ABSTRACT

An abstract of the thesis of Ren Guangsheng for the Master of Science in Electrical and Computer Engineering presented April 15, 1998.

Title: Realization of the Architecture for Ashenhurst-Curtis Decomposition in DECpeRLe-1 Board

Presented in this thesis are algorithms of hardware realization of Ashenhurst/Curtis decomposition for boolean functions.

Function Decomposition is one of the most general methods to solve engineering and science optimization problems. The objective of any function decomposition approach has always been to reduce the complexity of the problem. Function Decomposition is an NP-complete problem. Over the last twenty years, a lot of the software programs have been designed to realize the function decomposition. These programs however are usually time-consuming. It is therefore necessary to design a hardware accelerator to speed up these algorithms.

In this thesis, I use digital system design methodology to develop algorithms that can significantly improve the performance and quality of five steps of Ashenhurst/Curtis-

type decomposition for boolean functions. These five steps are variable partitioning, compatible columns, maximum cliques and predecessor and successor functions creation.

This hardware accelerator has been implemented with DECpeRLe-1 Board with VHDL codes. The simulation results demonstrated that this hardware accelerator greatly speeded up the entire process of the function decomposition when compared to the software programs that are used for the same purpose.

ACKNOWLEDGEMENTS

I would like to thank Dr. Marek A. Perkowski, my advisor, who introduced me to the areas of Logic Synthesis, ASIC System Design and Decomposition of Functions and Relations. This thesis would not have been possible without the guidance and encouragement I received from Dr. Perkowski. I would also like to thank Dr. Fu Li and Dr. Y. C. Jenq for their valuable comments which helped to improve this thesis.

I would also like to thank every member of my group in EE610 VHDL for their many question, answers, and feedback related to my thesis research. They are Shruti Vij, Chen Qihong, Ryan Mascarenhas, Easwar Sarangapani and Liu Rong. Also, I would also like to thank Laura for all her assistance and support.

I would also like to thank my sister and her husband for their support and encouragement.

REALIZATION OF THE ARCHITECTURE FOR ASHENHURST-CURTIS DECOMPOSITION IN DECPERLE-1 BOARD

by

REN GUANGSHENG

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

MASTER OF SCIENCE
in
ELECTRICAL AND COMPUTER ENGINEERING

Portland State University
1998

THESIS APPROVAL

The abstract and thesis of Ren Guangsheng for the Master of Science in Electrical and Computer Engineering were presented April 15, 1998, and accepted by the thesis committee and the department.

COMMITTEE APPROVALS: ______________________________

Marek A. Perkowski, Chair

____

Fu Li

____

Y.C. Jenq
Representative of the Office
of Graduate Studies

DEPARTMENT APPROVAL: ______________________________

Rolf Schaumann, Chair
Department of Electrical and Computer
Engineering

* *

ACCEPTED FOR PORTLAND STATE UNIVERSITY BY THE LIBRARY

by ______________________________ on ____________________

CONTENTS

LIST OF TABLES iii

LIST OF FIGURES iv

1 INTRODUCTION 1

2 Concept of Functional Decomposition 5
- 2.1 Introduction . . . 5
- 2.2 Concepts of Function Decomposition . . . 5
- 2.3 Fundamental Definitions . . . 6
  - 2.3.1 Cube Calculus . . . 6
  - 2.3.2 Compatible Columns . . . 7
  - 2.3.3 A Karnaugh Map vs a Decomposition Chart . . . 10
  - 2.3.4 Maximum Cliques . . . 10
  - 2.3.5 Predecessor Function(G Function) . . . 11
  - 2.3.6 Successor Function(H Function) . . . 12
- 2.4 Summary: One Example for the Process of Boolean Function Decomposition . . . 14

3 Hardware System Architecture of Function Decomposition 20
- 3.1 My Hardware Accelerator Design Methodology with Prototyping Board . . . 20

4 PAM and DECpeRLe-1 26
- 4.1 Programmable Active Memory(PAM) . . . 26
- 4.2 DECpeRLe-1 Board . . . 27
  - 4.2.1 Computational Matrix . . . 29
  - 4.2.2 Switches and I/O buses . . . 31
  - 4.2.3 Control Resource . . . 33
  - 4.2.4 Memory Subsystem . . . 36
  - 4.2.5 Host Interface . . . 37

5 Optimal Design Variable Partitioning 39
- 5.1 Circuit Algorithm Description . . . 40
- 5.2 Circuit Design Description . . . 45

5.3 Mapping Variable Partitioning Circuit into DECpeRLe-1 Board . 48

6 Optimal Design Compatible Columns **52**

6.1 Circuit Algorithm Description 53
6.2 Circuit Design Description 56
6.3 Mapping Compatible Columns Circuit into DECpeRLe-1 Board . 59
6.4 Summary 59

7 Optimal Design Maximum Clique Circuit **61**

7.1 Circuit Algorithm Description 62
7.2 Circuit Architecture 66
7.3 Mapping Maximum Cliques Circuit onto DECpeRle-1 Board . . . 69
7.4 Summary 70

8 Optimal Design G Function Block **72**

8.1 Circuit Algorithm Description 73
8.2 Circuit Design Description 75
8.3 Mapping G1 Function Circuit into DECpeRL-1 Board 78
8.4 Summary 78

9 Optimal Design H Function Block **81**

9.1 Circuit Algorithm Description 81
9.2 Circuit Design Description 84
9.3 Mapping H Function Circuit into DECpeRL-1 Board 87
9.4 Summary 87

10 Simulation **92**

10.1 Design Capture 92
10.2 Functional Verification 93
10.2.1 Simple Test Result for Function Decomposition 93

11 MY CONTRIBUTIONS TO THE FUNCTIONAL DECOMPOSITION GROUP, CONCLUSIONS AND FUTURE WORK **98**

BIBLIOGRAPHY **100**

LIST OF TABLES

LIST OF FIGURES

2.1 the Structure of Function Decomposition 6
2.2 Five Variables Karnaugh Map 8
2.3 the Graph of Compatible Column and Incompatible Column . . . 9
2.4 Five Variables Decomposition Chart 9
2.5 Columns Encoding 11
2.6 G1,G2 Function Karnaugh Map 12
2.7 H Function Karnaugh Map 12
2.8 Cube Representation for Example 13
2.9 Column Compatiblity 14
2.10 Listings of On and Off Cofactors 18
2.11 Value of Decomposition Block 18

3.1 Stages in Design 21
3.2 Outline of Hardware and Software Implementation 25

4.1 PAMs as Virtual Machines 27
4.2 DECpeRLe-1 Architecture 28
4.3 DECpeRLe-1 Matrix 30
4.4 DECpeRLe-1 Direct Connection 31
4.5 DECpeRLe-1 Matrix Data Buses 32
4.6 DECpeRLe-1 Matrix Rings 32
4.7 DECpeRLe-1 Control Wires 34

5.1 Diagram Approach to Finding Bound Sets 42
5.2 Search Tree for Finding the Bound Set 43
5.3 Calculating the Bound Set Variable Array 45
5.4 the Architecture of Variable Partitioning 47
5.5 the Flowchart of Variable Partitioning 49
5.6 the State Diagram of Variable Partitioning 50
5.7 the Outline of Mapping 51

6.1 Five Variable Karnaugh Map 52
6.2 Compatible Columns Circuit Process 54
6.3 the Flowchart of Compatible Column Circuit 55
6.4 the Architecture of the Compatible Column Circuit 57
6.5 the State Diagram of the Compatible Column 58
6.6 State Table for Control Unit 59

6.7 the Outline of Mapping 60

7.1 Compatible Column 62
7.2 Compatible Column in Maximum Cliques 64
7.3 the Process of Maximum Cliques 65
7.4 the Flowchart of Maximum Cliques Circuit 67
7.5 the Architecture of Maximum Cliques Circuit 68
7.6 the State Diagram of Maximum Cliques Circuit 69
7.7 the State Table for Control Unit 70
7.8 the Outline of Mapping 71

8.1 Bound Set, Columns and Encoding Bits 73
8.2 G1 Circuit Process 74
8.3 Flowchart of G1 Circuit Algorithm 76
8.4 the Architecutre of G1 Circuit 79
8.5 the State Diagram of G1 Function Circuit 80
8.6 the Outline of Mapping 80

9.1 Free Set and G1, G2 Combination Structure 82
9.2 H Function Data Member 83
9.3 Flowchart of H Circuit Algorithm 85
9.4 the Architecutre of H Function Circuit 88
9.5 the State Diagram of H Function Circuit 89
9.6 the Outline of Mapping 90

10.1 Hierarchical Structure of the Hardware Accelerator for Function Decomposition 93

CHAPTER 1

INTRODUCTION

Function Decomposition is one of the most general methods to solve engineering and science optimization problems. Function Decomposition involves breaking down a larger complex structure into several smaller and less complex structures and each smaller structure can be broken down by themselves.

Recently, Function Decomposition has been used in some areas. For example, in TRADE [1], Functional Decomposition can provide very good results for mapping a function into a Xilinx Field Programmable Gate Array(FPGA). Function Decomposition can be used for Look Up Table(LUT) based FPGAs to minimize the number of LUTs used. Decomposition can also be used in areas like VLSI design, PLA Decomposition, Image Processing, Machine Learning, etc. In machine learning, Decomposition can be used to extract the pattern in a function. This pattern is basically the reduced decomposed function.

All of these applications are still realized with software. Although there are many new released software which are tried to solve the problems of one of function blocks of function decomposition, they still couldn't breakthrough the bottleneck of function decomposition, which as the number of input variables and terms increases, the decomposition time increases usually exponentially or super-exponentially, depending on the effectiveness of the algorithms are applied. The time consuming is still much.

Function Decomposition is an NP complete problem. (*Function Decomposition's function blocks, except G and H function block creation , are NP complete problem.*) Hence, even if more efficient algorithms can be found for exact calculations, they will all eventually become impractical. One approach to finding solutions to NP-hard problem is not to resist on an exact solution, but to be satisfied with one which is almost optimal and obtainable in a reasonable time. This type of algorithm is base on heuristics, or rules which can be applied which are likely to improve the solution. [2]

This time, I choose the algorithms which are simple, easy, fast and can be run in hardware. I use digital system design theory to make a special hardware accelerator to realize one kind of function decomposition - disjoint binary function decomposition. This hardware accelerator integrates all of steps of function decomposition. Then it is implemented into DECpeRle-1 Board. After it is created, I design the function test and performance test to demonstrate whether this hardware acclerator can greatly speedup the whole process for function decomposition and get a higher quality results compared to software methods.

This hardware accelerator's limitation is high cost. But if the test result is better than software, then function decomposition can be used more widely in industry to fit for the increasing needs. This hardware accelerator can be a processor unit in real circuit applied for function decomposition field.

This thesis is organized as follows:

Chapter 2 of this thesis gives an introduction to Function Decomposition concepts. Chapter 3 presents the digital system design methodology and the architecture of my hardware accelerator. In Chapter 4, my hardware accelerator's foundry DECpeRLe-1 Board is explained.

Chapter 5, 6, 7, 8, 9 present the optimal design - five function blocks of the function decomposition and how to map the desing in DECpeRLe-1 Board. Chapter 5 describes the variable partitioning algorithm. Chapter 6 describes the column compatible algorithm. Chapter 7 describes the maximum cliques algorithm. Chapter 8 describes the processor function realization algorithm. Chapter 9 describes the successor function realization algorithm.

Chapter 10 shows the function test and performance test which can explain the hardware accelerator's efficiency.

Chpater 11 is the summary of the entire system, communication, loading, results, etc.

Finally, Chapter 12 summarizes the work and addresses my point of view of what the future tasks of the Functional Decomposition group at Portland State University should be.

CHAPTER 2

Concept of Functional Decomposition

2.1 Introduction

Decomposition is a method to break down a large function into a number of smaller functions, which are easier to analyze and realize than the large input function. Boolean Decomposition uses a Boolean representation. Many different kinds of Decompositions have been developed. The first was the Ashenhurst Decomposition defined in [3], then Curtis [4] extended the Ashenhurst Decomposition, making it more general. This thesis deals with the Curtis decomposition, and in this chapter, the Curtis Decomposition will be presented.

2.2 Concepts of Function Decomposition

Speaking of decomposition, a complex logic function **F** can be divided into two logic blocks: Block **G**, the predecessor block; and Block **H**, the successor block. In order to accomplish this, the input variables to the function block **F** must be divided into a free set of the variables to block **H** and a bound set of the variables to Block **G**.

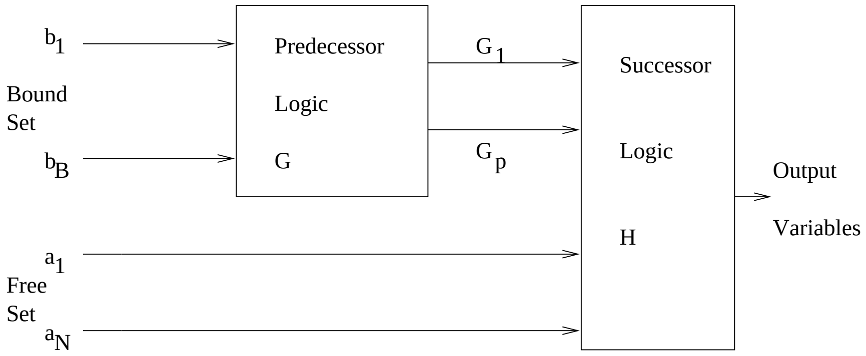

Figure 2.1: the Structure of Function Decomposition

In Fig 2.1, the free set of variables are $a_1......a_N$, which are unmodified variables and send directly to the new logic block H. The bound set of variables are $b_1......b_B$, which will be passed through the logic block G. In order to reduce the size and complexity of the implementation, the logic block G has produced intermediate variables $G_1......G_p$.

2.3 Fundamental Definitions

2.3.1 Cube Calculus

Cube calculus is a representation method of switching functions in which the main concepts used are of CUBES and an ARRAY OF CUBES. A cube is a compact expression of a set of minterms. For example, the minterms *11010* and *11000* can be expressed as a cube *110X0* where X means it takes the value of both *0* or *1*.

If the output of the cube is *1*, it is called the ON-CUBE.

If the output of the cube is *0* , it is called the OFF-CUBE.

If the output of the cube is X, it is called the DONT-CARE (DC).

The ON-SET is a collection of ON cubes, while the OFF-SET is a collection of a set of OFF cubes. Cube calculus operations are the operations applied to cubes and array of cubes. The rules of the intersection operation are as follows: $0 \cap 0 = 0$, $0 \cap 1 = \in$, $0 \cap X = 0$, $1 \cap 0 = \in$, $1 \cap 1 = 1$, $1 \cap X = 1$, $X \cap 0 = 0$, $X \cap 1 = 1$, $X \cap X = X$. The cube calculus operations are very useful for the general representation of switching functions.

2.3.2 Compatible Columns

Two columns are compatible if their corresponding cells are compatible. For example in Fig 2.2, c_1 and c_8 are compatible as $c_1 = X10X$ and $c_8 = 0XX1$. These two columns are also called *column cofactors or K-map loops.* The formula to test compatibility is as follows. Two columns i and j are compatible if :

$$[ON(i) \cap OFF(j) = 0] \wedge [ON(j) \cap OFF(i) = 0]$$

AB \ CDE	000	001	011	010	110	111	101	100
00	-	-	1	0	-	-	-	0
01	1	0	0	1	0	0	0	-
11	0	1	-	1	0	1	1	-
10	-	-	1	0	1	0	-	1
	C1	C2	C3	C4	C5	C6	C7	C8

Figure 2.2: Five Variables Karnaugh Map

Or,

$$i\ compatible\ j\ \ iff\ \ (ON(i)\#ON(j) = \in)\ and\ (ON(j)\#ON(i) = \in)$$

The incompatibility graph is a graph which illustrates the incompatibility relation among the columns. Each node in the incompatibility graph corresponds to a column in the K-map. If two columns are incompatible, then in the incompatibility graph, there is an edge between the corresponding nodes and if they are compatible, there is no edge.

The compatibility graph is a graph that illustrates the compatibility relationship among the columns and as either two columns can be either compatible or incompatible; therefore the compatibility and incompatibility graphs are mutual complements.

Fig 2.3 demonstrates more clearly the relationship shown in Fig 2.2.

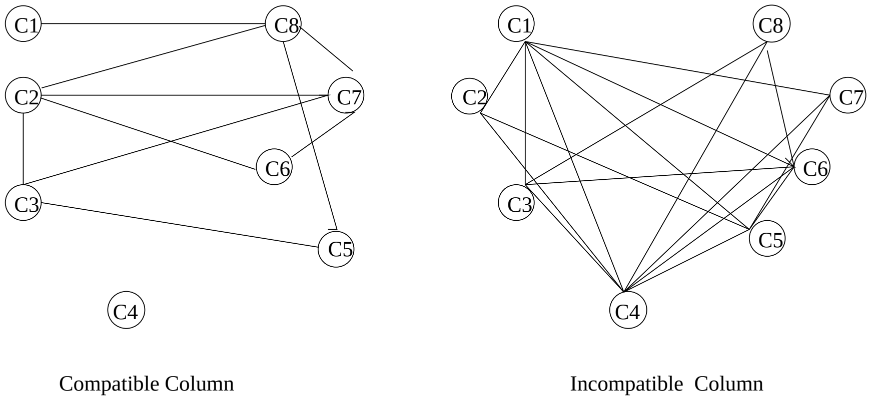

Figure 2.3: the Graph of Compatible Column and Incompatible Column

AB \ CDE	000	001	010	011	100	101	110	111
00	-	-	0	1	0	-	-	-
01	1	0	1	0	-	0	0	0
10	-	-	0	1	1	-	1	0
11	0	1	1	-	-	1	0	1

Figure 2.4: Five Variables Decomposition Chart

2.3.3 A Karnaugh Map vs a Decomposition Chart

Fig 2.4 shows the same function(as shown in Fig 2.2) represented in a Decomposition chart. The only difference between a Karnaugh map and a Decomposition chart is that the row and column indices in a Karnaugh map are in Gray code while they are in binary order in a Decomposition chart.

The *column multiplicity* denoted by $\mu(A/B)$, is the number of different columns in a decomposition chart. Here A is the free set and B is the bound set.

In Fig 2.4, its column multiplicity is "4". The best variable partitioning standard is to find the least column multiplicity of an individual decomposition chart.

2.3.4 Maximum Cliques

Once we have clearly partitioned the variables into bound set and free set in F function, the next step is to use maximum cliques technique to minimize and encode the columns in F function.

The nodes and their edges could form a complete geometry diagram such as an isolated node, a line, a triangle, a polygon, a pentagon, etc. In the meantime, each node can be connected with any other nodes, then these nodes in the same geometry diagram are defined as maximum cliques.

From Fig 2.3, the following combinations of the maximum cliques could be

AB \ CDE	000	001	011	010	110	111	101	100
00	-	-	1	0	-	-	-	0
01	1	0	0	1	0	0	0	-
11	0	1	-	1	0	1	1	-
10	-	-	1	0	1	0	-	1
	C1	C2	C3	C4	C5	C6	C7	C8
G1G2	00	01	11	10	11	01	01	00

Figure 2.5: Columns Encoding

found and encoded by the two bits G_1 and G_2. 1) C_1 and C_8 are encoded as 00. 2) C_2, C_6 and C_7 are encoded as 01. 3) C_3 and C_5 are encoded as 11. 4) C_4 is encoded as 10.

Fig 2.5 is the refinement of Fig 2.2 and Fig 2.3 to show how the columns have been encoded. The columns that have the same encoding come from the same maximum clique.

2.3.5 Predecessor Function(G Function)

Because the number of the column encoding bits are 2, the G function block should have 2 bits, G_1 and G_2, that are constituted by the bound set variables(C, D, E).

From Fig 2.5, the onset and the offset values in G1 and G2 could be easily derived as the karnaugh map as shown in Fig 2.6.

