \in R$ is the i th literal of the j th cube in r -d1-set.

Let us define, for all $Q_j, j \in R$ in r -d1-set,

set $Lq^i = \{q_1^i, q_2^i, \dots, q_r^i\}$, where $i \in N$ represents the i th literal of all $Q_j, j \in R$.

The Initial-split cube operation is defined as follows: For those $p^i, i \in N$ of cube P , which has a don't care X, if the set Lq^i has different truth values, then the don't care X in $p^i, i \in N$ of cube P is split into 0 and 1.

Example VII. 2 The function f is given by the equation $f = \bar{a}b\bar{c}\bar{d} + \bar{c}d + abcd$. The cubes $\bar{a}b\bar{c}\bar{d}$, $\bar{c}d$, and $abcd$ are represented by cubes 1, 2 and 3 respectively, in Figure 26a. Let us split cube 2 = P with respect to cube 1 and 3 using the Initial-split cube operation. Cube P can be split in two ways with respect to cube 1 and in two ways with respect to cube 3. Only one of these ways, Figure 26b, created by Initial-split, is good for both cubes 1 and 3 in the sense that the three cubes 1,2,3

Figure 26. Example for Initial Split.

will be replaced with two cubes.

The initial Split for several possible combinations are shown in Figure 27.

VII.2 Definitions

Definition VII. 1: The set **r-d1-set** is a set of nodes, which are distance-1 from the candidate node in the distance-1 graph, and are not compatible or conditionally compatible to the candidate node in GCCC.

Definition VII. 2: **Initial-split-set** of a node is a set of cubes generated by splitting a node using the Initial-split cube operation.

Definition VII. 3: The **Consensus Set** is formed by the consensus of each of the cubes in Initial-split set and r-d1-set.

The **New-list** consists of the set of cubes generated while splitting a node. The input nodes of the graph GCCC form the **original-list**, which gets updated with the split cubes, when a node is split. The algorithm uses also a (Hamming) distance-1 graph with edges between nodes corresponding to cubes of distance 1.

Figure 27. Initial Split for several possible combinations of occurrence of input cubes.

VII.3 Algorithm to create Split Cubes

step 1:

- If the input cubes are non-disjoint,

For all the cubes that are non-disjoint, sharp the cube which has lower degree of distance-1 edges from the cube that has a higher de-

Figure 28. Example for Cube Split Algorithm.

gree of distance-1 edges, until there are no more non-disjoint cubes.

- If the input cubes are disjoint, go to step 2.

step 2:

- Select the node which has the highest degree of distance-1 edges in GCIC.
 - If the selected node is a minterm, then choose another node with the next highest number of distance-1 edges in the Distance-1 graph.
 - If all the distance-1 nodes of the selected node have compatible edges and/or conditional edges in GCIC, then discard that selected node and choose another node with the next highest number of distance-1 edges in the Distance-1 graph.
- If more than one node, then choose the node with the highest degree of incompatible edges in GCIC.
- If more than one node, then choose the node with the lowest degree of compatible edges in GCIC.

step 3:

- The candidate node has to be split with respect to the Distance-1 nodes connected by distance-1 edges in GCIC. Disregard the Distance-1 nodes connected to the candidate node by compatible or conditionally compatible edges and form the r-d1-set.

step 4:

- If the r-d1-set consists of two cubes or less
 - Perform the Initial-split cube operation on the candidate node with respect to the cubes in the r-d1-set and form the Initial-split-set.
 - skip step 5 and go to step 6.

- If the r-d1-set consists of more than two cubes
 - Perform the Initial-split cube operation on the candidate with respect to the cubes in the r-d1-set and form the Initial-split-set.
 - If all the cubes in Initial-split-set are minterms, then go to step 6, otherwise go to step 5.

step 5:

- Find the consensus for each of the cubes in the Initial-split-set and each of the cubes in the r-d1-set and form the consensus-set.
- For each cube in the Initial-split-set
 - Sharp each cube in the consensus-set with the selected cube in the Initial-split-set and add the resultant cubes to the New-list.
 - Find the intersection of each cube in the consensus-set with the selected cube in the Initial-split-set and add the resultant cubes to the New-list.
- Remove the similar cubes in the New-list and add to the Original-list.

step 6:

- continue step 2 until there are no more non-split nodes with distance-1 edges that are not covered by compatible or conditional edges.

Example VII. 3 Let us illustrate the cube splitting algorithm with an example.

The input cubes are given by the function: $f = \bar{a}\bar{c}d + b\bar{c}\bar{d} + b\bar{c}d + acd + ac\bar{d}$. Let us denote the cubes $\bar{a}\bar{c}d$, $b\bar{c}\bar{d}$, $b\bar{c}d$, acd , $ac\bar{d}$ by 1, 2, 3, 4 and 5 respectively. The

function f is shown in Figure 28a. The distance-1 graph and the GCCC for function f are shown in Figure 28b and 28c. As the input cube set consists of non-disjoint cubes, step 1 is first executed. Cubes 1 and 2 are non-disjoint and as cube 2 has a higher degree of distance-1 edges (3 distance-1 edges) in the distance-1 graph than cube 1 (2 distance-1 edges), cube 1 is sharpened from cube 2 giving the result cube 21: $cube1 \# cube2 = 0X00 \# X100 = 1100 = cube21$ Figure 28d shows the input cube list with all the cubes being disjoint cubes. The respective distance-1 graph and the GCCC graph are shown in Figure 28e and 28f. Nodes 21 and 3 have the highest degree of distance-1 edges (3 distance-1 edges) in the distance-1 graph shown in Figure 28e. As node 21 is a minterm, that node is discarded and node 3 is chosen as the candidate node. Now node 3 is connected to nodes 1, 2 and 4 by distance-1 edges. But as node 2 is connected to node 3 by a compatible edge, node 2 is discarded and nodes 1 and 4 form the r-d1-set. Hence the candidate node 3 has to be split with respect to nodes 1 and 4 from the r-d1-set. Performing the Initial-split cube operation, gives cubes 31 and 32. It can be observed in Figure 28i that nodes 21, 32, 4 and 5 can be colored with color A, and nodes 1 and 31 with color B, which leads to a solution with 2 *CDEC*-implicants.

