

STANDARD CELL IMPLEMENTATION OF A MICRO-CONTROL
UNIT FOR A PROLOG UNIFICATION COPROCESSOR

A Thesis

by

HABIBOLLAH GOLNABI

Submitted to the Graduate College of
Texas A&M University
in partial fulfillment of the requirement for the degree of

MASTER OF SCIENCE


May 1988


Major Subject: Electrical Engineering

STANDARD CELL IMPLEMENTATION OF A MICRO-CONTROL UNIT
FOR A PROLOG UNIFICATION COPROCESSOR

A Thesis
by
HABIBOLLAH GOLNABI

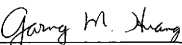
Approved as to style and content by:


Karan Watson
(Chairman of Committee)


Philip Noe
(Member)


Dan Colunga
(Member)


Nasser Kehtarnavaz
(Member)


J. W. Howze
(Head of Department)

May 1988

ABSTRACT

Standard Cell Implementation of the Micro-Control Unit

For A Prolog Unification Coprocessor. (May 1988)

Habibollah Golnabi, B.S., Texas A&M University

Chairman of Advisory Committee: Dr. Karan Watson

This work contains the standard cell implementation of the micro-control unit for a prolog unification coprocessor using Mentor Graphics software package. Standard cell design tools have been interfaced with custom layout facilities to achieve the complete chip design. The performance of the micro-control unit has been compared to that of Parikh's functional simulation.

To My Parents

ACKNOWLEDGMENTS

I would like to thank my advisor, Dr. Karan Watson, for her helpful guidance and suggestions throughout this research. I would also like to thank my committee members: Dr. Philip Noe, Dr. Nasser Kehtarnavaz and Dan Colunga for thier helpful contributions.

TABLE OF CONTENTS

CHAPTER		Page
I	INTRODUCTION	1
	A. Objective	2
	B. Previous Research	3
II	FUNCTIONAL DESCRIPTION OF THE UNIFIC	6
	A. Data Formats	6
	B. Block Diagram of the UNIFIC Execution Unit	6
	Registers	7
	Stack	9
	Stack Registers and Stack Pointer	9
	Binding Memory	9
	Arithmetic Logic Unit (ALU)	10
	C. Micro-Control Unit	10
	Microsequencer	12
	Array	12
	Combinational Logic	12
	Instruction Decode	14
	Control Store	14
	Micro-Instruction Register	14
III	DESIGN HIERARCHY OF THE MICRO-CONTROL UNIT	18
	A. Schematic Capture	18
	NETwork and SYMbol EDitor	18
	MOSIS_EXPAND_COMP	18
	MOSIS_DESIGN_CHECKER	20
	MOSIS_EXPAND_DESIGN	20

TABLE OF CONTENTS (Continued)

CHAPTER	Page
	MOSIS_ADD_DELAY 20
B. Circuit Simulation	20
	QUICKSIM 21
C. Physical Layout	21
	LOGIC_ENTRY 21
	CELLFLOOR 21
	CELLPLACE 23
	CELLPOWER 23
	CELLROUTE 23
	CELLSQUEEZE 23
	MINROUTE 23
	CELLVERIFY 23
	PREGRAPH 24
	CELLGRAPH 24
D. Chipgraph	24
	MCIF_READ and MCIF_WRITE 24
	BUILD_LIB 26
	TDF_CHIP_INPUT 26
IV OPERATION OF THE MICRO-CONTROL UNIT	27
	A. Phase I 27
	B. Phase II 29
	C. Phase III 30
V CIRCUIT DESCRIPTION OF THE MICRO-CONTROL UNIT	31
	A. Two Phase Clock 31
	B. Combinational Logic 31
	TAGDECODE 34
	SETCAMPBIT 34
	CAMSRCH 34
	DECODE_SR_SPARITY 34
	DEC.BOUND.VV 38
	DEC.BOUND.VC 38

TABLE OF CONTENTS (Continued)