C \ DE	00	01	11	10
0	0	0	1	1
1	0	0	0	1

(G1 Karnaugh Map)

C \ DE	00	01	11	10
0	0	1	1	0
1	0	1	1	1

(G2 Karnaugh Map)

Figure 2.6: G1,G2 Function Karnaugh Map

AB \ G1G2	00	01	11	10
00	0	-	1	0
01	1	0	0	1
11	0	1	0	1
10	1	0	1	0

Figure 2.7: H Function Karnaugh Map

2.3.6 Successor Function(H Function)

After the predecessor functions have been set up, the successor function could be found by combining G1, G2 with free set variables(A, B). H function has only one bit and maybe has *don'tcare* set.

Fig 2.7 shows the successor function's Karnaugh map.

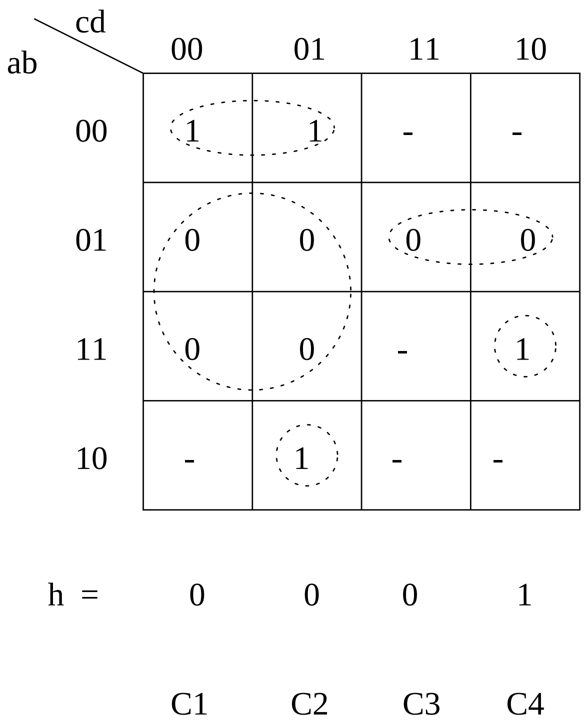

Figure 2.8: Cube Representation for Example

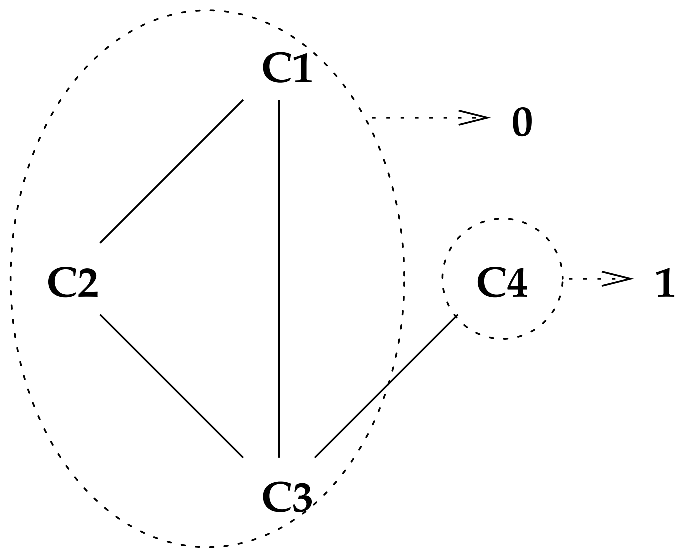

Figure 2.9: Column Compatiblity

2.4 Summary: One Example for the Process of Boolean Function Decomposition

The example shown in Fig 2.8 can be used to summarize the whole process of boolean function decomposition.

The bound set $\{c, d\}$ is represented as columns in the K-map and the free set is represented as rows. Having a close look at first three columns of the K-map, we see that the values are the same i.e $c_1 = 1\ 0\ 0\ X$, $c_2 = 1\ 0\ 0\ 1$, $c_3 = X\ 0\ X\ X$. So, the three columns can be represented by the same combined column c, while $c_4 = X\ 0\ 1\ X$ is a different combined column.

The maximum clique is therefore obtained as shown in Fig 2.9.

Thus, the columns c_1, c_2, c_3 are encoded as 0 and and the column c_4 as 1. In case only 2 columns are similar, a two bit representation has to be done giving result of not just g but g_1 and g_2. The way to go about it is to firstly calculate all cofactors of the columns represented by $\bar{c}\bar{d}, \bar{c}d, cd, c\bar{d}$.

For example, the cofactor of value i of variable X : $f \mid X^i = f(X = i, ...)$ and cofactor of product of literals $X_1^i X_2^j$; $f \mid X_1^i X_2^j = f(X_1 = i,\ X_2 = j, ...)$, and so on. All unions are represented as an array of cubes. The objective of doing this is to find out all the 1's (ON's) and all the 0's (OFF's) that are in a particular column, and to see if one column is compatible with the other or not.

For an incompletely specified binary function, two nodes for a graph for coloring are incompatible if and only if the corresponding columns are not compatible. Thus, c_1 is incompatible with c_2 iff $(ON(c_1) \cap OFF(c_2) \neq \phi)$ or $(ON(c_2) \cap OFF(c_1) \neq \phi)$. Therefore only two basic cube calculus operations, SHARP and COFACTOR are used for incomplete functions.

This is explained in the example in Fig 2.8. Looking at the K-MAP of Fig 2.8, $ON_f = \{000X, 1001, 1110\}$, $OFF_f = \{X10X, 011X\}$

The cofactors of columns are:

- For $\bar{c}\bar{d} = XX00$

    1. $XX00 \cap 000X = 00XX$. (Here since we are only interested in values of a and b, we put X in place of c and d to find out g for a given value of h).

2. $1001 \ \cap \ XX00 \ = \ \phi$

3. $1110 \ \cap \ XX00 \ = \ \phi$

So, $ON_{\bar{c}\bar{d}} \ = \ 00XX$ Similarly,

1. $X10X \ \cap \ XX00 \ = \ X1XX$

2. $011X \ \cap \ XX00 \ = \ \phi$

So, $OFF_{\bar{c}\bar{d}} \ = \ X1XX$

- Similarly, for $\bar{c}d \ = \ XX01$

1. $000X \ \cap \ XX01 \ = \ 00XX$

2. $1001 \ \cap \ XX01 \ = \ 10XX$

3. $1110 \ \cap \ XX01 \ = \phi$

So, $ON_{\bar{c}d} \ = \ 00XX, 10XX$

1. $XX01 \ \cap \ X10X \ = \ X1XX$

2. $XX01 \ \cap \ 011X \ = \ \phi$

So, $OFF_{c\bar{d}} \ = \ X1XX$

- Similarly, for $cd \ = \ XX11$

1. $XX11 \ \cap \ 000X \ = \ \phi$

2. $XX11 \cap 1001 = \phi$

3. $XX11 \cap 1110 = \phi$

So, $ON_{cd} = \phi$

1. $XX11 \cap X10X = \phi$

2. $XX11 \cap 011X = 01XX$

So, $OFF_{cd} = 01XX$

- Lastly, for $c\bar{d} = XX10$

1. $XX10 \cap 000X = \phi$

2. $XX10 \cap 1001 = \phi$

3. $XX10 \cap 1110 = 111X$

So, $ON_{c\bar{d}} = 11XX$

1. $XX10 \cap X10X = \phi$

2. $XX10 \cap 011X = 01XX$

So, $OFF_{c\bar{d}} = 01XX$

Therefore, the ON and OFF sets of the blocks G and H are as explained in Fig 2.11, $ON_{c_1} = \bar{a}\bar{b}$, $ON_{c_2} = \bar{b}$, $ON_{c_3} = \phi$, $ON_{c_4} = ab$, $OFF_{c_1} = b$, $OFF_{c_2} = b$, $OFF_{c_3} = \bar{a}b$, $OFF_{c_4} = \bar{a}b$.

Columns	ON	OFF
$\bar{c}\ \bar{d}$	00XX ($\bar{a}\,\bar{b}$)	X1XX (b)
$\bar{c}\ d$	00XX, 10XX ($\overline{ab}$, $a\bar{b}$)	X1XX (b)
$c\ d$	$\emptyset$	01XX ($\bar{a}b$)
$c\ \bar{d}$	11XX (ab)	01XX ($\bar{a}b$)

Figure 2.10: Listings of On and Off Cofactors

c \ d	0	1
0	0	0
1	1	0

G

ab \ G	0	1
00	1	-
01	0	0
11	0	1
10	1	-

H

Figure 2.11: Value of Decomposition Block

as shown in Fig 2.10

Similarly, $ON_G = c\bar{d}$,

$OFF_G = \bar{c}\bar{d} \cup \bar{c}d \cup cd$,

$ON_H = ON_{c_4} = ON_{c\bar{d}}$, and,

$OFF_H = (OFF_{c_1} \cup OFF_{c_2} \cup OFF_{c_3})$.

Also, we have the following results:

$ON_H = (ON_{c_1} \cup ON_{c_2} \cup ON_{c_3})\,\bar{G} + (\mathrm{ON}_{c_4})\,G$,

$OFF_H = (OFF_{c_1} \cup OFF_{c_2} \cup OFF_{c_3})\,\bar{G} + (\mathrm{OFF}_{c_4})\,G$.

In this chapter, Function Decomposition's process and definitions have been presented. The design methodology with prototyping board will be introduced in the next chapter.

CHAPTER 3

Hardware System Architecture of Function Decomposition

3.1 My Hardware Accelerator Design Methodology with Prototyping Board

My design of hardware accelerator for function decomposition is a digital system, which operates on digital signals with only a finite set of values. This hardware accelerator holds combinational and sequential behavior. It is specified by listing of its inputs and outputs in a table, or by writing down a formular, which computes the output in a table. Architecture defined is by converting algorithmic behavior into a block diagram. The algorithms for my hardware accelerator design are mainly expressed in VHDL codes, a hardware description languages which can be synthesized in the FPGA materials with tools. The architectures all have control and data sections. Signals are stored and transfered in the data part. Sometimes, it is not easy to seperate these two subsystems.

Learning from this methodology, I break my whole system design process up into 4 levels as shown in Fig 3.1.

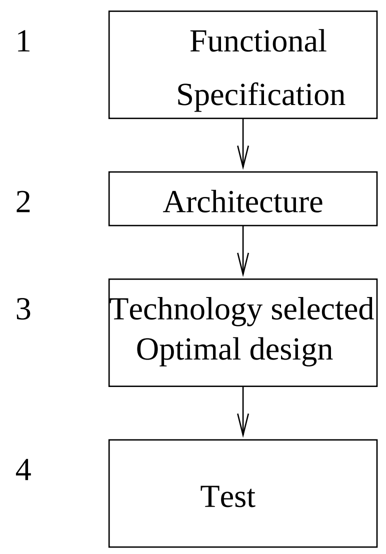

Figure 3.1: Stages in Design

Level 1 is my specification of the system's function. In my hardware accelerator, there are five major function blocks:

1) partitioning variables as bound set and free set.

2) finding the compatible column.

3) minimizing and encoding the columns by using the maximum cliques techniques

4), 5) realizing the predecessor function and successor function.

These five function block specifications will be shown in Chapter 5, 6, 7, 8, 9's algorithm parts.

Level 2 is my each function architecture. This shows why the algorithms are chosen and how to design each function block's control and data part separately, where this can be done conveniently. They will be shown in Chapter 5, 6, 7, 8, 9's algorithm parts, too.

Level 3 is my optimal design. The product technology selected strongly determines the quality of product. In this level, I choose computer aided prototyping board as my design foundry, which combines software versatility and hardware performance. This computer aided prototyping board can minimize the costs, move high-level design quickly to functioning hardware, verify easily every aspect of the design such as functional, timing, electrical and mechanical in the same processing, and have high controllability and observability.

DECpeRLe-1 board is one of the earlist of pioneers in this kind of computer aided prototyping boards, which was built at Digital's Paris Research Laboratory in 1992. System design, logic design and physics design can be implemented totally in the same time in the same process. The main elements can be adjusted together once to fit for these three design cycle requirements. That's the most advantage of DECpeRLe-1 board. Without this kind of reconfigurable computing board, these three design cycles always are implemented separately. The same elements in each cycle need to be changed three times. It really costs a lot of time.

DECpeRle-1 Board is a single board PAM(programmable active memory) system. It is a 4 by 4 matrix of Xilinx XC3090 FPGA's connected in a regular mesh and this sytem is connected to the host computer through a turbo channel interface that allows CPU access. The general structure of DECpeRLe-1 Board will be presented and shown in Chapter 4.

After the product foundry selected, each architecture's behavior is described with VHDL codes. Through Xilinx tools, synthesize these codes and then map into DECpeRLe-1 board. The mapping diagram will be drawn as shown in Chapter 5, 6, 7, 8, 9 in turn.

Level 4 is my test process. In this level, I would test each function block, thefunctional result will be shown in Chapter 10. Then I test the whole hardware accelerator and compare it with the most new version software code available in PSU. The performance result will be shown in Chapter 10, too.

Fig 3.2 approximately describes the signals flow and outlines of function blocks which are mapped in DECpeRLe-1 Board(as shown in Fig 4.2).

ON Sets and OFF Sets data members are calculated using software from the host workstation on which this hardware accelerator is installed, then they are stored into memory A, which is the first function block *variable partitioning*'s input data memory. After the data are processed in *variable partitioning* step, the results are put into memory B, which is the second function block *compatible columns* input memory. After the data are processed by this step, the results are put into memory C. Memory C is the input memory of the fourth function block *maximum cliques.* After the data are processed by this fourth step, the results are put into memory D, which is input memory of the fifth function block *creation of G and H function block* . After the data are processed by this step, the results are put into memory E.

Of course, in the real mapping, all of components shown in Fig 3.2 would be tried to be put inside the DECpeRLe-1 Board. Considered some limited parameters, such as easily handled, some of components would be put outside the DECpeRLe-1 Board.

Next chapter will present my hardware accelerator foundry's structure, DECpeRLe-1 Board.

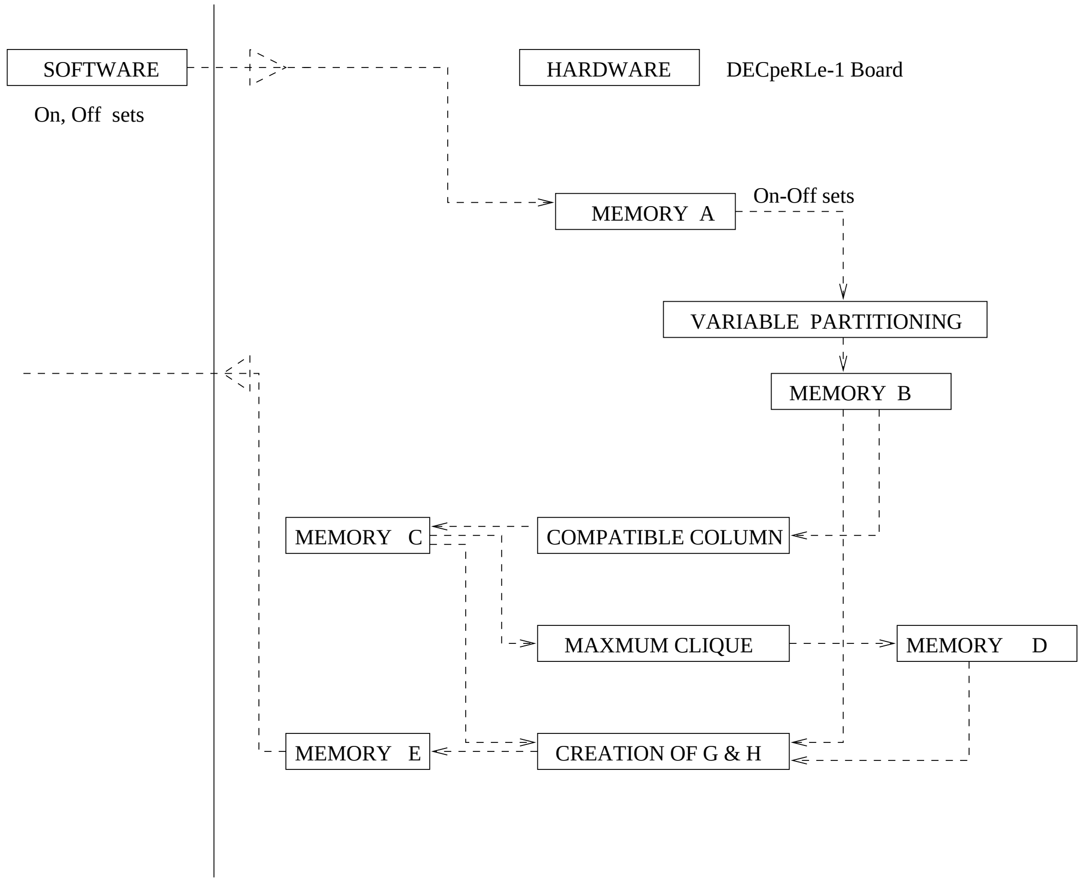

Figure 3.2: Outline of Hardware and Software Implementation

CHAPTER 4

PAM and DECpeRLe-1

This chapter mainly presents DECpeRLe-1 Board's structure, all programmable resources and the way they are interconnected with each other. Other detail materials such as DECpeRLe-1's performance, clocking, runtime library and programming etc, can be found in DECPAM [7]. DECpeRLe-1 Board is available in the Department of Electrical Engineering at Portland State University.

4.1 Programmable Active Memory(PAM)

In this board, PAM is the central part. PAM is a novel form of universal reconfigurable hardware copressor based in Field-Programmable Gate Array(FPGA) technology. A PAM is a virtual machine, (shown in Fig 4.1), controlled by a standard microprocessor which can be dynamically and indefinitely reconfigured into a large number of application-specific circuits. PAMs offer a new mixture of hardware performance and software versatility.

PAMRSCA [8] shows PAM's application can span a wide range of current

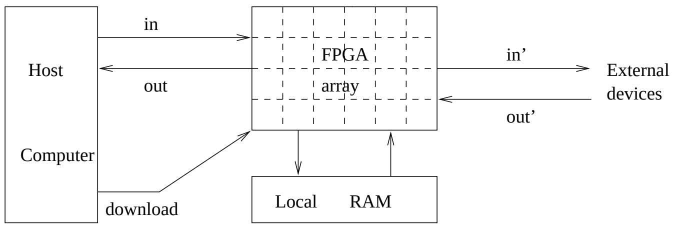

Figure 4.1: PAMs as Virtual Machines

leading-edge computational challenges, such as computer arithmetic, cryptography, error correction, image analysis, stereo vision, video compression, sound synthesis, neural network, high-energy physics, thermo dynamics, biology and astronomy. For each, PAM technology outperforms all other existing implementation media.

DECpeRLe-1 is the third generation PAM board.