CHAPTER VIII

MULTIPLE OUTPUT AND INCOMPLETELY SPECIFIED FUNCTIONS

This chapter illustrates the conversion of multi-output function to single-output function and the advantages of using SOCMIN for strongly unspecified function.

VIII.1 Multi-output Functions

The *CDEC* Minimization based on Graph Coloring as described in Chapter VI is applied to a single-output Boolean function f_i , specified by the ON-set $ON(f_i)$ and the OFF-set $OFF(f_i)$. In order to minimize multi-output functions, there are basically two possibilities. The first possibility is to minimize each single function concurrently and use certain terms in several functions. The minimization is only quasi-optimal. Another possibility is to perform a multi-output to single-output transformation, followed by a single-output minimization, and a retransformation to the multi-output function. The second approach is applied in the *CDEC*-minimization algorithm.

When the graph GCIC is constructed for a Boolean function f_i , only the ON- and OFF-set of f_i are used. Therefore, a multi-output to single-output transformation that produces many don't cares can be applied. Miller presented such a transformation (attributed to Muller) in [14] that transforms a multi-output function with m input and n output variables to a single-output function with $m+n$ input variables. The method is not applicable for multi-output function that are repre-

Figure 29. Example of the Muller Transform.

sented by the ON- and DC-set (minimization using Espresso), because it creates $m * (2^n - n)$ additional DC-minterms. This disadvantage of the *Muller Method* is not relevant in our *CDEC* minimization, because the DC-set is not used during the algorithm. The set of input variables is given by $\{i_1, i_2, \dots, i_m\}$ and the multi-output function F is represented by the set $F = \{f_1, f_2, \dots, f_n\}$, where f_i is a single-output Boolean function.

An example of the Muller Transform applied on the set $F = \{f_1, f_2\}$ ($m = 2$ outputs) with $n = 3$ input variables is shown in Figure 29. The transformation results in a single-output function $\tilde{F}$ with $m + n = 5$ input variables. The rows

of the Karnaugh map are the m input variables of F , and the columns are the additional input variables of the function $\tilde{F}$ resulting from the n outputs of the set of functions F . The columns of the Karnaugh Map that are hot encoded (columns 01 and 10) represent the single-output functions $f_i \in F$ (f_1 and f_2), and the other columns are filled with DCs. Hot encoded means, that only one literal of the cube associated to a column is "1", all the other literals are "0". If the i th literal of a hot encoded column is "1", then this column corresponds to the function f_i of the set F .

The Graph-Coloring-based *CDEC* minimization generates the *CDEC* implicants that are shown in Figure 29. They cover the function $\tilde{F}$. For the covering of the set of functions F only the initial input variables $i_1, \dots, i_3$ of the *CDEC* implicants have to be considered. Now, the function $\tilde{F}$ is retransformed to a multi-output function. The *CDEC* implicants that cover a hot encoded column have to be used for the corresponding function f_i . By using implicants we mean realizing function f_i as an OR of all *CDEC* implicants that cover the corresponding column.

In the example of Figure 30 the *CDEC* implicants are $\bar{i}_1\bar{i}_2\bar{i}_3$, $i_2(\overline{i_1i_3})$, and $i_1(\overline{i_2i_3})$. The additional variables o_1 and o_2 are not included as literals in the *CDEC* implicants. The first *CDEC* implicant has to be used for both functions f_1 and f_2 , the second only for f_1 , and the third only for f_2 . The result of the *CDEC* minimization is shown in Figure 30. Three *CDEC* gates are necessary in order to realize the functions f_1 and f_2 .

$$f_1 = \bar{i}_1\bar{i}_2\bar{i}_3 + i_2(\overline{i_1i_3}) \quad (9)$$

$$f_2 = \bar{i}_1\bar{i}_2\bar{i}_3 + i_1(\overline{i_2i_3}) \quad (10)$$

Figure 30. Realization of the example.

Figure 31. CDEC implicants for Incompletely specified function

In the example of Figure 30 the *CDEC* implicants are $\bar{i}_1\bar{i}_2\bar{i}_3$, $i_2(\bar{i}_1i_3)$, and $i_1(\bar{i}_1i_2i_3)$. The additional variables o_1 and o_2 are not included as literals in the *CDEC* implicants. The first *CDEC* implicant has to be used for both functions f_1 and f_2 , the second only for f_1 , and the third only for f_2 . The result of the *CDEC* minimization is shown in Figure 30. Three *CDEC* gates are necessary in order to realize the functions f_1 and f_2 .

VIII.2 Incompletely Specified Function

Strongly Unspecified functions find important applications in Machine Learning and KDD. The problems used in Machine Learning have a high percent of don't cares (almost 99.9 percent) and these have to be an important consideration for designing algorithms. The algorithm used in SOCMIN have been designed with the Strongly Unspecified functions, defined by ON and OFF sets. Let us illustrate how the don't cares affect the solution for *CDEC* implicants generated by SOCMIN and the prime implicants generated by Espresso.

Let the function shown in Figure 31 be given by the minterms $1, \dots, 16$ and don't cares $17, \dots, 20$ respectively. As none of them can be factored, the number of implicants generated by Espresso would be 16 implicants given by minterms $1, \dots, 16$. But, the SOCMIN algorithm takes into account the don't cares, and gives out 8 *CDEC* implicants, A,B,C,D,E,F,G and H, which is 50 percent reduction in the number of implicants given by Espresso. Hence, when the number of variables is large, the reduction in the number of terms given by SOCMIN would be considerably lower than Espresso.

CHAPTER IX

RESULTS AND COMPARISON

We created a tool, called SOCMIN, and tested it extensively on MCNC and Machine Learning benchmarks (which have very many don't cares). The goal of our experiments with SOCMIN was to answer the following questions:

1. How much improvement in implicants or in literals is gained by using the *CDEC* implicants instead of prime implicants.
2. How good is our heuristic minimization strategy based on cube splitting comparing to the exact solutions.
3. What is the improvement for Machine Learning examples, when data is randomly removed, in terms of implicants, literals and DFC.

IX.1 Single Output MCNC benchmarks.