CHAPTER	Page
C. Instruction Decode	38
D. Microsequencer	50
Micro-Stack Pointer	50
Micro-Stack	50
Micro-Program Counter	52
Register DATA	57
nROM	57
Register REG	62
Multiplexer	62
E. Array	62
F. Control Store	62
G. Micro-Instruction Register	65
VI LAYOUT DESCRIPTION OF THE MICRO-CONTROL UNIT	67
A. Cell Station Process Flow	67
Inputs	67
Processing Modules	68
Output	69
B. Chip Station Process Flow	69
Array	71
Micro-Stack	71
nROM	71
PLA	75
VII SIMULATION RESULTS AND CONCLUSIONS	77
A. Simulation Results	77
B. Conclusions	79
REFERENCES	80
APPENDIX A	82

TABLE OF CONTENTS (Continued)

CHAPTER	Page
APPENDIX B	94
APPENDIX C	108
VITA	116

LIST OF TABLES

Table	Page
I. The Simulation Results for Two Expressions with Increasing number of Terms	78

LIST OF FIGURES

Figure	Page
1. Block Diagram of the UNIFIC Execution Unit	8
2. Top Level Schematic of the Micro-Control Unit	11
3. Microsequencer	13
4. Combinational Logic	15
5. Instruction Decode	16
6. The IDEA Station Process Flow	19
7. The CELL Station Process Flow	22
8. The CHIP Station Process Flow	25
9. Clocking Scheme of the Micro-Control Unit	28
10. Root Symbol of the Micro-Control Unit	32
11. Two Phase Clock	33
12. TAGDECODE Routine	35
13. SETCAMBIT Routine	36
14. CAMSRCH Routine	37
15. DECODE.SR.SP.ARITY Routine	39
16. DEC.BOUND.VV Routine	40
17. DEC.BOUND.VC Routine	41
18. DEC_1	42
19. DEC_2	43
20. DEC_3	44
21. DEC_4	45
22. DEC_5	46
23. DEC_6	47

LIST OF FIGURES (Continued)

Figure	Page
24. DEC_7	48
25. DEC_8	49
26. Micro-Stack Pointer	51
27. Layout Spacing of the Micro-Stack	53
28. Micro-Stack	54
29. Memory Cell of the Micro-Stack	55
30. Data Buffer of the Micro-Stack	56
31. mPCI	58
32. mPCO	59
33. Register DATA	60
34. nROM	61
35. Register REG	63
36. Multiplexer	64
37. Micro-Instruction Register	66
38. Standard Cell Design of the Micro-Control Unit	70
39. The Array Layout	72
40. The Micro-Stack Layout	73
41. The nROM Layout	74
42. The PLA Layout	76

CHAPTER I

INTRODUCTION

The increasing importance of Artificial Intelligence (AI) has led to an extensive research with the logic-programming language Prolog [5]. Many expert systems being developed are implemented in Prolog because of its automatic backtracking control and internal data-base management facilities.

The Japanese Fifth Generation Computer Systems (FGCS) project has contributed significantly to the development of Prolog systems, in particular, to a Prolog unification hardware unit. In the future, high performance Prolog machines are going to play an important role in real time applications such as knowledge-based systems, natural-language processing and robotics. Prolog provides an appropriate base for implementation of a powerful hardware system using highly parallel architectures and VLSI technology [18]. Since the execution mechanism of Prolog is very much different from other conventional languages, there is a need to further investigate its potentials.

At present, Prolog is run on conventional "Von Neumann" machines, which slow down the execution rate [7]. One major source for the slow execution is the unify function. This function spends 55-70% of the total query processing time [19]. High performance Prolog machines are being developed in order to reduce the processing time of the unify function. The execution time can be reduced in two ways, (1) by reducing the total number of calls to the unify function, and (2) by designing a unification coprocessor replacing the unify function. This unification coprocessor can be employed in conjunction with a host processor on any computer system.

Journal model is *IEEE Transactions on Computers*.

Unification is one of the vital operations in Prolog systems, and is based on the Resolution Principle [11]. The implementation of the unification has a great effect on the organization and performance of Prolog systems. In formal terms, Unification is a process that finds substitutions of terms for variables to make expressions identical. The process of unification is analogous to that of finding a common denominator for fractions [20].