4.2 DECpeRLe-1 Board

The overall structure of DECpeRLe-1 is shown in Fig.4.2. The DECpeRLe-1 board is organized around a central computational matrix made up of 16 Xilinx XC3090 LCAs(Logic Cell Array), surrounded by a 4 1MB RAM banks, and 7 other LCAs to implement switching and controlling functions. The data buses

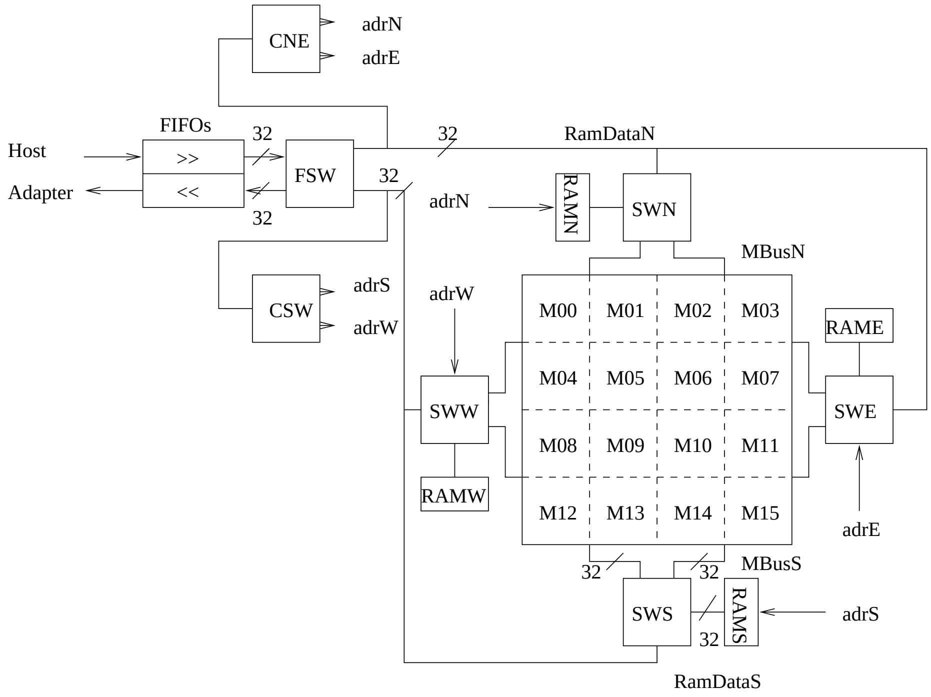

Figure 4.2: DECpeRLe-1 Architecture

are 32 bits. All control wires are shown in Fig.4.7.

4.2.1 Computational Matrix

The central computational matrix is a 4 x 4 matrix of Xilinx 3090 LCAs. These LCAs are interconnected with each other. The internal structure of the matrix is shown in Fig.4.3. The LCAs are named *LCA_M00* to *LCA_M15*. This matrix can be used to develop any kind of digital circuitry: data path, control unit and others. The interconnection resource between them can be classified into the following three categories.

Direct Connections

As shown in Fig.4.4, these wires connect the adjacent sides of adjacent LCAs. The main purpose of direct connections is to extend the internal regularity of the LCA to the matrix level. Each LCA has 16 such wires on each side.

Buses

As shown in Fig.4.5, these horizontal or vertical wires connect the corresponding side of all 4 LCAs in the same row or column. They can thus efficiently distribute global data in one direction, and are comparable to the longline interconnections resources in Xilinx internal architecture. Each LCA has 16 such wires on each side. According to their directions, these buses are named matrix *North*, *East*, *South* and *West* bus, respectively, and represented by *MBusN*, *MBusE*, *MBusS* and *MBusW* for short. Each bus has 64 wires which are connected to switches

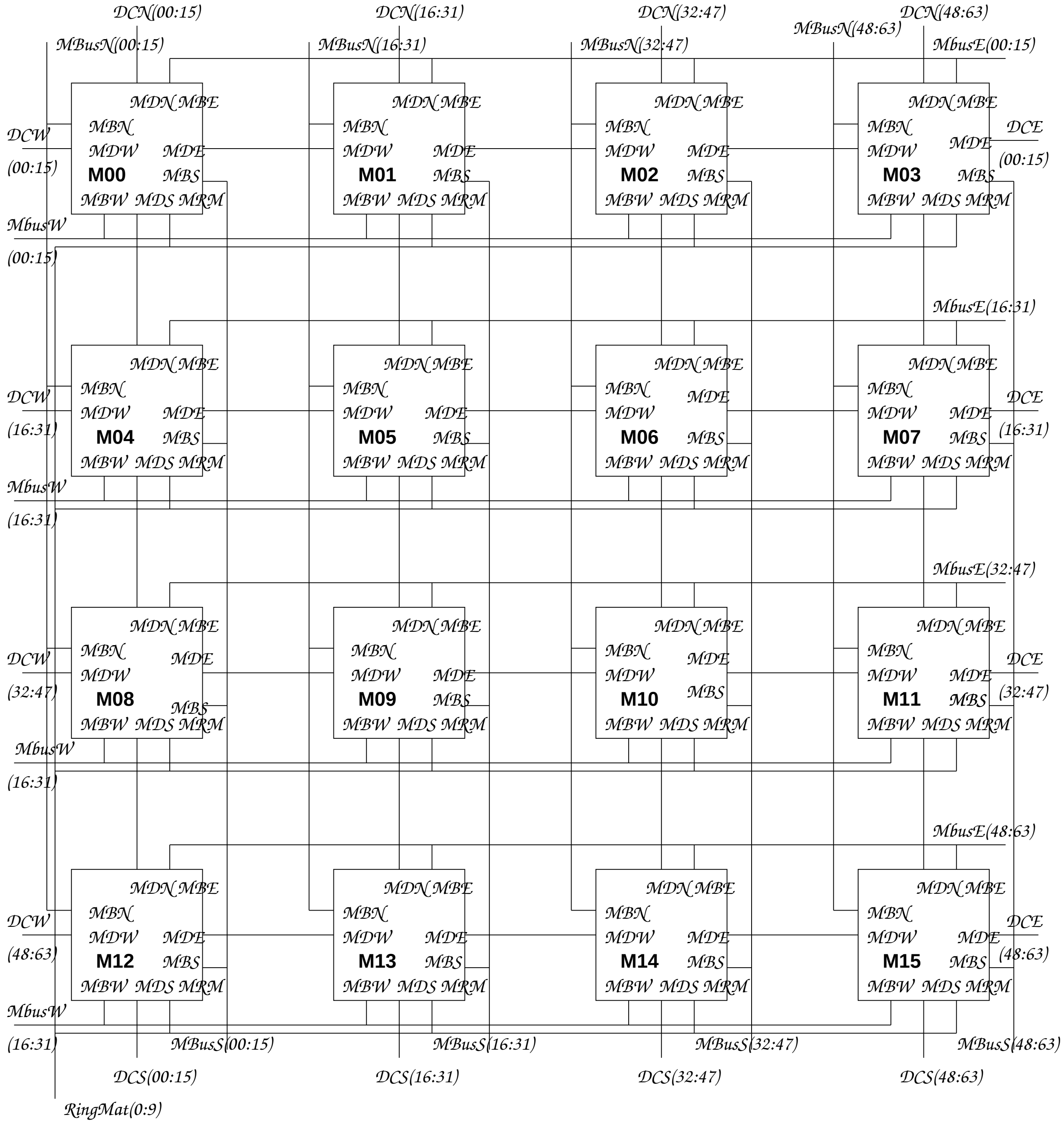

Figure 4.3: DECpeRLe-1 Matrix

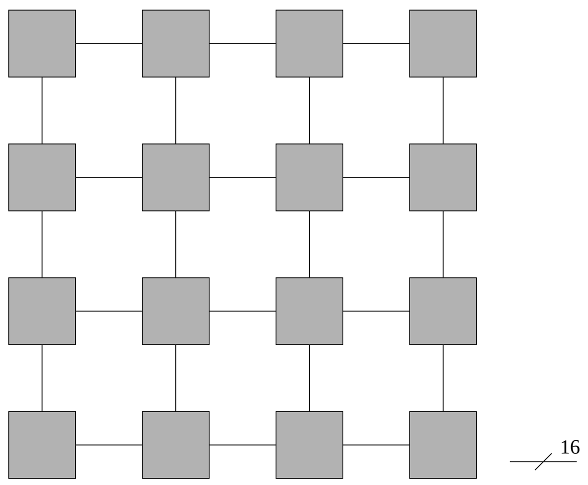

Figure 4.4: DECpeRLe-1 Direct Connection

on the corresponding side.

Rings

As shown in Fig.4.6, these wires connect all the matrix LCAs and two control LCAs, and are thus suitable for global control distribution. There are 10 such wires.

4.2.2 Switches and I/O buses

Fig.4.2 shows the way that FIFOs, RAM banks and the central matrix are connected through two 32-bit data buses and five programmable switches(FPGAs).

As shown in Fig.4.2, there is one matrix switch on each side of the matrix, respectively called North Switch(SWN), East Switch(SWE), South Switch(SWW)

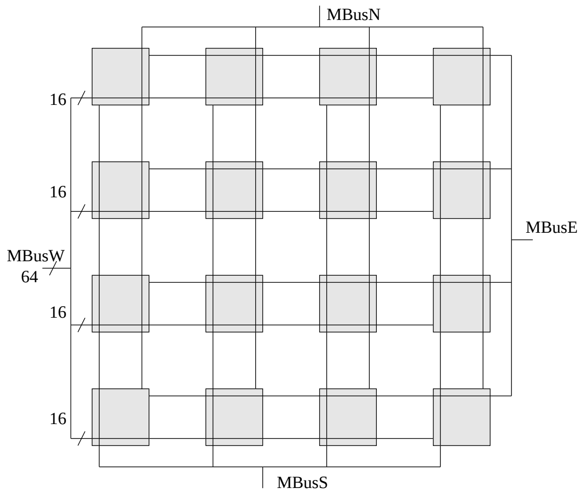

Figure 4.5: DECpeRLe-1 Matrix Data Buses

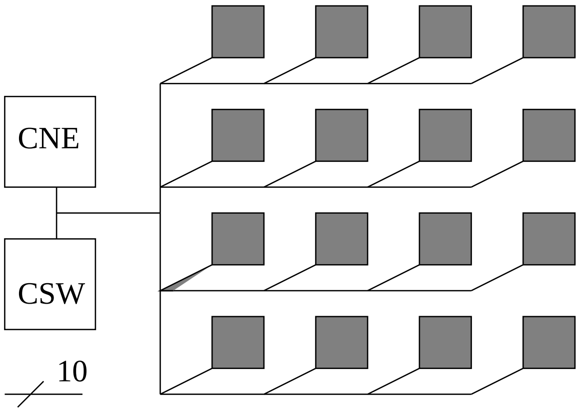

Figure 4.6: DECpeRLe-1 Matrix Rings

and West Switch(SWW), which connect the corresponding matrix data buses and correponding RAM banks. These 4 switches(*SWN*, *SWE*, *SWS*, *SWW*) also connect to two 32 bits I/O buses, called Northeast Bus(DBusNE) and SouthWest Bus(DBusSW) after the names of the switches they respectively connect.

Two I/O buses(DBusNE, DBusSW) connect to the input and output FIFOs through the fifth switch called FIFO switch(FSW), and also connect to corresponding controllers, called North-East Controller(CNE) and South-West Controller(CSW).

As their names imply, the FPGAs CNE and CSW are typically used to develop the controller of the application because they connect to all other parts of the DECpeRLe-1, FIFOs, Memory banks and other FPGAs.

4.2.3 Control Resource

The control resource is the programmable resource that can be used to develop the control part of the application other than datapath part. The datapath resource(matrix, RAM banks, FIFOs and switches) need the following set of control wires:

Matrix Rings: There are 10 matrix global wires.

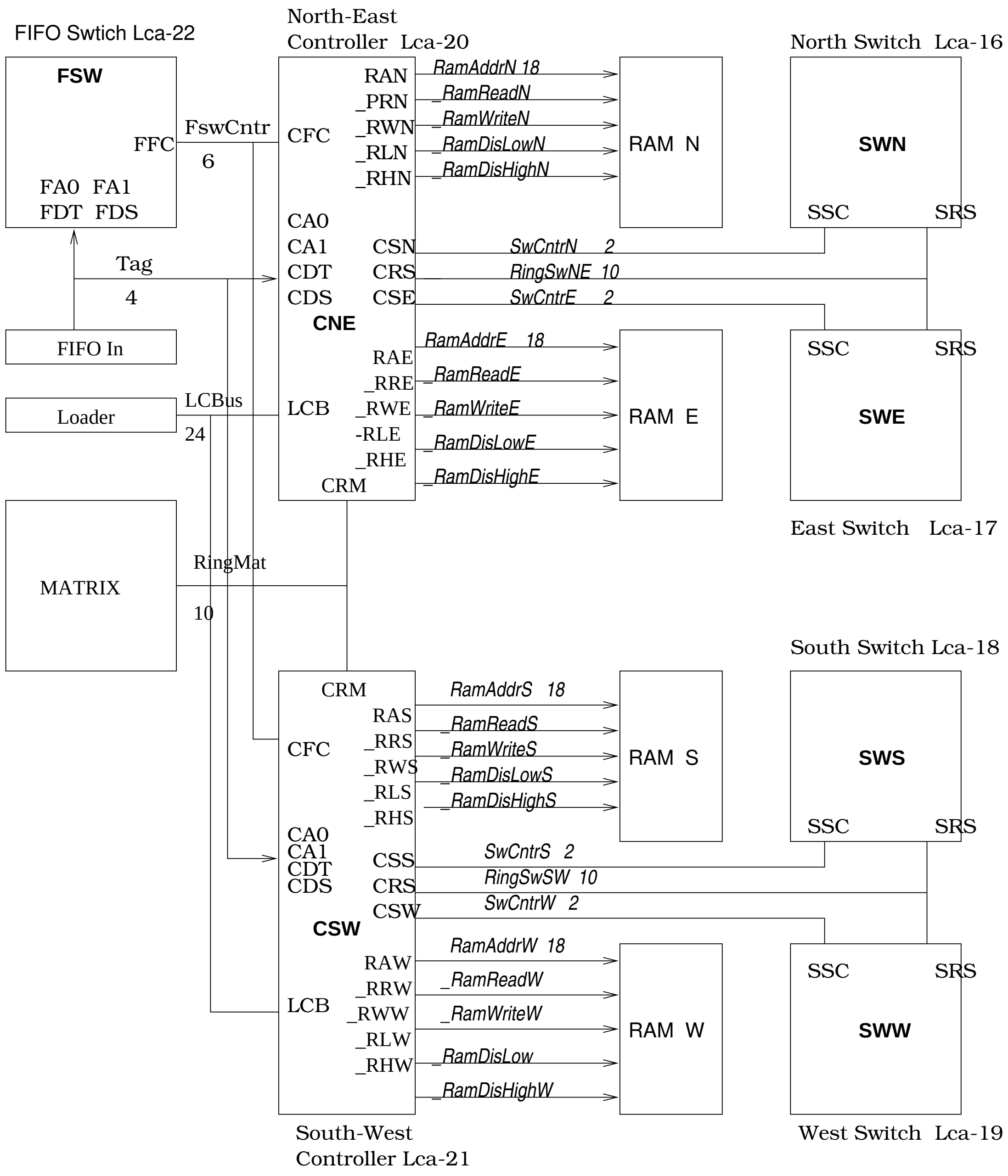

Figure 4.7: DECpeRLe-1 Control Wires

RAM Address: Each RAM bank has a 18-bit-wide address, that specify the word address of the current read or write operation.

RAM Controls: Each RAM bank has 4 control signals.

Switch Controls: Each pair of matrix switches(North and East/South and West) has 10 control wires that are the equivalent of the matrix rings, and are called switch ring. The FIFO Switch has 6 control wires of its own. In addition, each of the matrix switches has 2 dedicated control wires.

FIFO Controls: Each of the two FIFOs has one state wires: empty flag for input FIFO/full flag for output FIFO; and one control wire: write for output FIFO/ read for input FIFO.

TAGs: Four "tag" wires along the input data wires on the input FIFO.

Clock Control: The clock generator has two control wires that can be driven by the application design.

LCBus: There is a 24-bit-wide communication path between the board and the host.

As shown in Fig.4.7, all these control wires are connected to one of two controller LCAs(CNE, CSW) or both of them. These two controllers are identical except that each of them controls two of the four switches and memory banks. These two controllers also connect to corresponding I/O bus in order for it to be able to communicate with the main datapath.

4.2.4 Memory Subsystem

DECpeRLe-1 contains 4MB of high-speed static RAM organized in 4 banks of 256K 32-bit words(4 bytes a word). These banks are named North, South, East and West after the matrix switch to which they are connected. Each bank is completely independent of the others and has its own data, address and control signals:

Data Bus: 32 data wires connect to the corresponding matrix switch. They are represented by *RamdataX*, where X is one of N, S, E, W to respectively specify the North, South, East or West RAM bank.

Address Bus: 18 address wires(1MB = 2^{20} = 2^{18} x 2^{2}) connect to the corresponding controller. They are represented by $RamAddrX$.

Control Bus: 4 active-low control signals to specify the read/write operation, connect to the corresponding controller.

$Ram\bar{R}eadX$: read command.

$Ram\bar{W}riteX$: write command.

$RamD\bar{i}slowX$: disable lower half-word(bits 0 to 15).

$RamDi\bar{s}HighX$: disable upper half-word(bits 16 to 31).

The basic read and write transaction both last one clock cycle, and either may be performed at every cycle.

4.2.5 Host Interface

The DECpeRLe-1 application is running under the control of the software program executed on the host computer. The communication between DECpeRLe-1 application and its driving software program can be done through FIFOs or LCBus.

FIFOs

There is a 32-bit wide, 512-word deep FIFO in each direction. These FIFOs are respectively called input FIFO for the Host-to-PAM direction and output FIFO for the PAM-to-Host direction. On the application side, their data wires are connected to the FIFO Switch LCA and their control wires to the two Controller LCAs. Both FIFOs are purely synchronous devices when operated from the application side.

LCBus

The LCBus is a 24-bit wide general purpose register that can be read and written by both the software and the application design. The LCBus can be used for asynchronous communication between the Controller LCAs and the software program. Under the software control, the direction of each bit can be set independently of the others. Initially(after download), all bits are set for PAM-to-Host communication.

In the following chapters, I would describe details of my each step design of Function Decomposition. They would include algorithms, circuit diagram, state diagram and mapping.

CHAPTER 5

Optimal Design Variable Partitioning

The objective of the Variable Partitioning is to select such a set of bound sets that will give a good decomposition.

The variable partitioning can be realized as one of two types:

Partitioning of completely specified functions. (used in circuit design)

Partitioning of incompletely specified functions. (used mostly in machine learning)

Completely specified functions are those in which every possible combination of input variables has a fixed output. In other words the output is either a 1 or a 0. In FPGA mapping, Look up table's functions are this kind of functions.

Incompletely specified functions are those in which every possible combination of input variables, not only has a fix output, 1 or 0, but also has an undetermined output, "Dontcare", which will not affect the behavior of the circuit. "Dontcare"

are denoted by "X" or "-". In these type of functions, only those input combinations for which the output is either a 0 or 1 are useful. In machine learning, there may be as many as 200 variables. But most of their combinations are "Dontcare", these "Dontcare" sets are useless.

SLFD [6] lists eleven search stategies to solve the completely specified functions. The most representative is Wan Wei's method [6]. It is called the pair weighting method. This heuristic method will produce no more than 4 "good" partitions which are to be used for decomposition. The basic idea of this method is to arrange cubes(minterms) in the karnaugh map in such a way that they become more concentrated in either certain columns or rows. There are 3 steps of this method: a) calculate intensive cubes. b) calculate partition cubes. c) calculate the relationship factors.

5.1 Circuit Algorithm Description

In real life, majority of variable partition types are incomplete specified functions, for example, in machine learning. Here, a new method is introduced to solve this kind of problem. It is called etching method.(provided by Dr Perkowski). The etching method is straightforward and implementated easily in hardware. It uses the formular: Y $= |1_{X_i} - 0_{X_i}| - |1_{\bar{X}_i} - 0_{\bar{X}_i}|$

If Y=0, it means X_i and $\bar{X}_i$ have no difference.

Y $>= 0$, it means X_i and $\bar{X}_i$ are separated greatly, and X_i has more ON Set values; $\bar{X}_i$ has more OFF Set values.

Y $<= 0$, it means X_i and $\bar{X}_i$ are separated greatly, and X_i has more OFF Set values; $\bar{X}_i$ has more ON Set values.

1_{X_i} means the number of minterm is 1 in cofactor X_i.

0_{X_i} means the number of minterm is 0 in cofactor X_i.

$1_{\bar{X}_i}$ means the number of minterm is 1 in cofactor $\bar{X}_i$.