SOCMIN was tested on several Single Output MCNC benchmarks, and compared with the results generated from Espresso and the disjoint cube producing program Spectra [21]. Figure 32 shows the results for those Single Output MCNC examples for which SOCMIN generates better solution than Espresso. The number of terms generated by Espresso, the disjoint cube producing program Spectra and the total number of minterms for the examples are shown in columns 5, 7 and 3 of Figure 32 respectively. Also, the number of *CDEC* implicants using minterms,

MCNC Examples	input literals	Minterms		Espresso		Spectra disjoint		cdmin	Time (sec)
		Terms	cdec	Terms	cdec	Terms	cdec		
5x01	7	76	4	7	4	7	4	4	0.03
5x5	7	88	6	10	7	10	7	6	0.09
bw10	5	28	2	3	3	3	3	2	0.03
bw18	5	11	4	6	5	6	5	4	0.04
bw20	5	18	4	6	5	6	5	4	0.05
duke10	22	-	-	14	10	16	9	9	0.51
f501	8	244	13	23	14	23	14	13	0.45
f52	8	216	12	18	12	18	12	12	0.26
f53	8	192	8	14	9	14	9	8	0.16
f54	8	176	6	10	7	10	7	6	0.09
misex20	10	8	7	7	7	7	7	7	0.11
misex21	6	52	5	11	6	11	6	5	0.10
misex22	6	20	4	6	5	7	5	4	0.05
misex23	6	22	4	6	5	8	5	4	0.06
misex48	6	52	5	11	7	12	7	5	0.09
misex54	6	52	5	11	6	12	6	5	0.11
misex55	6	20	4	6	5	7	5	4	0.05
misex56	6	22	4	6	5	8	5	4	0.06
rd533	5	20	7	10	10	8	10	7	0.09
rd731	7	84	22	42	25	10	25	22	2.4
rd733	7	280	13	35	25	42	13	13	1.2
sao23	10	1664	9	22	9	31	10	9	0.76
sao24	10	672	9	21	12	33	9	9	0.85
z42	7	128	15	28	16	28	16	15	0.78

Figure 32. Single output MCNC benchmarks with lower terms than Espresso.

input cubes generated by Espresso and the disjoint cube producing program Spectra are shown in columns 4, 6 and 8 respectively. Finally, the number of *CDEC* im-

MCNC Example	Number of CDEC implicants				
	minterm	espresso	spectra disjoint	esop to sop	Cube split alg
5x01	4	4	4	13	4
5x10	2	2	2	4	2
5x2	6	6	6	15	6
5x3	12	12	12	14	12
5x4	9	9	9	10	9
5x5	6	7	7	10	6
5x6	4	4	4	4	4
5x7	2	2	2	3	2

(a)

MCNC Example	Number of CDEC implicants				
	minterm	espresso	Spectra disjoint	esop to sop	Cube split alg
f21	2	2	2	2	2
f22	2	2	2	2	2
f23	2	2	2	2	2
f24	2	2	2	2	2
f501	13	14	14	15	13
f52	12	12	12	14	12
f53	8	9	9	11	8
f54	6	7	7	10	6
f55	4	4	4	4	4
f56	2	2	2	3	2
f57	2	2	2	2	2

(b)

MCNC Example	Number of CDEC implicants				
	minterm	espresso	Spectra disjoint	esop to sop	Cube split alg
bw01	4	4	4	5	4
bw10	2	3	2	5	2
bw11	2	2	3	2	2
bw12	3	3	3	3	3
bw13	3	3	3	3	3
bw14	4	4	4	4	4
bw15	2	2	2	2	2
bw16	3	3	3	3	3
bw17	3	3	3	3	3
bw18	4	5	5	4	4
bw19	3	4	3	3	3
bw2	2	2	2	3	2
bw20	4	5	5	4	4
bw21	4	4	4	4	4
bw23	5	5	5	5	5
bw24	5	5	5	5	5
bw25	2	2	2	2	2
bw26	4	4	4	4	4
bw27	3	3	3	3	3
bw3	3	3	3	3	3
bw4	4	4	4	4	4
bw5	3	3	3	3	3
bw6	2	3	3	2	2
bw7	4	4	4	4	4
bw8	2	2	2	2	2
bw9	3	3	3	3	3

(c)

Figure 33. Single output MCNC benchmarks for 5x, f, bw.

licants generated by SOCMIN, using the Cube Splitting algorithm, starting with input cubes generated from Espresso is shown in column 9.

As we can see, in all cases, the results of SOCMIN using the Cube Splitting algorithm, have the same number of *CDEC* implicants as in the exact minimum

MCNC Example	Number of CDEC implicants				
	minterm	espresso	Spectra disjoint	esop to sop	Cube split alg
misex60	2	2	2	2	2
misex61	2	2	2	2	2
misex62	7	7	7	7	7
misex63	7	7	7	7	7
misex64	4	4	4	4	4
misex65	12	12	12	12	12
misex66	3	3	3	4	3
misex67	2	2	2	3	2
misex68	2	2	2	3	2
misex69	2	2	2	3	2
misex70	4	4	4	4	4
misex71	3	3	3	4	3
misex72	2	2	2	3	2
misex73	2	2	2	3	2
misex74	2	2	2	3	2
misex75	4	4	4	4	4

(a)

MCNC Example	Number of CDEC implicants				
	minterm	espresso	Spectra disjoint	esop to sop	Cube split alg
sao21	-	8	8	8	8
sao22	-	20	20	20	20
sao23	-	9	9	10	9
sao24	-	12	11	12	11

(b)

MCNC Example	Number of CDEC implicants				
	minterm	espresso	Spectra disjoint	esop to sop	Cube split alg
misex20	7	7	7	2	2
misex21	5	6	6	5	5
misex22	4	5	5	6	4
misex23	4	5	5	4	4
misex24	2	2	2	3	2
misex25	5	5	5	6	5
misex26	4	4	4	5	4
misex27	4	4	4	4	4
misex41	4	4	4	5	4
misex42	2	2	2	3	2
misex43	2	2	2	3	2
misex44	2	2	2	2	2
misex45	3	3	3	4	3
misex46	3	3	4	4	3
misex47	4	4	4	4	4
misex48	5	7	7	9	5
misex49	4	5	5	6	4
misex50	4	5	5	6	4
misex51	2	2	2	3	2
misex52	5	5	5	6	5
misex53	4	4	4	5	4
misex54	5	6	6	9	5
misex55	4	5	5	6	4
misex56	4	5	5	7	4
misex57	5	5	5	5	5
misex58	4	4	4	6	4
misex59	4	4	4	4	4

(c)

Figure 34. Single output MCNC benchmarks for misex, sao.

generated with minterms. Also, the number of *CDEC* implicants generated by SOCMIN is always better or equal to the number of prime implicants generated by Espresso and the program is substantially fast.

The number of *CDEC* implicants generated using various types of input cubes

MCNC Example	Number of CDEC implicants				
	minterm	espresso	Spectra disjoint	esop to sop	Cube split alg
duke02	-	7	7	12	7
duke10	-	10	9	18	9
duke11	-	5	5	6	5
duke13	-	13	13	16	13
duke15	-	7	7	10	7
duke17	-	13	13	22	13
duke18	-	5	5	7	5
duke21	-	7	7	12	7
duke22	-	4	4	4	4
duke23	-	4	4	4	4
duke25	-	5	5	5	5
duke26	-	2	2	2	2
duke27	-	1	1	2	1
duke28	-	9	10	11	9
duke29	-	10	10	12	10
duke3	-	10	10	14	10
duke4	-	5	5	6	5
duke6	-	7	7	7	7
duke7	-	13	13	22	13
duke8	-	5	5	5	5

(a)

MCNC Example	Number of CDEC implicants				
	minterm	espresso	Spectra disjoint	esop to sop	Cube split alg
rd531	3	3	3	3	3
rd532	16	16	16	16	16
rd533	7	10	10	9	7
rd731	22	25	25	22	22
rd733	13	15	13	14	13

(b)

MCNC Example	Number of CDEC implicants				
	minterm	espresso	Spectra disjoint	esop to sop	Cube split alg
cm152a	-	8	8	12	8
con11	4	4	4	4	4
con12	4	4	4	6	4

(c)

MCNC Example	Number of CDEC implicants				
	minterm	espresso	Spectra disjoint	esop to sop	Cube split alg
majority.esp	3	3	3	8	3
majority.pla	3	3	3	8	3

(d)

MCNC Example	Number of CDEC implicants				
	minterm	espresso	Spectra disjoint	esop to sop	Cube split alg
vg22	-	8	8	12	8
vg24	-	8	8	12	8
vg25	-	32	32	36	32
vg27	-	24	26	30	24
vg28	-	4	4	6	4

(e)

MCNC Example	Number of CDEC implicants				
	minterm	espresso	Spectra disjoint	esop to sop	Cube split alg
z41	8	8	8	9	8
z42	16	16	16	17	15
z43	8	8	8	8	8
z44	4	4	4	4	4

(f)

Figure 35. Single output MCNC benchmarks for con, duke, majority, rd, vg.

and input cubes generated by the Cube Splitting algorithm for the Single Output examples are also tested. The different kinds of input cubes used are minterms,

MCNC Examples	Variables		Minterms		Espresso		Spectra disjoint		cdmin
	input	output	Terms	cdec	Terms	cdec	Terms	cdec	
5xpl	7	10	-	-	65	37	73	36	36
b12	15	9	-	-	43	27	52	27	27
con1	7	2	161	7	9	8	10	8	7
cu	14	11	-	-	19	19	19	19	19
f51m	8	8	-	-	76	20	76	20	20
il	25	16	-	-	28	20	30	20	20
inc	7	9	305	29	30	29	34	34	29
misex1	8	7	547	12	12	12	14	12	12
misex2	25	18	-	-	28	27	28	27	27
rd53	5	3	42	26	31	29	31	27	26
rd73	7	3	185	20	127	22	127	20	20
rd84	8	4	665	30	255	30	255	30	30
sao2	10	4	754	13	58	13	93	13	13
squar5	5	8	85	22	25	25	26	23	22
temp	5	3	33	7	13	7	16	7	7
vg2	25	8	-	-	110	38	219	36	36

Figure 36. Multi output MCNC benchmarks.

prime implicants generated by Espresso, disjoint cubes generated by Spectra and the SOP cubes generated from ESOP cubes (from Exorcism) using the Spectra ESOP to SOP algorithm. The goal of this testing is to determine which type of input cubes gives more optimal solution when used by the graph coloring algorithm in SOCMIN. The results of this testing are shown in the Tables in Figures 33, 34, 35. The columns minterm, espresso, Spectra disjoint, esop to sop, and Cube Split alg in the Tables in Figures 33, 34, 35 give the number of *CDEC* implicants generated when the cubes generated by the respective programs are used as input cubes to the graph

coloring algorithm in SOCMIN. As we can see, in all cases, the *CDEC* implicants generated using the input cubes given out by the Cube Splitting algorithm is always the lowest and this equals the the number of *CDEC* implicants generated using the minterms as the input cubes. This proves that the Cube Splitting algorithm, in conjunction with the graph coloring algorithm, always gives the same number of *CDEC* implicants as in the exact minimum generated with minterms.

IX.2 Multiple Output MCNC benchmarks.

SOCMIN was tested for the Multiple Output MCNC benchmarks and compared with results from Espresso and the input cubes generated from Spectra disjoint program. The results of the Multiple Output MCNC examples is shown in Figure 36. Figure 36 displays from left to right the columns showing the number of input and output variables, the number of minterms and the number of *CDEC* implicants generated using minterms as input cubes, the number of prime implicants given by Espresso and the number of *CDEC* implicants generated using these prime implicants as input cubes, the number of disjoint cubes given by Spectra disjoint program and the number of *CDEC* implicants generated using these disjoint cubes as input cubes and finally the number of *CDEC* implicants generated by SOCMIN with Cubes Splitting done on prime implicants given by Espresso and using the graph coloring routine.

In all cases, the number of *CDEC* implicants is less than or equal to the number of prime implicants generated by Espresso and the disjoint cubes produced by Spectra disjoint program. Some functions, *rd73*, *rd84*, *vg2*, demonstrate that *CDEC* cover is substantially smaller than the SOP cover - in case of *rd84* there are only 30 *CDEC* implicants in the cover found by SOCMIN versus 255 primes in the

Figure 37. CDEC gate and its NAND representation.

cover from Espresso. Also, the number of *CDEC* implicants generated by SOCMIN with Cube Split always equals the *CDEC* implicants generated using minterms as input.