$0_{\bar{X}_i}$ means the number of minterm is 0 in cofactor $\bar{X}_i$.

This etching method can give a pattern of elimination search and reach the outcomes in a faster rate. It is not necessary to do an exhaustive calculation for the whole graph, just like complete function method.

Fig 5.1 is an example to show how to use this etching method. The graph in question is a binary graph for single value function. Keeping this in mind, the number of bound set variables for 4 input variables will be equal to 2 or 3. Let

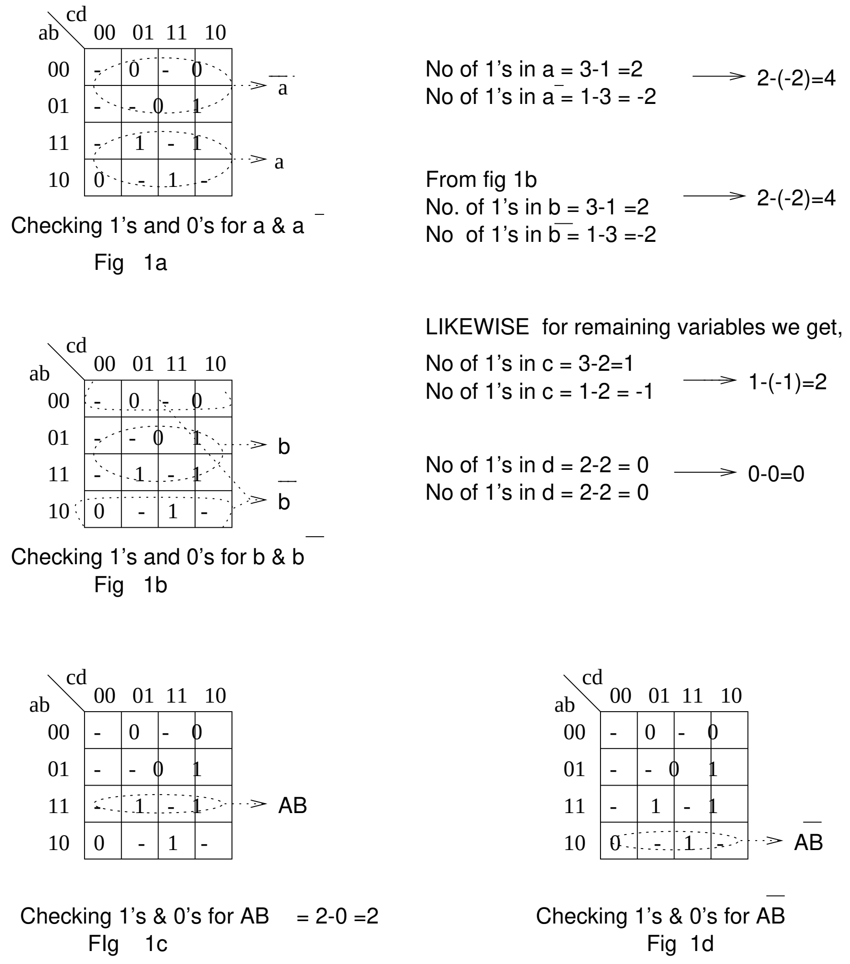

Figure 5.1: Diagram Approach to Finding Bound Sets

us assume that 2 bound set variables are selected. Now the question is which two are taken as the bound set.

Fig 5.2 shows the search tree of bound set.

First, calculate 1_A and 0_A when A is 1. Find the difference(called as result1) between these two numbers.

Second, calculate $1_{\bar{A}}$ and $0_{\bar{A}}$ when A is 0. Find the difference(called as result2)

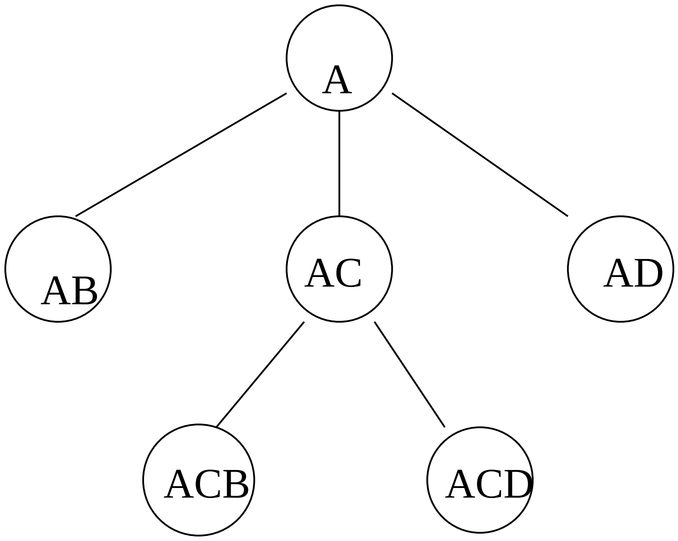

Figure 5.2: Search Tree for Finding the Bound Set

between these two numbers.

Third, find the difference between result1 and result2. This difference is the net weighting factor for the term "A".

Next step, respectively, by the same methods, calculate the net weighting factor for the terms "B, C, D".

From Fig 5.1, as the results shown, there is a tie between the net weighting factor of "A" and "B". (Both happen to be 4 in this case under consideration). Thus one of two of bound set varaibles is obtained and chosen to do a similar search for obtaining the other bound set variables.

The next question is which(B/C/D) is the best pairup variable with "A".

Let's choose B.

The pair"AB" is considered. Once again check for the weighting factor of this pair when A=1, B=1. Repeat the same process for the pair"A, $\bar{B}$" where A is 1, B is 0. Calculate the net weighting factor for AB.

This process is repeated for the other variable pair combinations with A.

The result of each of pair weighting factor is 2. It means any of the above mentioned pairs can be selected to result in a similar decomposition.

This example is very special and simple. Supposed if the function is more complicated, the pair up variable weighting factor should be calculated step by step until the bound set variables can be determined. But the process is still the same.

For example, there are five variables A, B, C, D and E. The bound set variables can be 4. So the calculating should be done in these situations as shown in Fig 5.3.

Right now, the etching method introduced here is limited to be run in the first step: calculating the net weighting factor for a single variable. Because this step is the most important. The value of the net weighting factor can show which is the bound set variable. The larger the value of the net weighting factor is, the more concentrated this single variable's ON set and OFF set is.

The variables whose the value of the net weighting factor is larger are chosen as bound set variables.

A	AB	ABC	ABCD
B	AC	ABD	ABCE
C	AD	ABE	ACDE
D	AE	ACD	BCDE
E	BC	ACE	
	BD	ADE	
	BE	BCD	
	CD	BCE	
	CE	BDE	
	ED	CDE	

Figure 5.3: Calculating the Bound Set Variable Array

5.2 Circuit Design Description

Figure 5.4 shows the circuit diagram. The circuit combines the following components.

Memory a is a 18-bit address bus and 32-bit data bus memory. It stores the initial all of variables' combinations and their corresponding specific minterm value. For example, there are 5 variables, A, B, C, D and E. If their combination is 01001 and their minterm value is 1, then the data input for memory a would be 00000013_{16}.

Memory b's size is the same as memory a. It is used to store every variable's weighting factors.

Counter 1 is to calculate the number of minterms whose value is 1 when the variable is Xi.

Counter 2 is to calculate the number of minterms whose value is 0 when the variable is Xi.

Counter 3 is to calculate the number of minterms whose value is 1 when the variable is $\bar{X}i$.

Counter 4 is to calculate the number of minterms whose value is 0 when the variable is $\bar{X}i$.

These four counters have a same pin that if *out* is 1, they can issue their results.

Counter 5 is to calculate the number of variables. Counter 6 is to calculate the number of all of minterms for one variable. Counter 5, counter6, comparator 5 and comparator 6 are used to determine the whole circuit one working cycle.

Comparator 1, Comparator 2, Comparator 3, Comparator 4, And 1, And 2, And 3 and And 4 constitute a network which is used to test whether one variable is Xi or $\bar{X}i$, one minterm is 1 or 0.

Substrator 1, Substrator 2 and Substractor 3 can constitute a network which is to calculate one variable's weighting factor.

There are 8 states in the variable partitioning circuit. Each state's content is

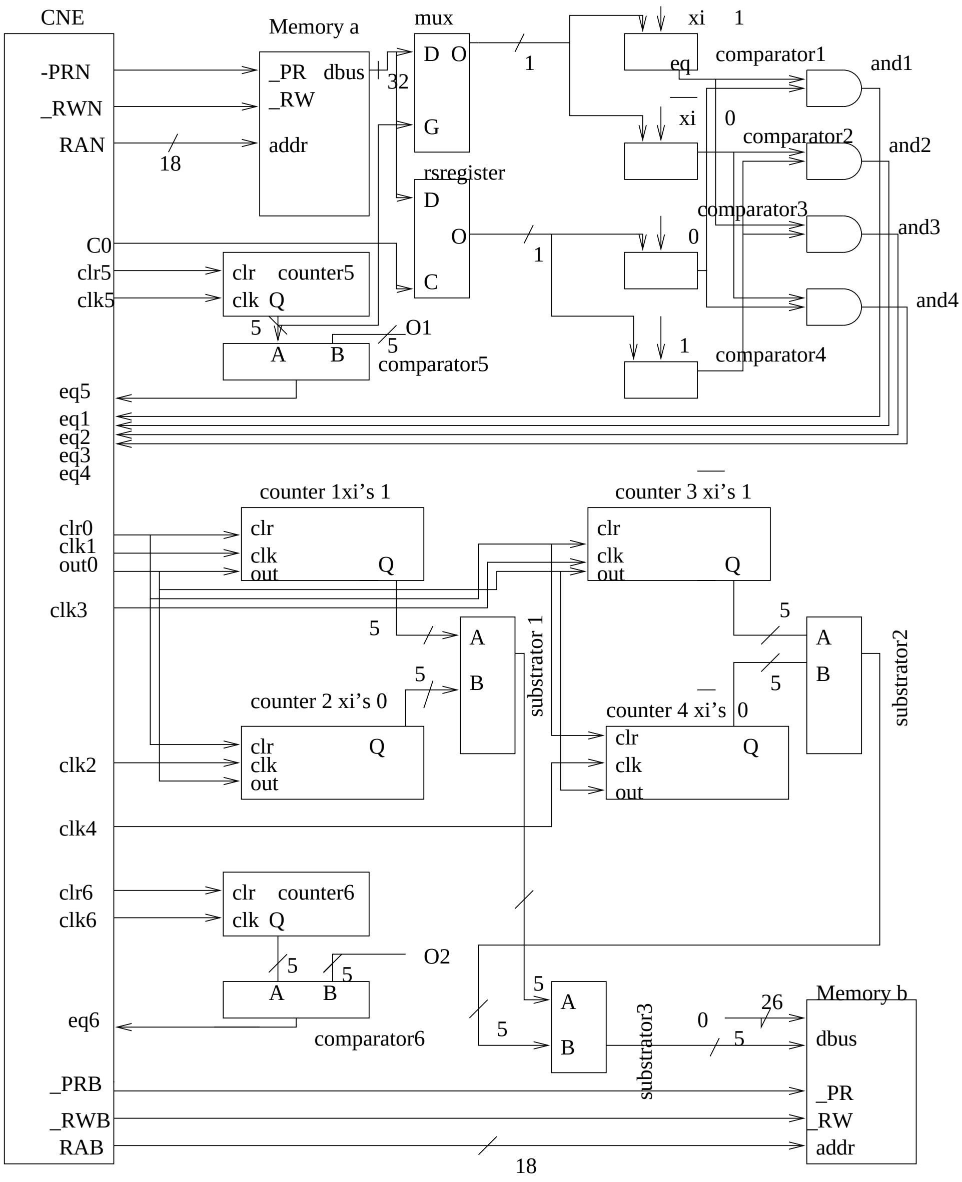

Figure 5.4: the Architecture of Variable Partitioning

shown as Fig 5.5. The whole control state diagram is shown as Fig 5.6. According to this state diagram, variable partitioning controller's VHDL code can be generated.

5.3 Mapping Variable Partitioning Circuit into DECpeRLe-1 Board

Placement and routing are the two key elements for a successful mapping. With Xilinx tools, the variable partitioning circuit can be mapped in DECpeRLe-1 Board.

The general placement problem is NP-complete, too. Hence, the algorithms used are generally heuristic in nature. [9]

Here, I use full customs method [9] to solve this problem. The whole variable partitioning controller would be put in CNE. Memory a is placed in RAM N.Memory b is placed in RAM E. There are 14 pins except memory control pins in this controller, so the whole data path is put in M03. M03 can be reached by SWN and SWE. The outline of mapping is shown in Fig 5.7.

Full customs method is my main methodology to place the other circuits into DECpeRle-1 Board. To the following mapping, I would try to minimize the net length, prevent two blocks from overlapping and utilize the whole DECpeRle-1 Board area fully to enhance the whole circuit performance.

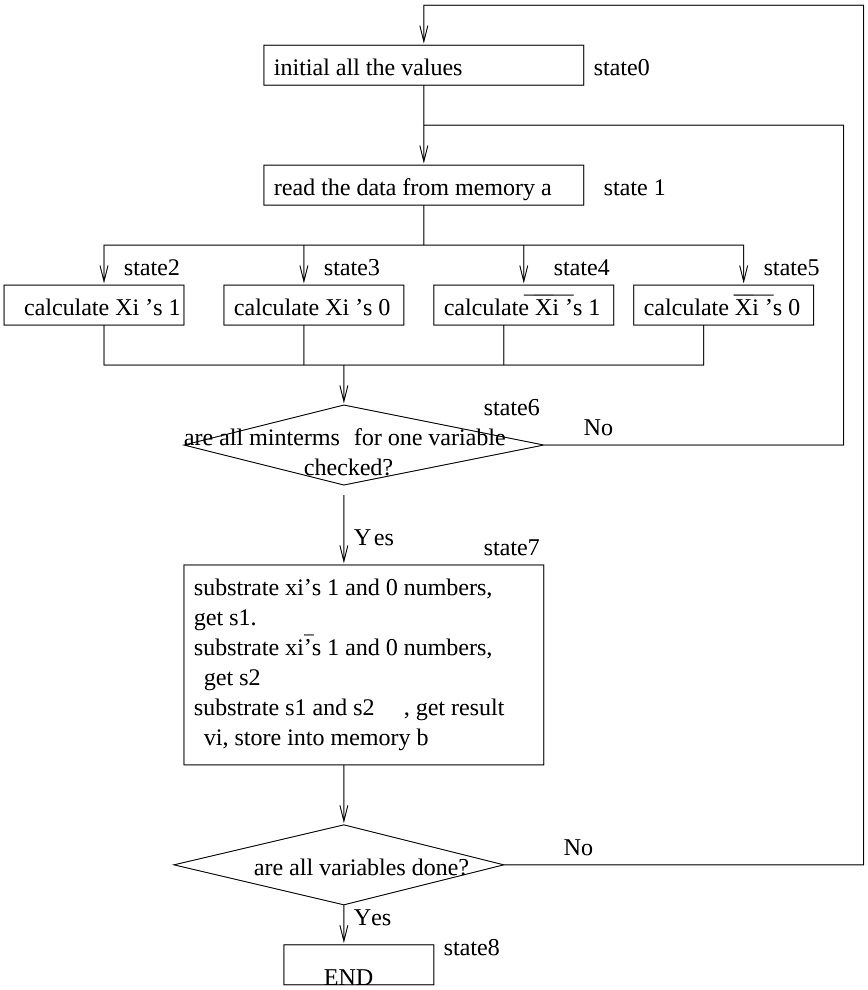

Figure 5.5: the Flowchart of Variable Partitioning

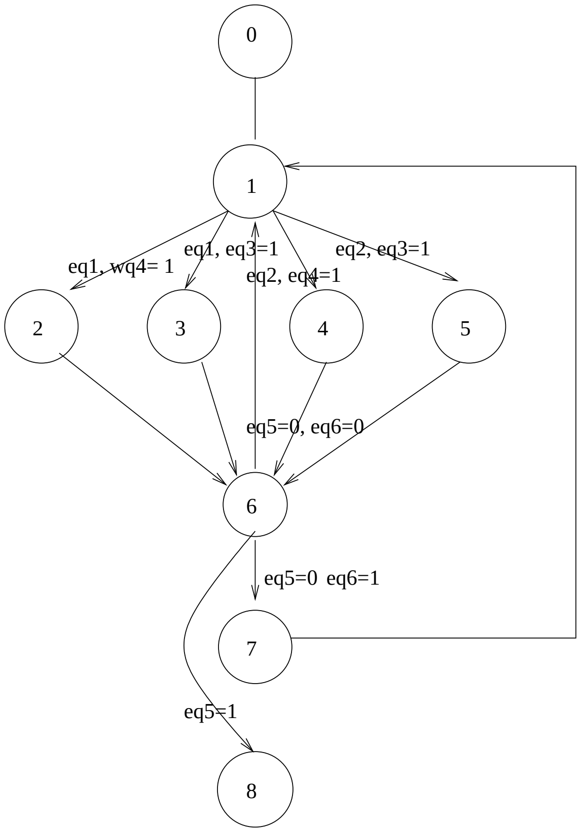

Figure 5.6: the State Diagram of Variable Partitioning

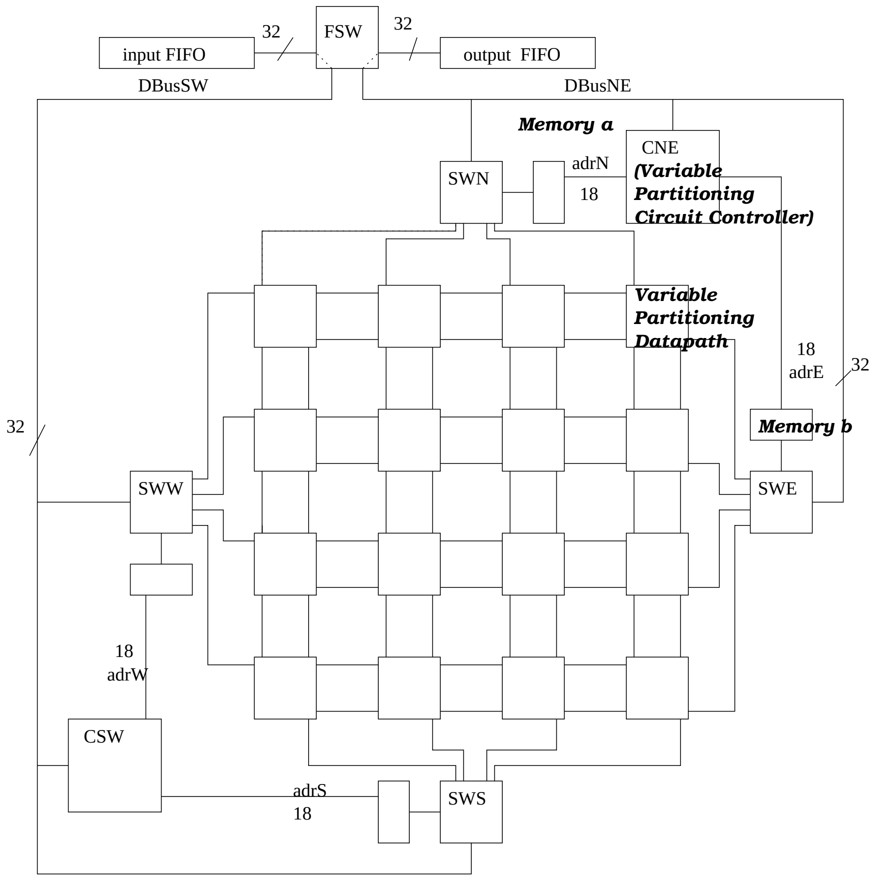

Figure 5.7: the Outline of Mapping

CHAPTER 6

Optimal Design Compatible Columns

In function decomposition, after the variables have been partitioned, the compatible columns should be found. How to find the compatible columns? The basic algorithm is that each column should be compared with other columns. If they are equal, they are compatible. If they are not, they aren't compatible.