IX.3 Delay of CDEC

Sum of CDEC gates or the CDEC/OR structure can be represented in CMOS using NAND gates. The structure of CDEC gate is the AND of AND and NAND gate ($CDEC = P \cdot \overline{Q}$, where P and Q are product terms). The AND gate which forms the product P , can be represented with a NAND gate feeding an inverter. The product term $\overline{Q}$ is the output of another NAND gate in the first level. These two product terms P and $\overline{Q}$ are then fed to another NAND gate which then feeds an inverter to get the CDEC implicant, $CDEC = P \cdot \overline{Q}$. Thus, we could represent the CDEC gate with NAND gate and inverters, as shown in Figure 37b.

The sum of CDEC gate could be formed by feeding the output of all CDEC gates to another OR gate. By replacing the inverter and the OR gate in the last two levels with a NAND gate, we get a sum of CDEC gate structure with only NAND gates and inverters, as shown in Figure 38.

The classical AND/OR structure, as shown in Figure 39a, can be represented

Figure 38. Sum of CDEC in NAND representation.

by implementing the AND gate as a NAND gate feeding an inverter and the OR gate as a NOR gate feeding an inverter, as shown in Figure 39b. Also, the AND/OR can be realized by an array of NAND gates feeding another NAND gate, as shown in Figure 39c.

Let us discuss the delay and compare the delay caused due to the implementation of the boolean function in AND/OR structure using SOP form and the CDEC/OR structure using SOC form.

Figure 39. AND OR structure and its NAND and NOR representation.

Consider the boolean function given by $F = ab(c+d+e) + uvw(x+y+z)$. There are two CDEC implicants and can be realized by a sum of two CDECs, as shown in Figure 40a. The same function, when represented by SOP form, with 6 terms, can be implemented with AND/OR structure, using NAND and NOR gates, as shown in Figure 40b, or with only NAND gates, as shown in Figure 40c.

We observe that the function F , when implemented by sum of CDEC struc-

Figure 40. Example function in AND OR and CDEC OR structure.

ture has gates with lower fan-in and fan-out, compared to the AND/OR implemen-

tation using SOP. On an average, gates with high fan-out have larger transistors, which increases the capacitive load causing more delay in the circuit. Gates with high fan-in leads to more transistors in series, causing more delay. When two identical transistors are connected in series, the rise or fall time will approximately double that for a single transistor with the same capacitive load. When the fan-in of the gates increase, it might become necessary to increase the number of levels, which have a drastic impact leading to higher delay. As it is advisable to use minimum sized gates on high fan-out nodes to minimize the load presented to the driving gate, and use gates with fan-out below 5-10, in addition to keeping rising and falling edges sharp, we see that the AND/OR structure has more disadvantages causing more delay, compared to sum of CDEC structure.

In PLA/PAL structures, the circuit is more wired with more connection causing an increase in delay. Also, the delay depends on the output parasitic capacitance of the circuit, which include wire capacitance and transistor parasitic capacitance. As the PLA/PAL are more wired with larger circuit area, this leads to higher delay.

IX.4 Machine Learning examples

SOCMIN was tested for Machine Learning examples which have a high percentage of don't cares. As the algorithms used in SOCMIN have been designed for Strongly Unspecified functions, defined by ON and OFF sets, we achieved good results when comparing to Espresso and Exorcism. For the Machine Learning examples, the number of terms, literals and DFC for Espresso, Exorcism and SOCMIN was tested and compared and the results are shown in Figures 43 and 44. Let us illustrate how to calculate the DFC for a *CDEC* implicant in the following subsection IX.4.1.

Figure 41. Decomposition to calculate DFC.

Figure 42. DFC for a CDEC gate

IX.4.1 DFC for SOC

DFC is calculated by adding the cardinalities of each of the subfunctions in the decomposition. For arbitrary non-decomposable block in Curtis Decomposition the DFC of the block is calculated as 2^k where k is the number of inputs to the block. In "gate-based" minimizers such as Espresso and EXORCISM it is then fair to assume that a DFC of a decomposable gate (such as AND, OR or EXOR) is equal to the

FUNCTION	ON	DC	TERMS			LITERALS			DFC			TIME (sec)
			Espresso	Exorcism	Socmin	Espresso	Exorcism	Socmin	Espresso	Exorcism	Socmin	
add0	120	0	15	15	8	64	68	42	252	268	164	0.23
add2	128	0	16	5	8	68	10	44	268	36	180	0.22
add4	128	0	2	2	2	4	2	4	12	4	12	0.50
ch15f0	88	0	12	9	10	60	37	54	236	156	244	0.14
ch176f0	64	0	2	2	2	6	4	6	20	12	20	1.04
ch177f0	128	0	2	2	2	4	2	4	12	4	12	0.50
ch22f0	48	0	6	6	4	30	30	24	116	116	92	1.44
ch30f0	64	0	7	7	5	32	32	40	124	124	156	5.98
ch47f0	52	0	9	7	7	48	34	40	188	132	176	1.66
ch52f4	50	0	18	15	15	108	93	96	428	392	428	1.54
ch70f3	24	0	5	5	4	28	28	24	108	108	104	1.73
ch74f1	39	0	10	10	8	58	58	55	228	248	224	1.88
ch83f2	38	0	17	13	11	115	77	82	456	304	344	1.65
ch8f0	224	0	7	7	6	16	28	15	60	124	60	0.06
contains_4_ones	70	0	70	40	70	560	224	560	2236	968	2236	14.7
greater_than	120	0	15	15	8	64	72	42	252	284	168	0.38
interval1	58	0	16	16	13	96	98	90	380	388	380	3.49
interval2	128	0	22	19	14	110	88	92	436	308	380	4.09
kdd1	160	0	2	2	2	3	4	3	8	12	16	0.03
kdd10	120	0	8	4	4	28	10	18	108	40	68	0.09
kdd2	24	0	2	2	1	8	8	5	28	28	16	0.01
kdd3	80	0	2	2	2	5	6	5	16	20	24	0.02
kdd4	128	0	1	1	1	1	1	1	0	0	4	0.06
kdd5	106	0	4	6	4	13	27	13	48	108	13	0.11
kdd6	240	0	4	2	1	4	4	4	12	12	12	0.76
kdd7	175	0	4	15	4	8	64	8	28	268	28	0.85
kdd8	64	0	2	2	2	6	4	6	20	12	28	0.14

Figure 43. Machine Learning examples

FUNCTION	ON	DC	TERMS			LITERALS			DFC			TIME (sec)
			Espresso	Exorcism	Socmin	Espresso	Exorcism	Socmin	Espresso	Exorcism	Socmin	
kdd9	64	0	8	4	4	36	12	24	140	44	100	0.76
majority_gate	93	0	56	34	20	280	189	151	1116	732	612	10.3
modulus_2	43	15	10	9	6	45	52	37	160	140	152	1.5
mux8	128	0	4	4	4	12	12	12	44	44	60	0.57
pal	16	0	16	16	16	128	72	128	508	268	508	0.30
pal_dbl_output	160	0	29	22	23	151	92	136	600	412	600	2.95
pal_output	118	0	46	37	40	291	206	262	1156	820	1168	12.7
parity	128	0	128	8	128	1024	8	1024	4092	36	4092	22.8
remainder2	88	15	23	19	16	137	108	118	528	416	492	2.97
rnd_m1	1	0	1	1	1	8	8	8	28	28	28	1.79
rnd_m10	10	0	9	9	9	71	62	71	280	268	280	2.15
rnd_m25	25	0	20	18	19	154	122	148	612	448	612	5.8
rnd_m5	5	0	5	5	5	40	38	40	156	148	156	1.9
rnd_m50	50	0	34	27	28	250	169	214	996	680	940	6.58
rnd1	122	0	50	39	39	324	212	302	1320	812	1232	11.5
rnd2	124	0	47	34	39	292	179	269	1164	684	1184	9.42
rnd3	134	0	49	39	32	306	213	240	1240	812	972	7.39
substr1	142	0	6	14	6	18	69	18	68	272	68	1.24
substr2	79	0	5	6	5	20	28	20	76	108	76	0.95
subtraction1	104	0	34	20	22	200	98	145	796	404	616	4.87
subtraction2	128	0	2	2	2	4	2	4	12	4	12	0.48

Figure 44. Machine Learning examples

total DFC of a circuit equivalent to this gate, that is constructed from two-input gates. The DFC of a four input AND gate, OR gate, or EXOR gate is then $2^2 + 2^2 + 2^2 = 12$, since such gates can be decomposed to balanced trees of three two-input

gates. The decomposition is shown in Figure 46. The block could be any of AND, OR or EXOR gates. Figure 46a shows the function on four variables with cardinality $2^4 = 16$. In Figure 46b, we show the same function after it is decomposed with DFC $= 2^2 + 2^2 + 2^2 = 12$.

The DFC of a *CDEC* gate is calculated in a similar way. As each implicant in a SOC is composed of an AND function and a NAND function, the DFC is calculated separately for AND and NAND functions and a 2 input AND gate which has the AND and NAND function as inputs. Let us illustrate the DFC of SOC with an example.

Example IX. 1 The function $F = abc(\bar{d} + \bar{e} + \bar{f})$, shown in Figure 42a, of 6 variables with cardinality $2^6 = 64$, has DFC of AND with 4 inputs and NAND with 3 inputs. The AND with 4 inputs is split to a two-level balanced tree with 2 levels and 3 2-input gates, so its DFC cost is $4 + 4 + 4 = 12$. The NAND (OR) gate has 3 inputs so is split to two 2-input gates. So its DFC is $4 + 4 = 8$. So total DFC is $12 + 8 = 20$. The decomposed function is shown in Figure 42b.

Please note that the repeated variables are not calculated. For instance in the function, $ab(\bar{a} + \bar{c})$, the $\bar{a}$ of the NAND function is not considered for calculating DFC and hence should be removed.

IX.4.2 Results for Machine Learning examples

The *CDEC* implicants for Machine Learning examples is generated using SOCMIN, and compared with results generated from Espresso and Exorcism. Comparison is done for the number of terms, number of literals and DFC for output generated from Espresso. Exorcism and SOCMIN as shown in Figures 43 and 44. The number of terms and literals for the *CDEC* implicants generated using SOCMIN is always

TABLE III

Machine Learning examples with data randomly removed

	25% don't cares				50% don't cares				90% don't cares			
	Espresso		Socmin		Espresso		Socmin		Espresso		Socmin	
example	#t	dfc	#t	dfc	#t	dfc	#t	dfc	#t	dfc	#t	dfc
add0	27	620	6	196	26	668	6	124	13	408	3	44
add2	26	576	9	268	21	528	6	172	10	308	2	48
add4	17	332	2	12	17	400	2	12	10	304	2	40
ch15f0	24	592	10	340	22	604	10	344	8	252	3	112
ch176f0	11	232	2	20	10	248	2	20	4	116	1	0
ch177f0	22	456	2	12	10	208	2	12	6	184	1	0
ch22f0	12	292	4	156	12	332	4	164	5	156	2	28
ch30f0	15	352	5	136	15	388	5	164	5	156	2	52
ch47f0	16	408	7	192	14	380	8	216	4	124	3	104
ch52f4	17	440	15	408	14	392	12	360	3	92	1	40
ch70f3	8	208	3	84	6	164	2	64	1	28	1	28
ch74f1	12	296	6	248	10	264	5	188	4	124	1	24
ch83f2	15	420	13	384	12	344	7	284	2	60	1	16
ch8f0	38	816	32	672	36	860	6	184	18	548	3	68
c_4_ones	50	1596	27	1320	33	1052	14	628	4	124	2	36
grt_than	28	628	6	188	31	776	10	264	12	372	2	28
intrvl1	15	368	12	324	13	332	8	236	6	180	1	0
intrvl2	27	612	13	468	27	692	9	248	10	300	3	48
kdd1	24	492	2	12	23	544	2	16	11	324	2	20
kdd10	28	640	6	160	25	652	4	112	7	204	2	44
kdd2	4	84	3	72	4	96	1	28	1	24	1	24
kdd3	14	312	2	28	14	336	2	28	6	176	1	0
kdd4	20	408	14	376	22	532	1	0	10	304	1	0
kdd5	19	400	4	76	23	588	4	48	7	216	3	88

lower or equal to the number of terms and literals for the prime implicants generated by Espresso. Also, the DFC is lower in most cases and hence SOCMIN can find important applications in Machine Learning.