One column is constituted by the minterms. There are three kinds of minterms: 1, 0, -(dontcare). -(dontcare) is defined to be equal to 1 or 0.

According to the standard, eight columns in figure 6.1 can be defined as follows:

AB \ CDE	000	001	011	010	110	111	101	100
00	-	-	1	0	-	-	-	0
01	1	0	0	1	0	0	0	-
11	0	1	-	1	0	1	1	-
10	-	-	1	0	1	0	-	1
	C1	C2	C3	C4	C5	C6	C7	C8

Figure 6.1: Five Variable Karnaugh Map

1. C1 is compatible with C8.
2. C2 is compatible with C3, C6, C7 and C8.
3. C3 is compatible with C2, C5 and C7.
4. C4 is independent.
5. C5 is compatible with C3 and C8.
6. C6 is compatible with C2 and C7.
7. C7 is compatible with C2, C3, C6 and C8.
8. C8 is compatible with C1, C2, C5 and C7.

6.1 Circuit Algorithm Description

Figure 6.2 has shown the process of finding compatible coulmns.

Memory a and memory b are two input data memories. They all store the same data. Their data are the column's minterm values. Their address is the column's number. Each column data in memory a should be compared with all the columns data in memory b. Notice: Each column data has an opportunity to be compared with itself. This set up is for the independent column. If the two columns' minterm values are equal, memory b's data corresponding address should be put into memory c. Each compatible columns group in memory c's result should be separated by the value "1111".

The whole circuit flow chart is shown as figure 6.3

1) input data in memory a.

Memory a

address		data(minterm)
C1	0000	-10-
C2	0001	-01-
C3	0010	10-1
C4	0011	0110
C5	0100	-001
C6	0101	-010
C7	0110	-01-
C8	0111	0--1

Memory b

address		data(minterm)
C1	0000	-10-
C2	0001	-01-
C3	0010	10-1
C4	0011	0110
C5	0100	-001
C6	0101	-010
C7	0110	-01-
C8	0111	0--1

Comparator

Memory c

address	data	address	data	address	data
00000000	0000	00001100	0010	00011001	0110
00000001	0111	00001101	0100	00011010	0111
		00001110	0111		
00000010	0001				
00000011	0010	00001111	0001		
00000100	0101	00010000	0101		
00000101	0110				
00000110	0111	00010001	0001		
		00010010	0010		
00000111	0001	00010011	0101		
00001000	0010	00010100	0110		
00001001	0100	00010101	0111		
00001010	0110				
		00010110	0000		
00001011	0011	00010111	0001		
		00011000	0100		

Figure 6.2: Compatible Columns Circuit Process

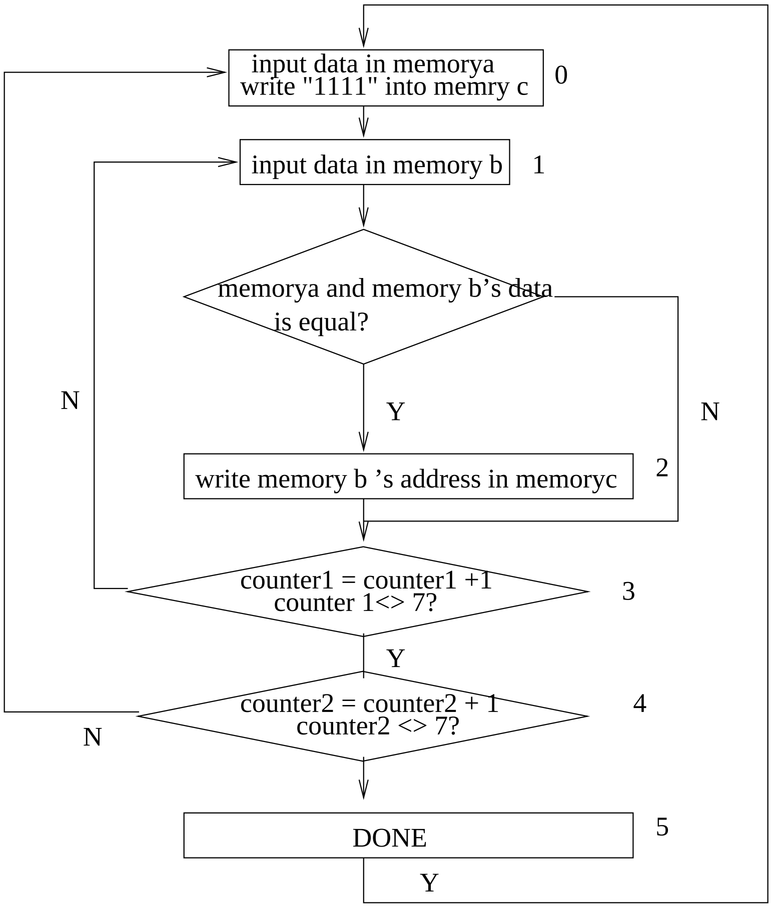

Figure 6.3: the Flowchart of Compatible Column Circuit

2) input data in memory b.

3) compare data in memory a and memory b. If the data in memory a and memory b are equal, write b's address in memory c, go to step 4; If they are not equal, directly go to step 4.

4) counter 1 plus 1, compare the value in counter 1with 7. If counter 1's value is equal to 7, (it means one column has finished comparison with other columns) write "1111" into memory c, go to step 5; If they are not equal, directly go to step 2.

5) counter 2 plus 1, compare the value in counter 2 with 7, if counter 2's value is equal to 7, the whole process is done. If they are not equal, directly go to step 1.

6.2 Circuit Design Description

The whole circuit's architecture is shown as figure 6.4. It consists of these components.

Memory a and memory b are 4-bit data bus and 4-bit address bus. They are all deposited the column data. Memory c is a 4-bit data bus and 8-bit address bus which are used to store the result. Each group result is separated by "1111".

The four comparators are all 1x1 bit comparator. They are used to compare the minterm value in memory a and memory b. Comparator 5 to Comparator 6 are all 3-bit 3x3 comparator. They are used to compare counter 1 and counter 2

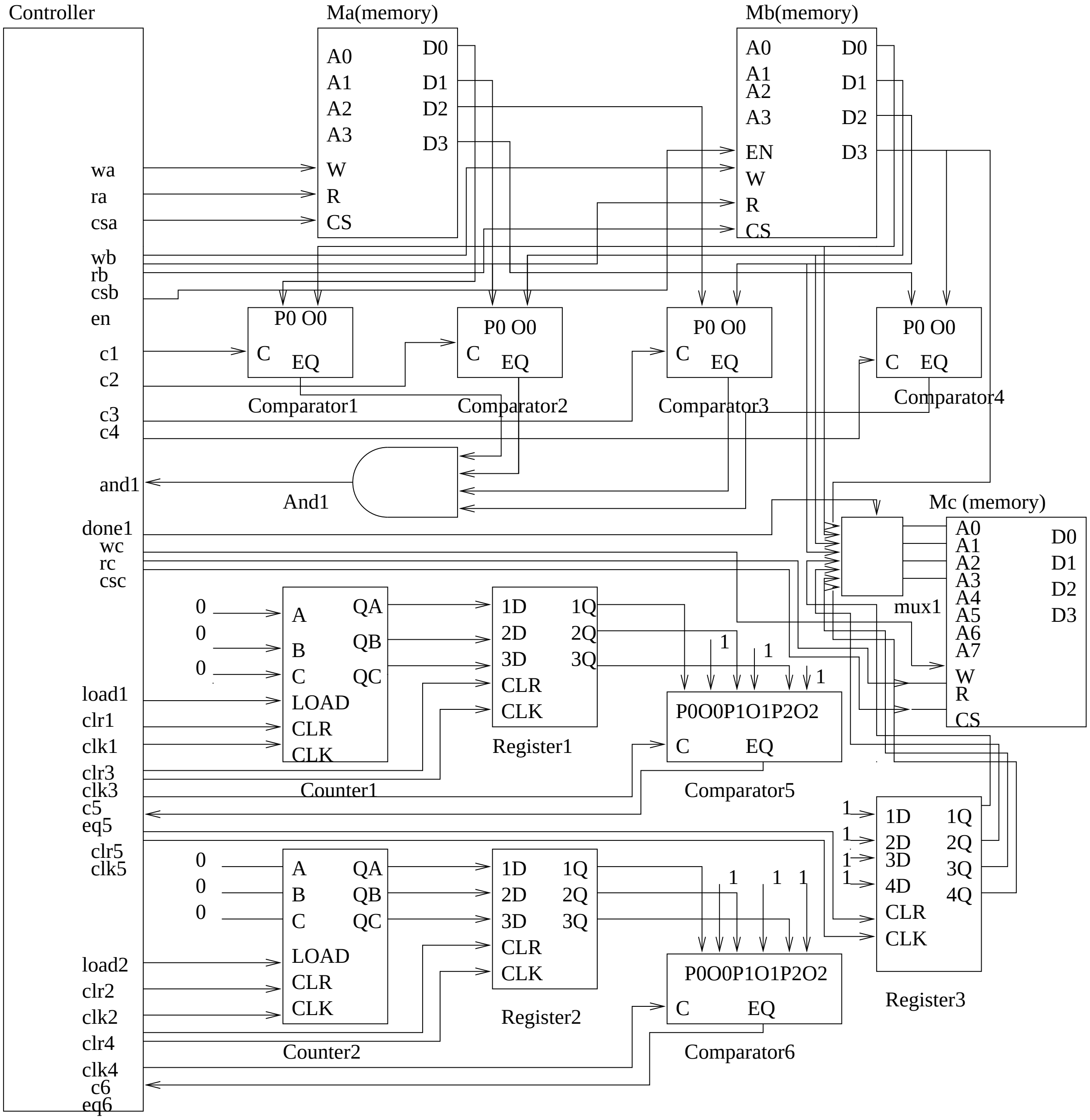

Figure 6.4: the Architecture of the Compatible Column Circuit

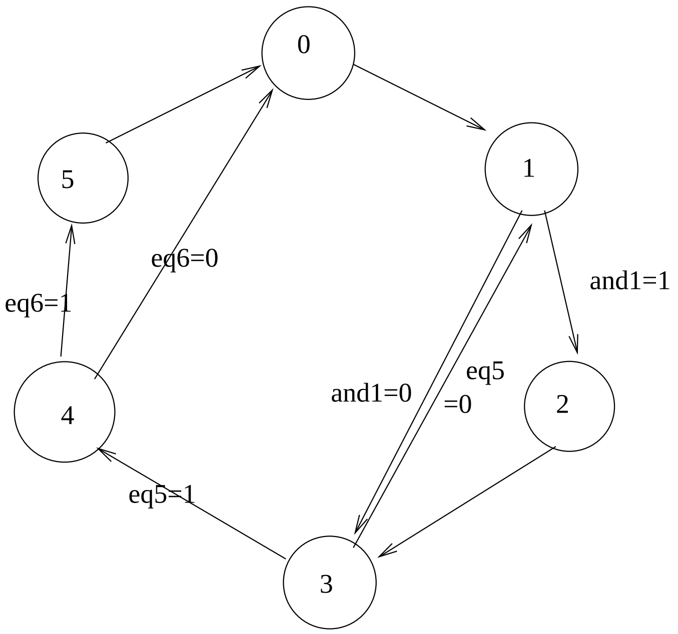

Figure 6.5: the State Diagram of the Compatible Column

with "111".

Gate and1 is four fan-in and gate. The fan-in signals are from the Comparator 1 to Comparator 4.

Counter 1 is used to count all the number of memory b. Counter 2 is used to count all the number of memory a.

Register 1 and register 2 are buffers of counter 1 and counter 2. Register 3 is 4-bit register to transfer signal "1111" in memory c after each compatible column group searching is done.

Control unit is designed as figure 6.5.

The state transfer table is shown as figure 6.6.

Present State	Next State						
	clock	and1		eq5		eq6	
	1	0	1	0	1	0	1
state0	state1						
state1		state3	state2				
state2	state3						
state3				state1	state4		
state4						state0	state5
state5	state0						

Figure 6.6: State Table for Control Unit

6.3 Mapping Compatible Columns Circuit into DECpeRLe-1 Board

The outline of mapping is shown in figure 6.7. As shown in the figure, for using Matrix as regular as possible, only datapaths are mapped in it; The controller is mapped in South-West controller(CSW) because it is easy to control input and output FIFOs and all matrix chips from switches. All other parts are shown in the figure. The components of DECpeRL-1 board that are represented by the dotted line are not used in this design.

6.4 Summary

If this compatible columns circuit adapts to more than 20 input variables, th e controller design isn't changed. Only the memories, registers and counters wo uld be added more bits to adapt to more contents. The whole design algorithm is still the same.

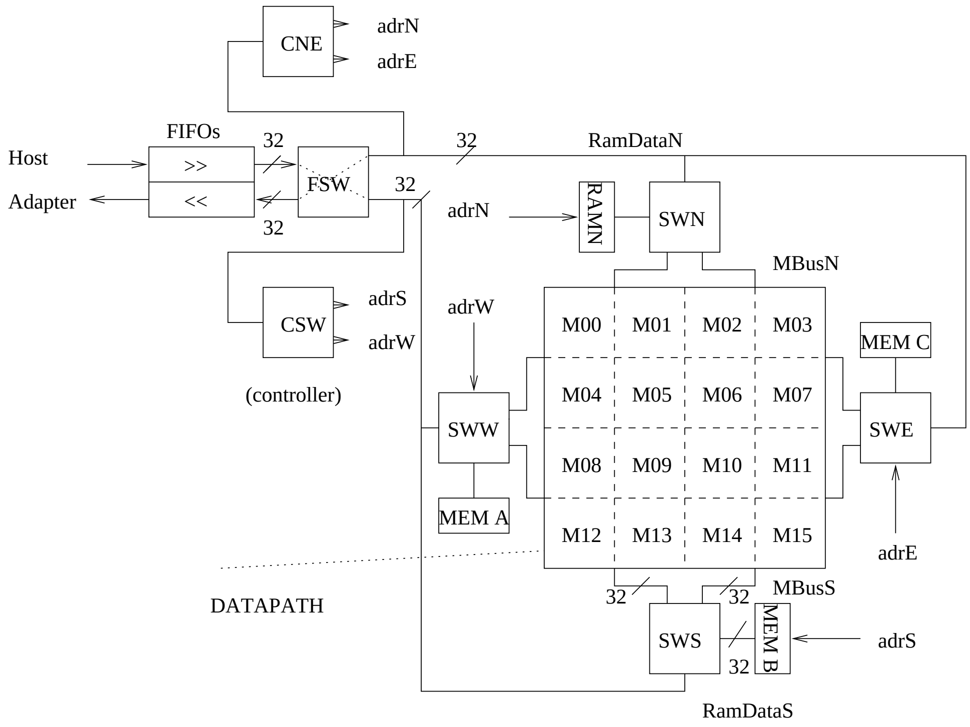

Figure 6.7: the Outline of Mapping

CHAPTER 7

Optimal Design Maximum Clique Circuit

The area of maximum clique generation is very competitive, even more than other steps. There are, basically, two groups of methods. Some methods start from small cliques and grow them, other methods start from large groups and remove nodes until cliques are found. It seems that it depends on the type of graph which method is better. Perhaps for graphs of Boolean functions it is better to start from small cliques and grow them instead of starting from large groups and shrinking them. In any case, which particular method is better, depends perhaps on the desity, type and other properties of the graph.

People use maximum clique technology to find the best combination of compatible columns in function decomposition. After these best combination of compatible columns are found, people can encode these columns to realize G function circuit. the best combination of compatible columns' standard is that the number of encoded bits of the combinations of compatible columns should be smaller than the number of variables of bound set.

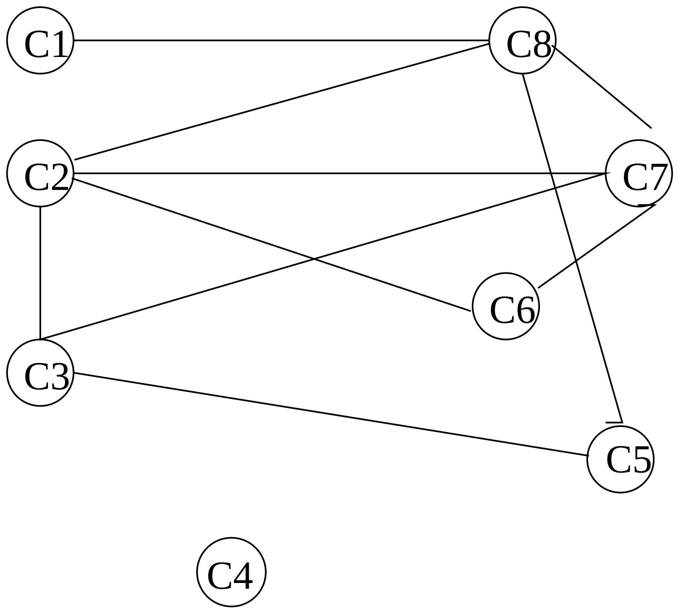

Figure 7.1: Compatible Column

7.1 Circuit Algorithm Description

A complete sub-graph of G is called a clique of G. . Clique partitions problem is to find the minimum set of disjointed cliques that cover all nodes of the given graph. Sometimes the solution for clique partition is not only one.

Figure 7.1 shows an example of clique concept. It comes from figure 6.1.

From the definition, there are many choices for the clique. For example, Node 1 and node 8 can be a clique.

Node 2, node 6 and node 7 can be a clique.

But node 2, node 6, node 7 and node 8 couldn't be a clique, because node 6 and node 8 are not compatible.

Other choices are, Node 2, node 7 and node 8 can be a clique.

Node 2, node 3 and node7 can be a clique.

Node 3 and node 5 can be a clique.

Node 3 and node 7 can be a clique.

Node 4 can be a clique.

There is no answer for which one is better. the best answer should be derived from analyzing the real object.

According to these relationship between the eight columns, figure 7.2 can be got to be data member for maximum clique's circuit. the compatible columns are defined as 1, the incompatible columns are defined as 0.

Put these data structures into two memories, memory a and memory b. In memory a, take the C1 to compare with every column in memory b. then get all the columns which are compatible with C1.

Continue to take one column left in memory a to compare with one column in memory b. then get the all compatible column.

The whole process is in figure 7.3. The main trick pat of the maximum clique circuit is how to find out the two lines data in figure 7.2 compatible and intersection.