TABLE IV

Machine Learning examples with data randomly removed

example	25% don't cares				50% don't cares				90% don't cares			
	Espresso		Socmin		Espresso		Socmin		Espresso		Socmin	
	#t	dfc	#t	dfc	#t	dfc	#t	dfc	#t	dfc	#t	dfc
kdd6	33	636	23	620	37	840	1	0	17	520	1	20
kdd7	31	664	22	644	29	676	5	64	13	400	3	60
kdd8	16	388	12	372	18	484	2	20	8	252	1	0
kdd9	18	448	16	320	15	404	4	144	6	184	2	24
maj_gate	44	1056	12	456	29	776	10	332	10	308	2	52
mod2	12	296	5	156	12	320	4	128	5	156	1	4
mux8	24	516	4	56	21	512	4	68	7	212	1	0
pal	14	444	14	444	12	380	6	268	2	60	1	12
pal_dbl_op	41	992	40	952	35	936	11	352	12	364	3	36
pal_op	42	1108	39	992	34	960	15	568	10	304	2	48
parity	94	3004	20	596	63	2012	14	544	8	252	2	44
remder2	28	720	13	500	23	608	10	404	5	152	2	36
rnd1	46	1228	44	1056	36	1004	18	824	12	372	4	60
rnd2	35	988	15	696	32	888	25	776	11	340	3	52
rnd3	48	1276	24	1076	37	1024	15	552	11	340	3	52
rnd_m1	0	0	1	0	1	0	1	0	1	0	1	0
rnd_m10	6	188	6	188	2	60	2	60	1	0	1	0
rnd_m25	15	468	12	432	8	248	6	196	1	28	1	28
rnd_m5	2	60	2	60	1	28	1	28	1	0	1	0
rnd_m50	27	804	24	744	18	540	9	436	3	92	1	40
sbstr1	28	620	7	104	30	784	9	200	11	340	4	116
sbstr2	20	472	5	108	15	384	5	136	7	216	2	48
sbtrct1	43	1160	15	536	31	868	9	340	10	312	0	0
sbtrct3	18	356	2	12	16	368	2	12	9	268	2	16

IX.4.3 Machine Learning examples with randomly generated data

For the Machine Learning examples, data is randomly removed and tested and compared with Espresso. The removed data is basically replaced with don't cares.

Figure 45. Scatter plot of input terms versus time.

Tables III and IV shows the results for SOCMIN and Espresso with 25, 50 and 90 percent data replaced with don't cares for the Machine Learning examples. As the percent of don't cares increase, SOCMIN gives very high quality solution and there is a huge decrease in the number of terms and DFC for the *CDEC* implicants generated using SOCMIN compared to the number of terms and DFC for the prime implicants generated by Espresso. This proves that SOCMIN works better than for all cases when Strongly Unspecified functions are considered.

In general, the examples demonstrate that our cube-splitting heuristic works very well on large (incompletely specified) functions. Another advantage of SOCMIN (of not much importance for *CY7C361*) is that it works even better for *very strongly* unspecified functions. Finally, in multi-coloring mode the nodes can be colored with

Probable-Color-Set, the set which comprises of all possible colors that the candidate node can be colored with, is large, then the compatibility check has to be done for all colors. The candidate color is chosen based on the minimal Color-Restriction-Matrix, in addition to compatibility check. Hence the computation time increases when the solution space or the number of final colors assigned or the number of cdec terms decreases. In general, when the number of input nodes is large and the number of solution is small, large amounts of time is needed.

If the graph GCCC consists of more normal or Incompatible edges, which means there is low compatibility between input cubes, computation time needed is less, since the solution space is large. In other words, the computation time increases with large number of nodes with high compatibility. In general, there is a strong corelation between the number of input cubes, the compatibility of these input cubes and the solution space.

CHAPTER X

CONCLUSION

A problem that is related to the fitting problem of the *Cypress CY7C361* chip is the *SOC Minimization*. Since the *CY7C361* has only 32 macrocells, a high quality logic minimization to reduce the number of macrocells is very important. The goal of this thesis was, however, more general, since we believe that *CDEC* can be used as a general-purpose gate for standard cell structures with few levels, and also for new PLD structures. We departed therefore in the thesis from the *Cypress* chip as a sole motivation of our work. We formulated a *generic logic synthesis problem of SOC minimization* and we developed and implemented a complete solution to this problem. An innovative aspect of our approach is that we propose graph coloring instead of set covering which is used by most of the authors for problems of this type. Moreover, we presented a modification to the well-known graph coloring problem from the literature, which is, proper graph coloring. In contrast, our approach, referred to as **compatible graph coloring** requires that for a group of nodes colored with the same color a global relation of compatibility is checked. Therefore, our graph has two kinds of edges: the edges (pairs of nodes) can be either incompatible or conditionally compatible. Such coloring problem is not known from the literature in graph coloring.

We developed also a Cube Splitting algorithm, whereby the input cubes are split in such a way, that the generated cubes are lower in number than the minterms, and when these cubes are used as nodes in graph coloring algorithm, gives near exact

solutions. The algorithms used in the SOC minimization program, SOCMIN, have been designed for Strongly Unspecified functions, defined by ON and OFF sets, and hence finds important applications for machine Learning and Pattern theory as well, where there is a high percent of don't cares. The approach to solving the covering problem dealt in this thesis, the Conditional Graph Coloring, can be used in other similar problems such as PLA minimization, or Column Minimization Problem in Curtis-like decomposition of Multi-valued Relations. We found also the Muller method very efficient for ON,OFF data representation: it can be used to extend any other single-output minimizer for incomplete functions to a multi-output one.

Concluding, the original achievements of this thesis are the following:

1. formulation of the SUM-of-CDECs Minimization Problem.
2. formulation of the compatible graph coloring problem for the first problem.
3. development of the efficient algorithm to solve the compatible graph coloring problem.
4. development of the efficient algorithm for Cube Splitting.
5. application of the rarely used Muller method and its implementation in ON OFF model of our minimizer, where, contrary to ON/DC model from there literature, it is efficient. proving by our numerical results that our approach is superior on circuit benchmarks to the approach used in the well-known SOP minimizer Espresso and the commercial tool from Cypress Corporation.
6. proving by our numerical results that our approach is superior to Espresso on Machine Learning examples, adding thus to the repertoire of the existing approaches to Machine Learning.

REFERENCES

References

- [1] ALGOTRONIX Ltd., "Configurable Array Logic User Manual," Edinburgh, UK 1991.
- [2] ACTEL Inc., Databook, 1990.
- [3] XILINX, "The Programmable Gate Array Design Handbook", 1986.
- [4] Brayton R.K., Hachtel G.D., McMullen C.T., .L. Sangiovanni-Vincentelli, "Logic Minimization Algorithms for VLSI Synthesis," *Kluwer Academic Publishers*, 1984
- [5] Cypress SEMICONDUCTOR, in *BIMOS CMOS Data Book*, 1991.
- [6] Anderson, R., "Programmable High-Speed State Machine with Sequencing Capabilities," *U.S. Patent*, no. 4,965,472, October 23, 1990.
- [7] Cypress SEMICONDUCTOR, QUICKLOGIC, "News Release," June 2, 1992.
- [8] Cypress SEMICONDUCTOR, Warp1 PLD Compiler, 1992.
- [9] Nguyen, L.B., Perkowski, M.A., and Goldstein, N.B., "PALMINI - Fast Boolean Minimizer for Personal Computers," *Proc. of the 24th DAC*, Miami, FL, pp.615-621, 1987.
- [10] Ciesielski, M., Yang, S., and Perkowski, M.A., "Multiple-Valued Minimization Based on Graph Coloring", *Proc. of the IEEE ICCD'89*, pp. 262-265.
- [11] Wan, W., Perkowski, M.A., "A New Approach to the Decomposition of Incompletely Specified Multi-Output Functions Based on Graph-Coloring and Local Transformations and its Application to FPGA mapping," *Proc. of EURO-DAC'92*, Hamburg, September, 1992.
- [12] Johnson, D.S., "Worst-Case Behavior of Graph Coloring Algorithms," *Proc. of the 5th Southeastern Conference on Combinatorics, Graph Theory and Computing*, pp. 513-528, Winnipeg, Canada, Utilitas Mathematica Publishing, 1974.
- [13] McDiarpid, C.J.H., "Determining the Chromatic Number of a Graph," *SIAM J. Computing*, Vol. 8, pp. 1-19. 1979.
- [14] Miller, R.E., "*Switching Theory*", Vol. 1 and 2, John Wiley, New York, 1965.
- [15] Anderson, R., Coppola, A., Freedman, J., Perkowski, M.A., Pierzchala, E., "Optimal Synthesis of Tokenized State Machines into a Programmable Logic Device", Portland State University EE Technical Report, 1991.

- [16] Anderson, R. E., Coppola, A., Perkowski, M. A., "VHDL Synthesis of Concurrent State Machines to a Programmable Logic Device", *VHDL International User's Forum*, Scottsdale, Arizona, May 3-6, 1992.
- [17] Perkowski, M.A. and Chrzanowska-Jeske, M.E., Coppola, A., and Pierzchala, E., "An Exact Algorithm for the Technology Fitting Problem in the Application Specific State Machine Device," *ISCAS*, 1991.
- [18] Perkowski, M.A., Wu, P, and K.A. Pirkel, "KUAL-exact: A new approach for multi-valued logic minimization in VLSI synthesis", *Proc. IEEE Int. Symp. on Circuits and Systems*, pp. 401-404.
- [19] A. Coppola, M.A. Perkowski, R. Anderson, J. Freedman, and E. Pierzchala, "Synthesis of Very Fast Distributed Controllers Based on Tokenized State Machine Model," *Proc. IFIP Work. on Control-Dominated Synthesis from a Register Transfer Level Description*, Sept. 3-4, Grenoble, France, 1992.
- [20] R.K. Brayton, P.C. Mc Geer, J.V. Sanghavi, and A.L. Sangiovanni-Vincentelli, "A New Exact Minimizer for Two-Level Logic Synthesis," in T. Sasao (ed), "Logic Synthesis and Optimization", *Kluwer Academic Publishers*, 1993, pp.1-31.
- [21] B. J. Falkowski, and M.A. Perkowski, "Algorithm for the Generation of Disjoint Cubes for Completely and Incompletely Specified Boolean Functions," *Intern. J. of Electr.*, Vol. 70, No. 3, pp. 533 - 538, March 1991.
- [22] M.A. Perkowski, and A. Coppola, "A State Machine PLD and Associated Minimization Algorithms", *Proc. of FPGA '92*, Berkeley, Febr. 16-18, 1992.
- [23] M.A. 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*, pp. 102-109, 1995.
- [24] R. Rudell, and A. Sangiovanni-Vincentelli, "Exact Minimization of Multiple-Valued Functions for PLA Optimization," *Proc. ICCAD'86*, pp. 352-355, Nov. 1986.
- [25] N. Song, and M.A. Perkowski, "Minimization of Exclusive Sum of Products Expressions for Multi-Output Multiple-Valued Input Switching Functions," *submitted to IEEE Trans. on CAD*.
- [26] R.E. Bryant, "Graph-Based Algorithms for Boolean Function Manipulation," *Trans. on Comput.*, Vol. C-35, No. 8, pp. 667-691, 1986.
- [27] Y.S. Abu-Mostafa, "*Complexity in Information Theory*", Springer-Verlag, New York, 150pp, ISBN 0-387-96600-5, 1988.
- [28] S. Goller, M.A. Perkowski, "Logic Minimization using "Condition Decoder" gates for the Speed Optimized State Machine PLD", report, PSU

- [29] S. Mohamedsadakathulla, M.A. Perkowski, S. Goller, A. Coppola, "Fast approximate minimization of multi-output boolean functions in sum-of-decoder structures," *submitted to DAC'96*.
- [30] H.A. Curtis, "A New Approach to the Design of Switching Circuits", *Princeton*, N.J., Van Nostrand, 1962.