For example, when memory a's data is line 1, in memory b, line 1 and line 8

	C1	C2	C3	C4	C5	C6	C7	C8
C1	1	0	0	0	0	0	0	1
C2	0	1	1	0	0	1	1	1
C3	0	1	1	0	1	0	1	0
C4	0	0	0	1	0	0	0	0
C5	0	0	1	0	1	0	0	1
C6	0	1	0	0	0	1	1	0
C7	0	1	1	0	0	1	1	1
C8	1	1	0	0	1	0	1	1

Figure 7.2: Compatible Column in Maximum Cliques

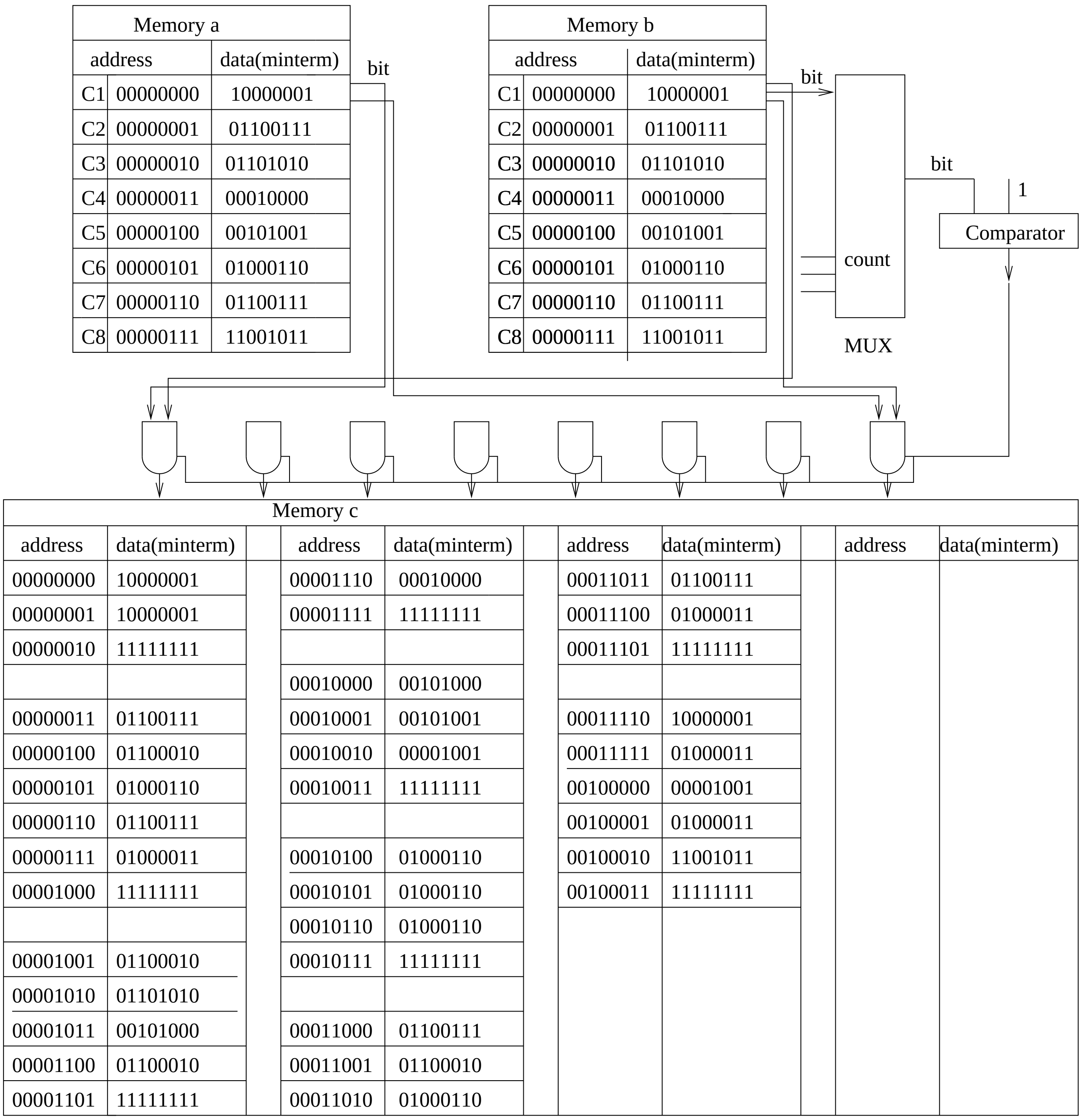

Figure 7.3: the Process of Maximum Cliques

should be found compatible because (1,1) =1 and (8,1) =1. Other dots (x,1) = 0.

When memory a's data is line 2, line 2, line3, line6, lin7 and line8 in memory b are compatible because their coresponding dots(x, 2) =1.

Figure 7.4 is the flow chart of maximum clique's technique.

7.2 Circuit Architecture

Figure 7.5 is the architecture of the maximum clique circuit.

There are three 8x8 memories. They are memory a and memory b whose data structure is as figure 7.2. Memory c is used to store the results.

There are eight 2-bit and gates. They are used to intersection of memory a and memory b.

Two 3-bit counters. Counter 1 is used to calculate the number of data in memory a. In the meantime, each time the result of counter 1 is used to be the number of mux1 select signal "count". Counte 2 is used to calculate the number of data in memory b.

Mux1 is a 8-bit multiplexer. Its input signal is the data of memory b. The selected signal comes from the counter 1's result.

Three comparators. Comparator 1 is 2-bit comparator. It is used to compare the correponding bit of the data in memory b and 1 to check the compatiblity of the two columns. Comparator 2 and comparator 3 are used to chaeck the counte

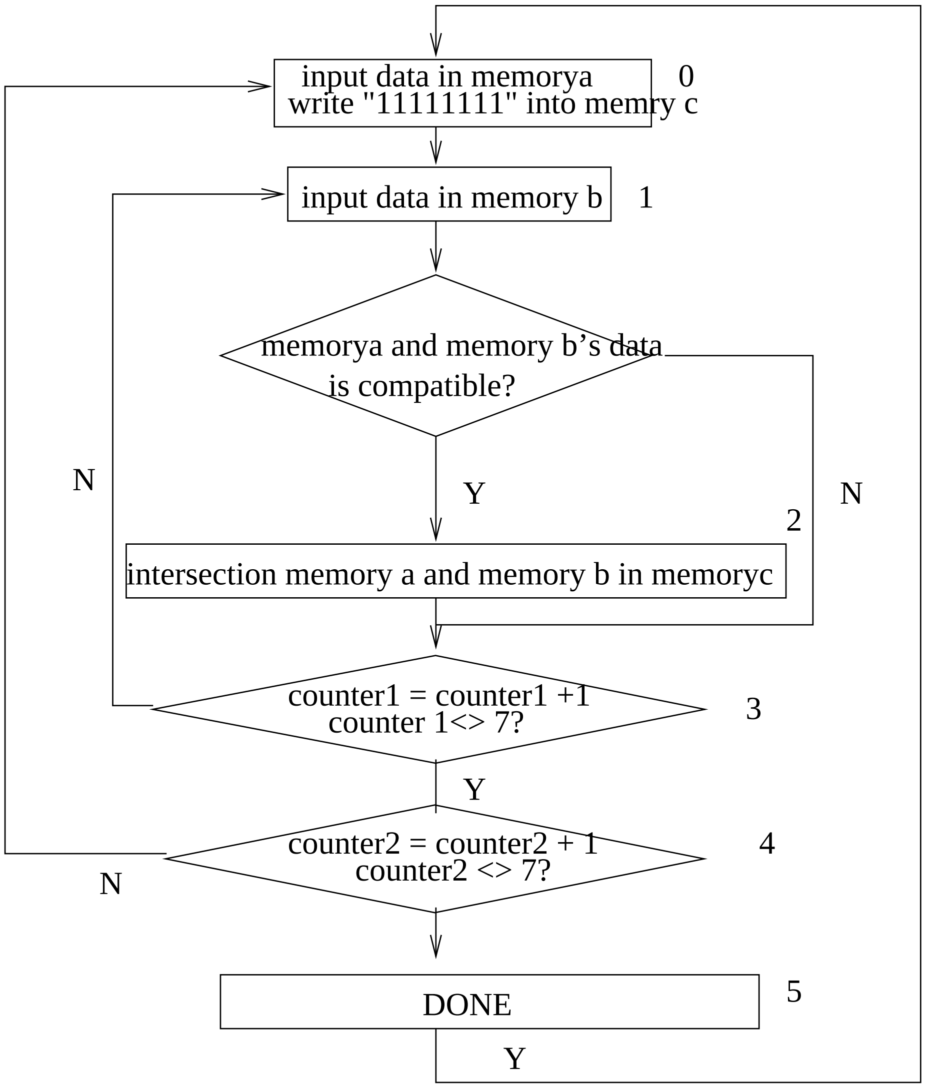

Figure 7.4: the Flowchart of Maximum Cliques Circuit

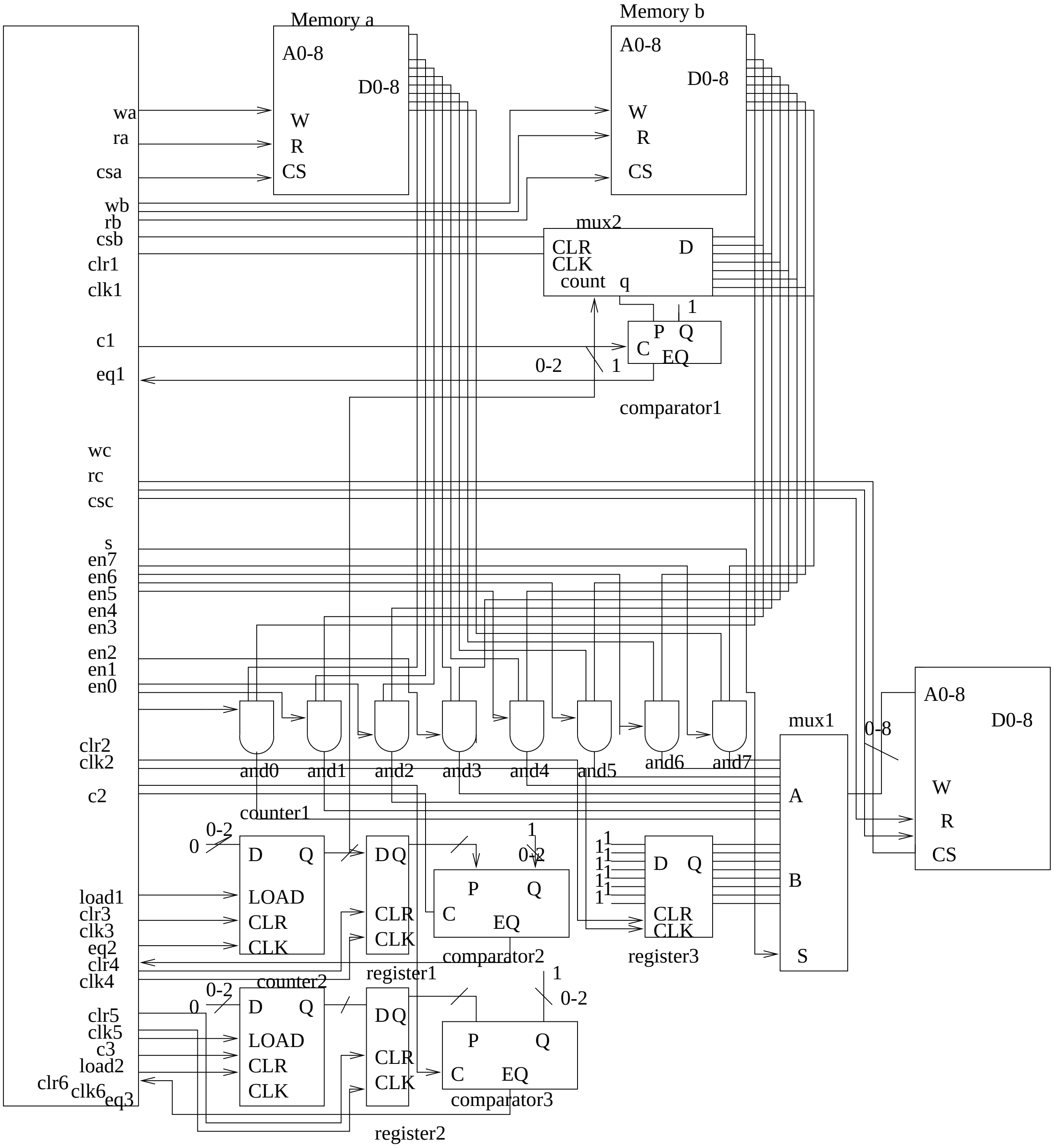

Figure 7.5: the Architecture of Maximum Cliques Circuit

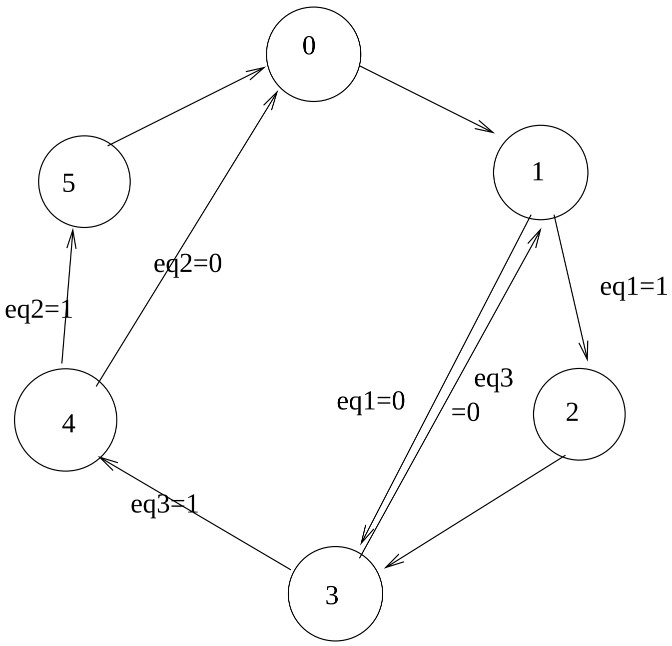

Figure 7.6: the State Diagram of Maximum Cliques Circuit

1 and counte 2 calculations.

Three registers are used to be buffers for calculations.

Figure 7.6 shows the control unit diagram.

Figure 7.7 shows the state transfer table for the maximum clique controller.

7.3 Mapping Maximum Cliques Circuit onto DECpeRle-1 Board

The outline of mapping is shown in figure 7.8. As shown in the figure, for using Matrix as regular as possible, only the datapaths are mapped in it; The controller is mapped in South-W est controller(CSW) because it is easy to control input and output FIFOs and matrix chips from switches. All other parts are shown in

Present State	Next State						
	clock	eq1		eq2		eq3	
	1	0	1	0	1	0	1
state0	state1						
state1		state3	state2				
state2	state3						
state3						state1	state4
state4				state0	state5		
state5	state0						

Figure 7.7: the State Table for Control Unit

the figure. The compo nents of DECpeRL-1 board that are represented by the dotted line are not used in this design.

7.4 Summary

If this maximum cliques circuit adapts to more than 20 input variables, th e controller design isn't changed. Only the memories, registers and counters wo uld be added more bits to adapt to more contents. The whole design algorithm is still the same.

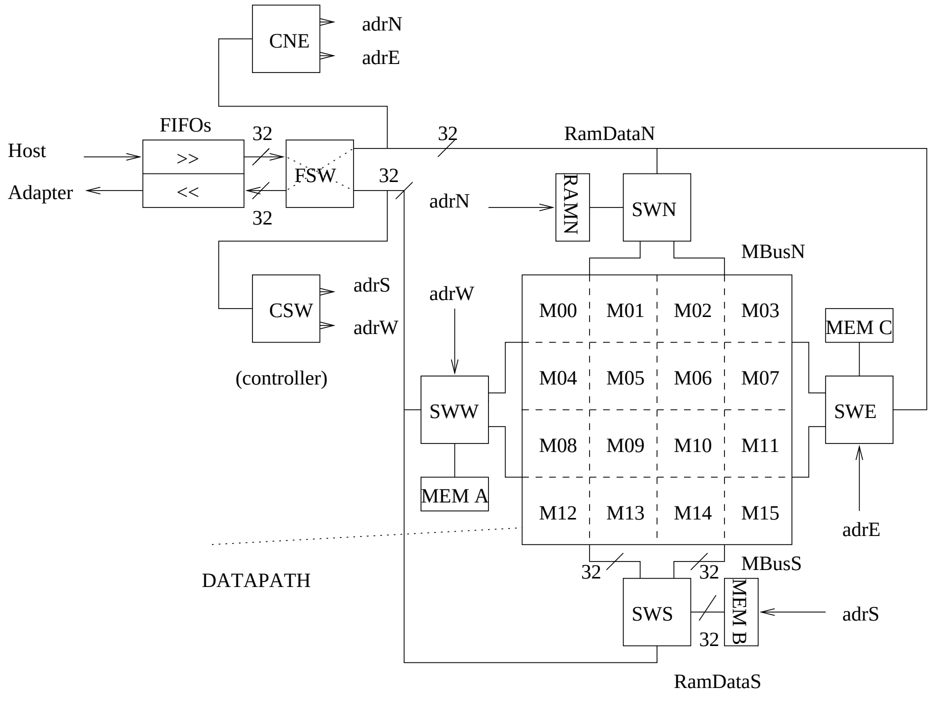

Figure 7.8: the Outline of Mapping

CHAPTER 8

Optimal Design G Function Block

In function decomposition, G function block are constituted by the bound set. With maximum cliques technique, the number of G function's bits should be smaller than the number of bound set's variables.

Figure 6.1 is still used to be an example to show how to design G function circuit.

In this karnaugh map, eight columns from C1 to C8 can be encoded as following:

C1 and C8 are encoded as 00.

C2, C6 and C7 are encoded as 01.

C3 and C5 are encoded as 11.

C4 is encoded as 10.

Figure 8.1 illustrates this relationship.

Because there are two encoding bits, G function block should be constituted by these two bits: G1 and G2. G1's circuit is to find all of variables' combination in bound set whose encoding G1 bit's logic value is equal to 1. G2's circuit is

C D E	000	001	011	010	110	111	101	100
	C1	C2	C3	C4	C5	C6	C7	C8
G1 G2	00	01	11	10	11	01	01	00

Figure 8.1: Bound Set, Columns and Encoding Bits

the same as G1's circuit. So G function block circuit consists of these two bits circuits.

8.1 Circuit Algorithm Description

Figure 8.2 illustrates G1's circuit process.

The data member structure of memory a is derived from figure 8.1. The variables (C, D, E) combination of bound set are treated as address. Their corresponding G1 bit's value is the value of memory a. If G1 bit's value is 1, this memory a's value is "0001". If G1 bit's value is 0, this memory a's value is "0000".

Each time, memory a's data is taken out to be compared with value "0001". If they are equal, put the memory a's address, which is C, D, E combination of bound set to memory b's data. If they are not equal, put the memory a's address to memory c. Other next data in memory a are taken out to be compared with value "0001".

Because there are 3 variables in bound set, these comparisons should be held

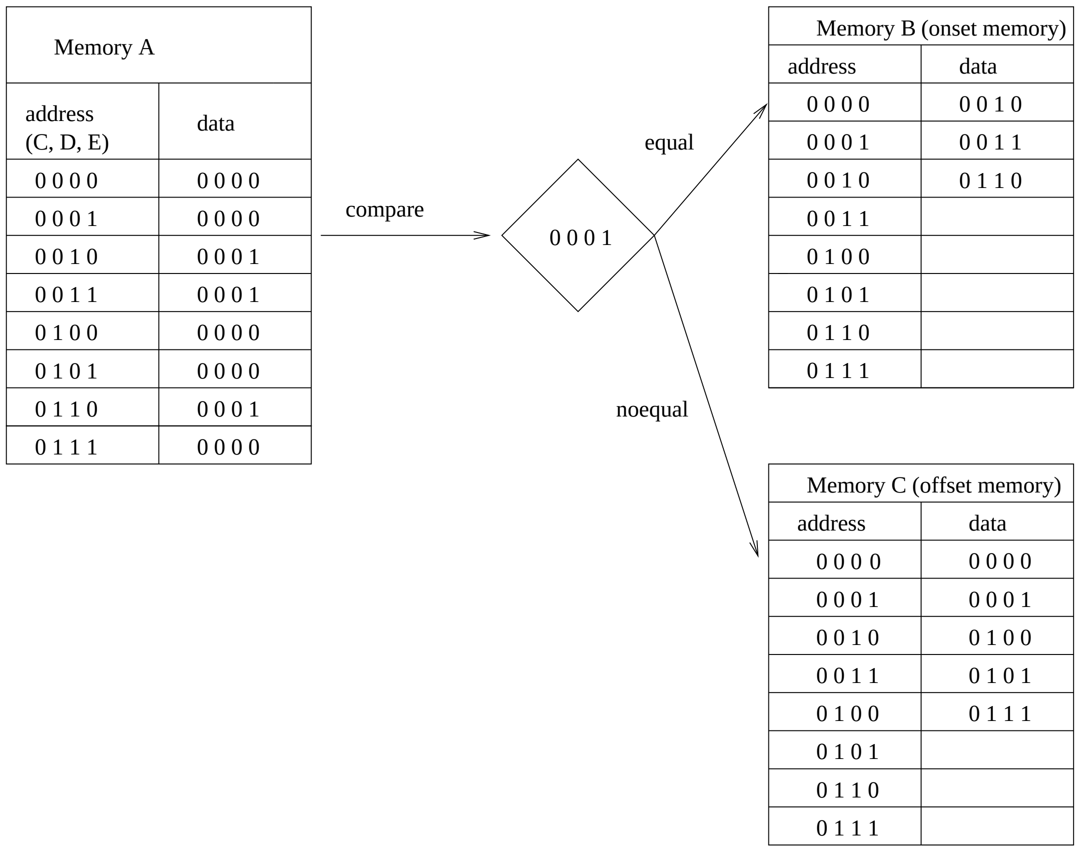

Figure 8.2: G1 Circuit Process

2^3 times. In the end, memory b stores all of variables combination of bound set whose G1's value is equal to 1, and memory c stores all of variables combinations of bound set whose G1's value is equal to 0.

Figure 8.3 is a flow chart of G1's circuit algorithm.

1) put the $2^3 - 1$ into register 1 and 0001 into register2.

2) initial counter 1 value as 0.

3) read one memory content from memory a.

4) compare this memory content with value "0001". If they are equal, write this memory content's address in memory a into memory b. If they are not equal, write this memory content's addresss in memory c. The counter 1's value automatically adds 1.

5) repeat step 3 and step 4, until counter 1's value is equal to the register 1.

8.2 Circuit Design Description

This circuit uses components as following:

Three memories are memory a, memory b aand memory c. Memory a stores all of bound set(C, D, E)'s combinations and their corresponding G1's value. Memory b stores all of bound set (C, D, E)'s combinations whose corresponding G1's value is equal to 1. Memory c stores all of bound set(C, D, E) combinations whose corresponding G1's value is equal to 0. The address bus of these memories

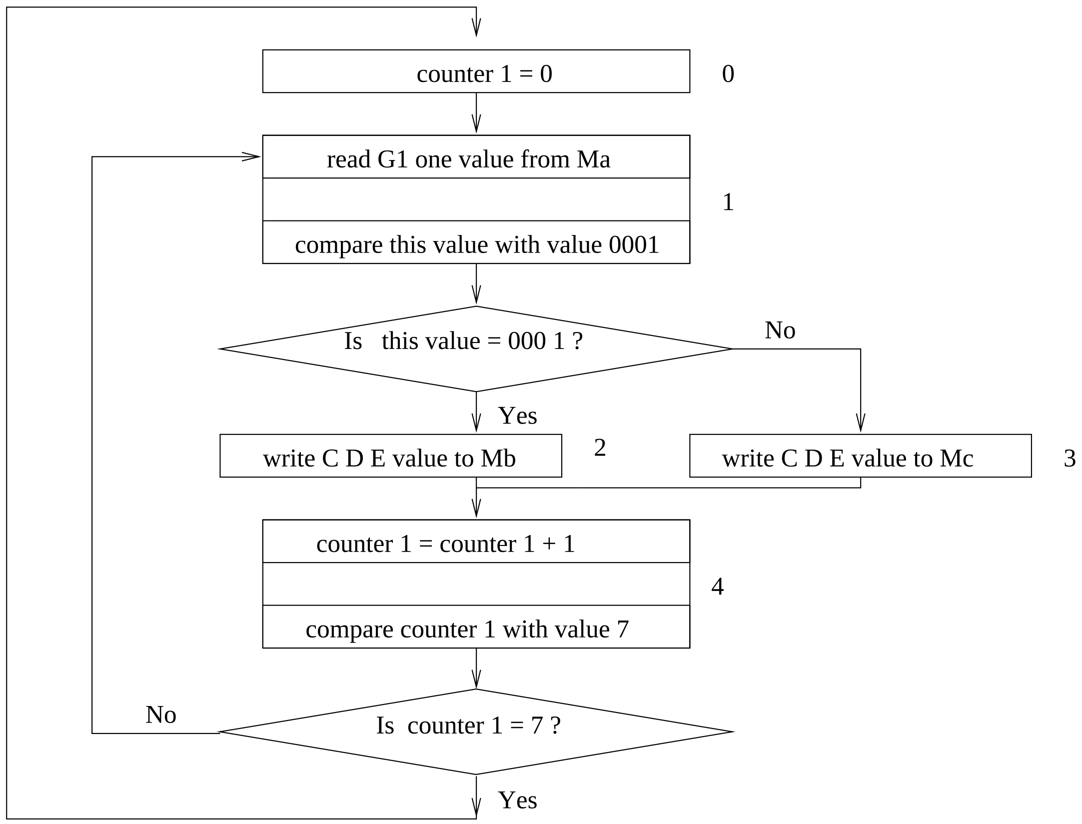

Figure 8.3: Flowchart of G1 Circuit Algorithm

is 4-bit, its msb is defined as "0"; The data bus is 4-bit, its first three bits (from left to right) are set up as "000", only the last bit is set up as G1's value, "1" or "0".

One counter is counter 1, it is used to count the number of the value variable combinations of the bound set which have been compared. Counter 1 is a 3-bit counter.

Two comparaotrs are 8-bit comparator 1 and 6-bit comparator2. Comparator 1 is used to compare the 4-bit data bus value from memory a with the value "0001". If they are equal, the comparator 1 outputs signal *eq1*, its calue is set as "1". If they are not equal, the comparator 1 outputs signal *eq1* as "0". This signal *eq1* is transported to the control unit of the circuit. Comparator 2 is used to compare the counter 1's value with the whole number of the variable combination(C, D, E), If they are equal, comparator2 outputs signal *eq2* as "1"; If they are not equal, it outputs signal *eq2* as" 0". This signal *eq2* is transported to control unit of the circuit. The control unit decides the state transfer according to these two signals.

Two registers are register 1 and register 2. Register 1 is 4-bit register. It is a buffer for comparator 1. Register 2 is 3-bit register. It is a buffer for comparator 2.

Figure 8.4 is a diagram of the circuit architecture. It includes the control unit and datapath unit. It has shown the control unit's output signals and datapath unit's output signal. The control unit's input signals are the datapath unit's output signals. The control unit's output signals are the datapath unit's input signals.

Each state's content has been shown as in figure 8.4. The control unit's state transfer diagram can be found as figure 8.5. This control unit of the circuit has 5 states.

8.3 Mapping G1 Function Circuit into DECpeRL-1 Board

The outline of mapping is shown in figure 8.6. As shown in the figure, for using Matrix as regular as possible, only the datapaths are mapped in it; The controller is mapped in South-West controller(CSW) because it is easy to control input and output FIFOs and all matrix chips from switches. All other parts are shown in the figure. Th e components of DECpeRL-1 board that are represented by the dotted line are not used in this design.

8.4 Summary

If this G function circuit adapts to more than 20 input variables, th e controller design isn't changed. Only the memories, registers and counters wo uld be added more bits to adapt to more contents. The whole design algorithm is still the same.

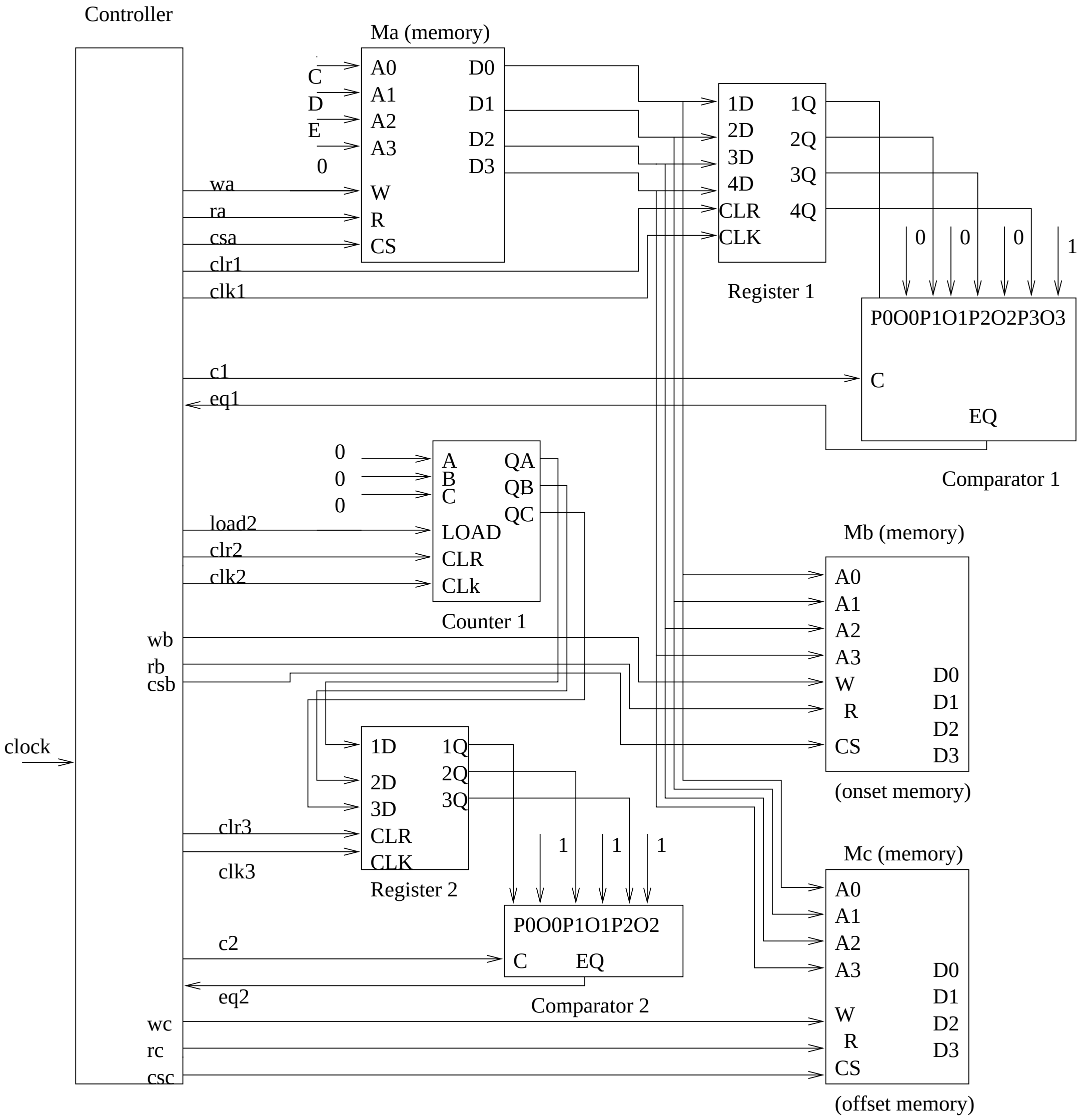

Figure 8.4: the Architecutre of G1 Circuit

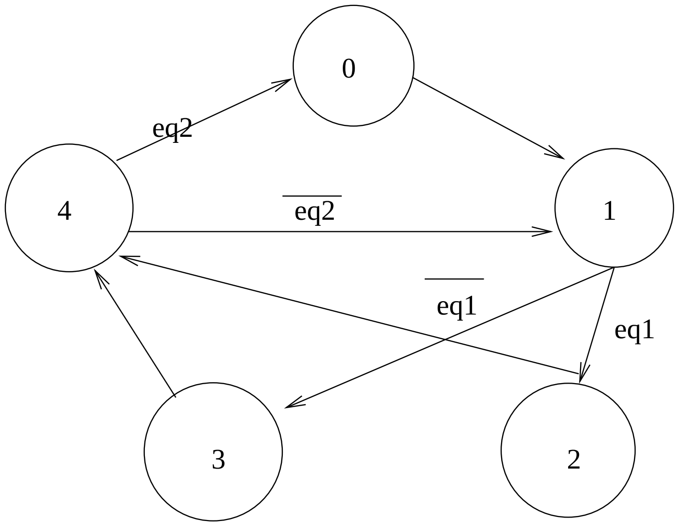

Figure 8.5: the State Diagram of G1 Function Circuit

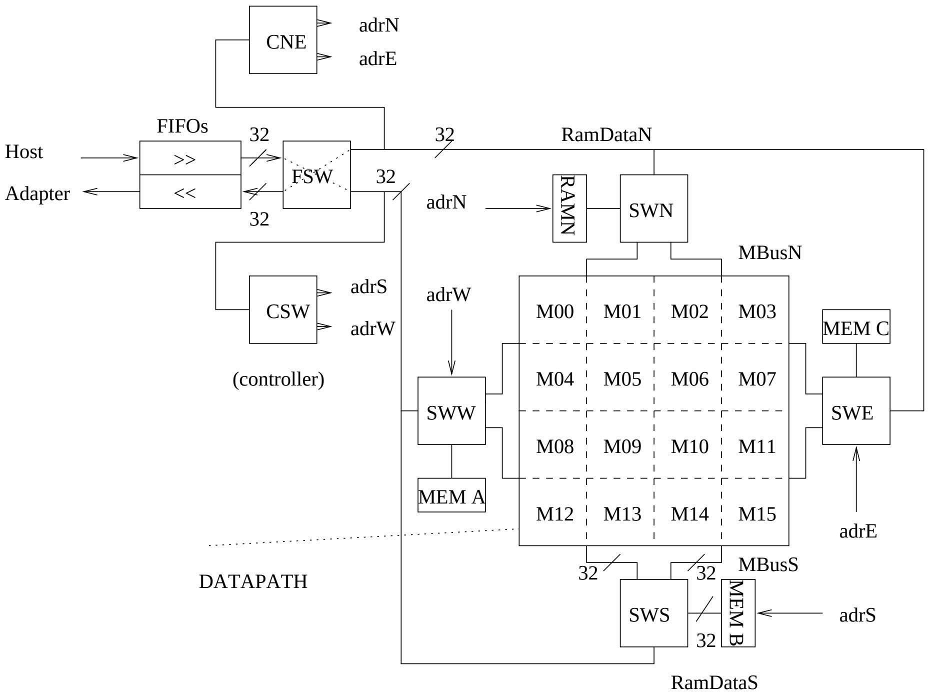

Figure 8.6: the Outline of Mapping

CHAPTER 9

Optimal Design H Function Block

H function block is constituted by the free set and G function block.

Figure 6.1 is still used to be an example to show how to design H function circuit.

This karnaugh map can be changed as the following structure with the compatible columns and maximum cliques techniques.

H function circuit is just to find free set(A, B) and G1, G2's combination in this structure whose minterm is equal to "1" (on set), "0" (off set) and "-" (dontcare).

9.1 Circuit Algorithm Description

Figure 9.1 can be reshaped as figure 9.2 according to the compatible column definition.

The data's order is made by these rules: G1G2=00, AB=00, 01, 10, 11

G1G2=01, AB=00, 01, 10, 11

G1G2=10, AB=00, 01, 10, 11

G1G2 C D E / A B	00 000	01 001	11 011	10 010	11 110	01 111	01 101	00 100
00	-	-	1	0	-	-	-	0
01	1	0	0	1	0	0	0	-
11	0	1	-	1	0	1	1	-
10	-	-	1	0	1	0	-	1
	C1	C2	C3	C4	C5	C6	C7	C8

Figure 9.1: Free Set and G1, G2 Combination Structure

G1G2=11, AB=00, 01, 10, 11

The number of minterm values is the same as the number of compatible columns. For example, when G1G2 is equal to 00, there are two compatible columns C1 and C8. So there are two minterm's value of all of G1G2AB's combinations. ABG1G2 is 0000, its minterm values are "-" and "0"; ABG1G2 is 0100, its minterm values are "1" and "0"; The data member structure of memory a is derived from figure 8.1. The variables (C, D, E) combination of bound set is treated as address. Their corresponding G1 bit's value is the value of memory a. If G1 bit's value is 1, this memory a's value is "0001". If G1 bit's value is 0, this memory a's value is "0000".

Each time, memory a's data is taken out to be compared with value "0001". If they are equal, put the memory a's address, which is C, D, E combination of bound set to memory b's data. If they are not equal, put the memory a's address

A	B	G1	G2	Minterm
0	0	0	0	-, 0
0	1	0	0	1, -
1	0	0	0	-, 1
1	1	0	0	0, -
0	0	0	1	-, -, -
0	1	0	1	0, 0, 0
1	0	0	1	-, 0, -
1	1	0	1	1, 1, 1
0	0	1	0	0
0	1	1	0	1
1	0	1	0	0
1	1	1	0	1
0	0	1	1	1, -
0	1	1	1	0, 0
1	0	1	1	1, 1
1	1	1	1	-, 0

Figure 9.2: H Function Data Member

to memory c. Other next data in memory a are taken out to be compared with value "0001".

Because there are 3 variables in bound set, these comparisons should be held 2^3 times. In the end, memory b stores all of variables combination of bound set whose G1's value is equal to 1, and memory c stores all of variables combinations of bound set whose G1's value is equal to 0.

Figure 9.3 is a flow chart of G1's circuit algorithm.

1) put the $2^3 - 1$ into register 1 and 0001 into register2.

2) initial counter 1 value as 0.

3) read one memory content from memory a.

4) compare this memory content with value "0001". If they are equal, write this memory content's address in memory a into memory b. If they are not equal, write this memory content's addresss in memory c. The counter 1's value automatically adds 1.

5) repeat step 3 and step 4, until counter 1's value is equal to the register 1.

9.2 Circuit Design Description

This circuit uses components as following:

Three memories are memory a, memory b and memory c. Memory a stores all of bound set(C, D, E)'s combinations and their corresponding G1's value.

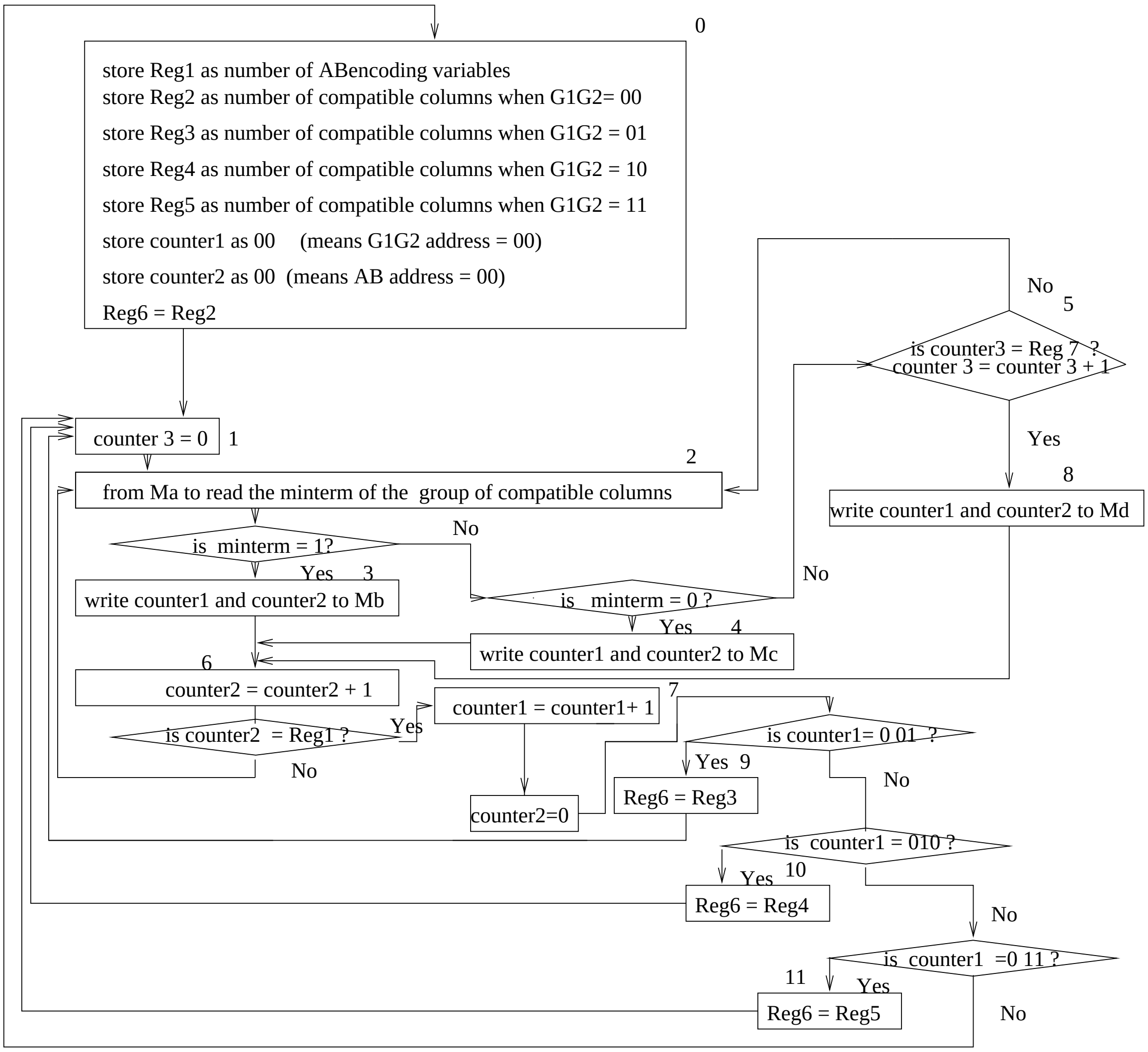

Figure 9.3: Flowchart of H Circuit Algorithm

Memory b stores all of bound set (C, D, E)'s combinations whose corresponding G1's value is equal to 1. Memory c stores all of bound set(C, D, E) combinations whose corresponding G1's value is equal to 0. The address bus of these memories is 4-bit, its msb is defined as "0"; The data bus is 4-bit, its first three bits (from left to right) are set up as "000", only the last bit is set up as G1's value, "1" or "0".

One counter is counter 1, it is used to count the number of the value variable combinations of the bound set which have been compared. Counter 1 is a 3-bit counter.

Two comparators are 8-bit comparator 1 and 6-bit comparator 2. Comparator 1 is used to compare the 4-bit data bus value from memory a with the value "0001". If they are equal, the comparator 1 outputs signal *eq1*, its value is set as "1". If they are not equal, the comparator 1 outputs signal *eq1* as "0". This signal *eq1* is transported to the control unit of the circuit. Comparator 2 is used to compare the counter 1's value with the whole number of the variable combination(C, D, E), If they are equal, comparator 2 outputs signal *eq2* as "1"; If they are not equal, it outputs signal *eq2* as" 0". This signal *eq2* is transported to control unit of the circuit. The control unit decides the state transfer according to these two signals.

Two registers are register 1 and register 2. Register 1 is a 4-bit register. It is a buffer for comparator 1. Register 2 is a 3-bit register. It is a buffer for comparator 2.

Figure 9.4 is a diagram of the circuit architecture. It includes the control unit and datapath unit. It has shown the control unit's output signals and datapath unit's output signal. The control unit's input signals are the datapath unit's output signals. The control unit's output signals are the datapath unit's input signals.

Each state's content has been shown as in figure 9.4. The control unit's state transfer diagram can be found as figure 9.5. This control unit of the circuit has 5 states.

9.3 Mapping H Function Circuit into DECpeRL-1 Board

The outline of mapping is shown in figure 9.6. As shown in the figure, for using Matrix as regular as possible, only datapaths are mapped in it; The controller is mapped in South-West controller(CSW) because it is easy to control input and output FIFOs and all matrix chips from switches. All other parts are shown in the figure. The components of DECpeRL-1 board that are represented by the dotted line are not used in this design.

9.4 Summary

If this H function circuit adapts to more than 20 input variables, th e controller design isn't changed. Only the memories, registers and counters wo uld be added

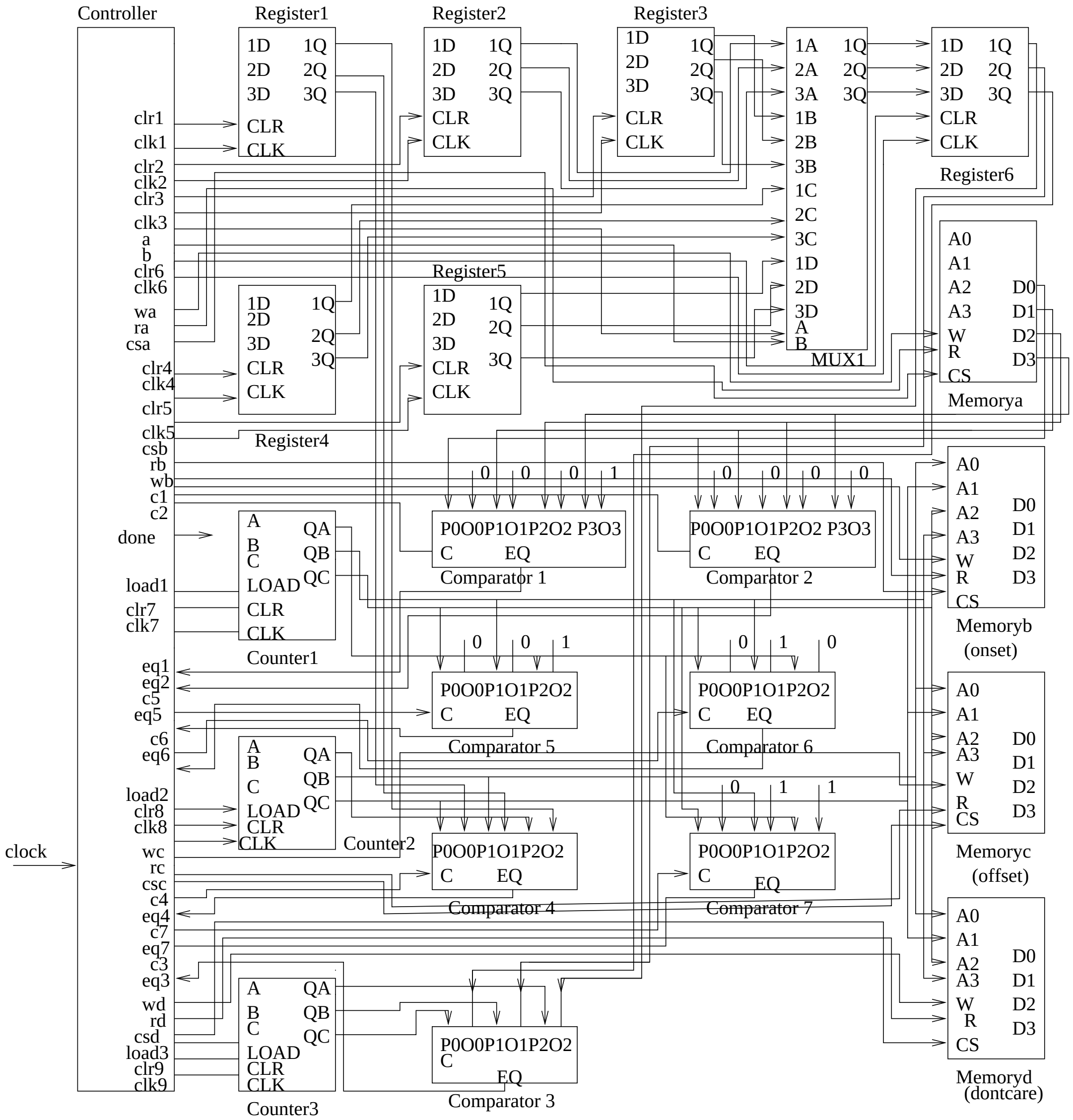

Figure 9.4: the Architecutre of H Function Circuit

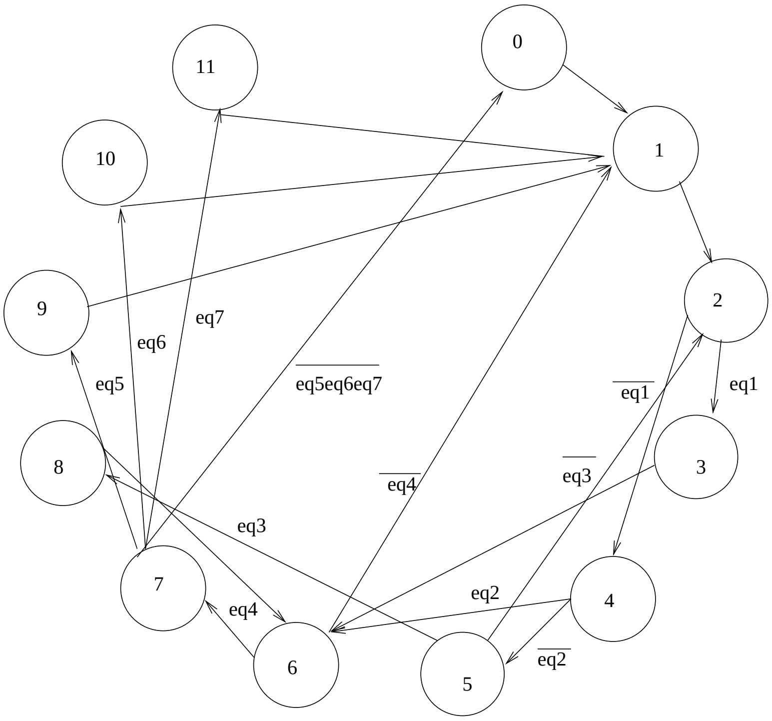

Figure 9.5: the State Diagram of H Function Circuit

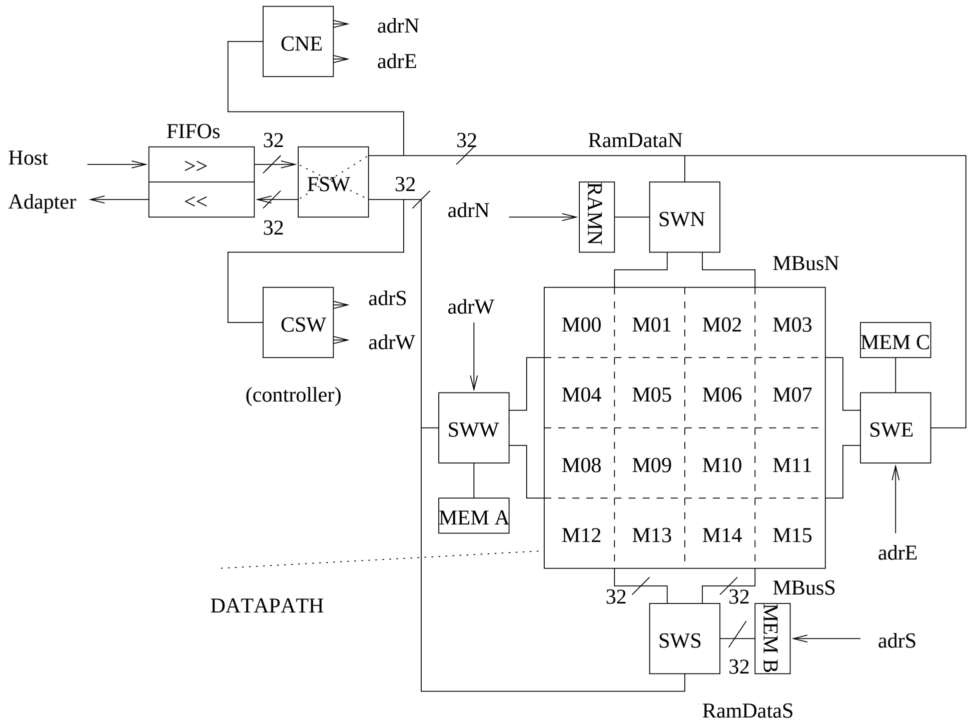

Figure 9.6: the Outline of Mapping

more bits to adapt to more contents. The whole design algorithm is still the same.

CHAPTER 10

Simulation

A complete design of the hardware accelerator for Function Decomposition is accomplished in this thesis. The design is captured in VHDL and is simulated by QuickVHDL, a digital simulator from Mentor Graphics. The functionality of this design is tested and approved to be correct.

10.1 Design Capture

The design of the hardware accelerator for Function Decomposition is captured in VHDL code hierarchically, which means that the VHDL codes of the lower level design blocks were captured first, then these blocks are tested and modified until their function is proven correct. This way, the design bugs can be identified earlier and can be fixed easier. Later, these blocks were used in the VHDL code of the upper level design blocks. Figure 10.1 shows the hierarchical structure of the hardware accelerator for Function Decomposition. The rectangular boxes in the figure represent design blocks, their names and corresponding VHDL file names(in parenthese) are shown in the boxes. The VHDl code of this

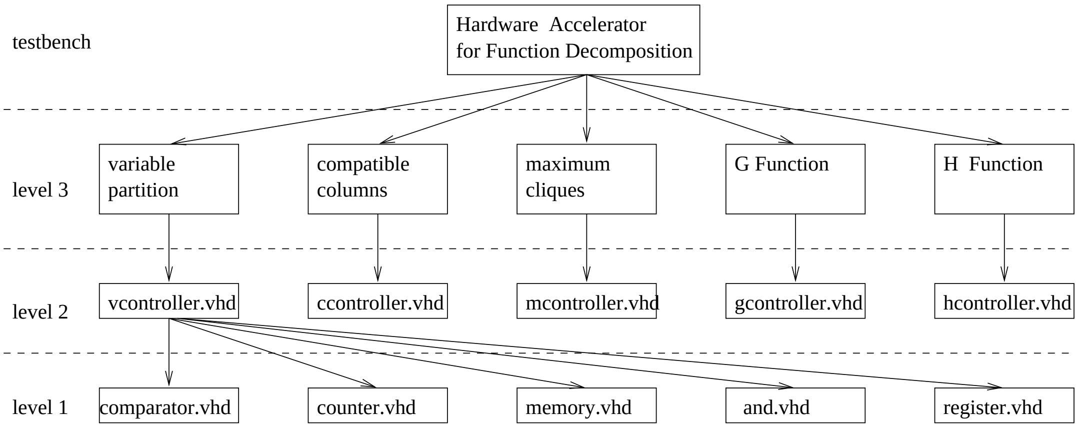

Figure 10.1: Hierarchical Structure of the Hardware Accelerator for Function Decomposition

design is available by contacting Dr Perkowski(mperkows@ee.pdx.edu).

10.2 Functional Verification

After the design of the hardware accelerator for Function Decomposition is captured, its functionality need to be fully tested to make sure that it does what it is supposed to. Functional verification on the Function Decomposition was performed through simulation by using QuickVHDL tool.

10.2.1 Simple Test Result for Function Decomposition

In order to prove the five blocks' design correct, figure 6.1 is not only used to be the design example, but also is used as test operation for the five blocks.

10.2.1.1 Test Result for Variable Partitioning

Test and address data input for memory a as shown in figure 6.1 //

address data

00000 -

00001 -

00010 0

00011 1

—– -

—– -

Test result for memory b

address data

00000 0111 (A1'- A0' =7)

00001 1111 (B1'- B0' =-1)

00010 1101 (C1'- C0' =-3)

00011 1111 (D1'- D0' =-1)

00100 0001 (E1'- E0' =1)

From this result, we can choose (b, c, d), (c, d, e) or (b, d, e) as bound set.Then calculate their column multiplicities, find out which group is the best bound set.

10.2.1.2 Test Result for Compatible Columns

The data input and data output are shown as figure 6.2.

The result is shown that

C1 is compatible with C8.

C2 is compatible with C3, C6 and C7.

C3 is compatible with C2, C5 and C7.

C4 is alone.

C5 is compatible with C3 and C8.

C6 is compatible with C2 and C7.

C7 is compatible with C2, C3, C6 and C8.

C8 is compatible with C1, C2, C5 and C7.

10.2.1.3 Test Result for Maximum Cliques

The data input and data output are shown as figure 7.3.

The result is shown that

C1 and C8 are maximum cliques.

C2, C6 and C7 are maximum cliques.

C3 and C5 are maximum cliques.

C4 is only himself clique.

Of course, there are other answers, But they are totally correct.

10.2.1.4 Test Result for G Function

The data input and data output are shown as figure 8.2.

The result is shown that ON(G1) = $\bar{C}$CE + $\bar{C}$D$\bar{E}$ OFF(G2) = $C\bar{D}E$ + $C\bar{}D$E + CDE + C$\bar{D}$E + C$D\bar{}E$

ON(G2) = $C\bar{}D$E + $\bar{C}$DE + CD$\bar{E}$ + CDE OFF(G2) = $C\bar{D}E$ + $\bar{C}$D$\bar{E}$ + C$D\bar{}E$

Furthermore, the above boolean equations of G1 and G2 can be simplified as follows: G1 = $\bar{C}$D + D$\bar{E}$ G2 = E + CD

10.2.1.5 Test Result for H Function

The data input is shown as figure 9.2

The data output is shown that ON(H) = $A\bar{}B$G1G2 + $\bar{A}$B$G1\bar{}G$2 + $\bar{A}$BG1$\bar{G}$2 + AB$\bar{G}$1G2 + ABG1$\bar{G}$2 + A$B\bar{G}1G$2 + A$\bar{B}$G1G2

OFF(H) = $A\bar{B}G$2 + $\bar{A}$BG2 + AB$G1\bar{}G$2 + ABG1G2 + A$B\bar{G}$1G2 + A$\bar{B}$G1$\bar{G}$2

DONTCARE(E) = $A\bar{B}G$1G2

CHAPTER 11

MY CONTRIBUTIONS TO THE FUNCTIONAL DECOMPOSITION GROUP, CONCLUSIONS AND FUTURE WORK

Since the Functional Decomposition Group was formed at Portland State University, we have spent a lot of time researching the different aspects of Functional Decomposition. Everybody in the group was involved in a different aspect of Functional Decomposition. My part was designing a hardware accelerator for Function Decomposition.

In order to finish this design task properly, it was necessary to do it very systematically in a number of steps. As the first step I did more research in Function Decomposition. I have found many algorithms for the five steps of Function Decomposition. Then I compared these various algorithms from each others to try to find the better one for me to realize it in hardware.

The second step I programmed the whole circuit using VHDL codes. Using Mentor Graphics QuickVHDL tool to simulate it. I should characterize some parameter's values in controller models to get a good result.

The third step I wrote down all of what I did to let the other students to develop Function Decomposition more advancely in the future.

Although the algorithm of variable partitioning is heuristic, it has shown that this algorithm is correct in most of time, specially in machine learning.

In the future, other students should design more bits to realize Function Decomposition by the base of my architecture. Let this architecture be realized in DECpeRLe-1 board. I believe this machine is once produced, it will become a very powerful tool in industry and research.

BIBLIOGRAPHY

[1] W.Wan, M.A. Perkowski, "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, pp. 230-235, September, 1992.

[2] Martin Bolton, "Digital Systems Design with Programmable Logic", ISBN 0-201-14545-6 pp.108

[3] R.L. Ashenhurst, "The Decomposition of Switching functions," *Proc. Int'l Symp.Theory of Switching Functions,* pp.74-116,1959.

[4] H.A. Curtis, "A New Approach to the Design of Switching Ciruits," *Princeton, N.J., Van Nostrand*,1962.

[5] Rahul Malvi, "The Column Multiplicity Problem in Decomposition of Functions and Relations" pp.2

[6] Marek A. Perkowski, Stainslaw Grygiel, and the Functional Decomposition Group, "A Survey of Literacture on Function Decomposition" version iv. Nov 20, 1995.

[7]

[8] J.Vuillemin, P.Bertin, D.Roncin, M.Shand, H.Touati, P.Boucard, "Programmable Active Memories: Reconfigurable Systems Come of Age" IEEE Transactions on VLSI Systems, VOL XX. NO Y, Month 1995

[9] Naveed Sherwani, "Algorithms for VLSI Physical Design Automation"