A. Objective

The objective of this work is to accomplish the following two tasks:

- (1) To design and implement the micro-control unit proposed by P. Parikh [9] using standard cell layout. The major reason for choosing P. Parikh's architecture [9] is that his coprocessor improves the execution speed for the unify function by 12%-16% over the coprocessor designed by R. Gollakota [3].
- (2) To interface custom layout facilities with standard cell design tools on the Apollo workstation to achieve the complete chip design. The Mentor Graphics software package includes a subpackage called the IDEA station. This station contains a component cell library, the Schematic Capture tool, and QUICKSIM (a simulation tool). The Mentor Graphics CELL station is used to describe the physical layout of the standard cells. Chip elements not built from standard cells, such as memory devices, are implemented with the Mentor Graphics CHIP station using CHIPGRAPH (a layout editor). These customized components in this project are then incorporated into the CELL station by developing a set of software tools. The CELL_MODEL file and CELL_LIST file have been developed to convert cells in the CHIPGRAPH data base into the cells in CELL station data base. Finally, a Technology File (TF) and a Process Definition File

(PDF) have been developed to convert Caltech Intermediate Format to Mentor Caltech Intermediate Format (MCIF-Chipgraph format) or vice versa. This is needed due to the lack of the CHIPGRAPH's design rule checker. Therefore, elements developed in CHIPGRAPH can ultimately be sent to MAGIC, a layout editor which has a design rule checker.

B. Previous Research

The first hardware unification unit was developed by S. Lien. He designed a simple unification chip called UNIF to implement Robinson's original unification algorithm [11]. This chip was never implemented.

J. Oldfield [7] and a team of researchers at Syracuse University are attempting to develop a unification coprocessor called Syracuse Unification Machine (SUM) based on the work done by S. Lien. The hardware unit consists of the LAMBDA Lisp machine which represents expressions by the combination of a tag and a pointer. SUM identifies the type of an expression from a tag field and proceeds by binding agents using Content Addressable Memory (CAM) for binding variable-to-expressions.

N.S. Woo [19] has developed a microprogrammed hardware unification unit at AT&T Bell Laboratories. Woo has demonstrated that his coprocessor in conjunction with a host processor significantly reduces the unification execution time.

A systolic-like architecture has been suggested to implement unification by Shobatake and Aiso [13]. Symbols are used instead of pointers by having a line of symbols and the arity of each symbol to represent the structure of terms on uniformly structured hardware cells. A broadcast bus is used in this design to search variables in a parallel fashion. The systolic algorithm is responsible for easy

execution of copying structures during the unification process.

The Parallel Inference Engine (PIE) is part of the Japanese Fifth Generation Computer Systems (FGCS) project. This system is being developed by Moto-Oka et.al. at the University of Tokyo [4]. It uses the goal rewriting model based on OR-parallelism [1] in which goals are independent of each other and stored in a goal pool. Each unify processor in this system fetches a goal, unifies it with definition clauses, and generates new goals. The PIE has a highly modular architecture.

A Sequential Inference Machine (SIM) has been developed as a part of the Japanese Fifth Generation computer systems (FGCS) project [18]. The SIM consists of two main parts: a sequential inference machine and a high speed Prolog machine. The unification unit is an integral part of this machine. CPU is designed on a pipelined local parallelism basis and a Current Mode Logic (CML) circuit is used to speed up the processor. Currently, this project has not been completed.

A Sequential Prolog Machine (PEK) is being developed by Tamura et.al. at Kobe University, Japan [15]. This machine is designed mainly for unification and backtracking. The PEK theoretically achieves a very high performance level of more than 100K logical inferences per second (LIPS).

A hardware unification unit called UNIFIC is designed and simulated by Ram Gollakota [3] under the supervision of Dr. Karan Watson at Texas A&M University. This architecture uses one internal data bus and makes use of Content Addressable Memory (CAM) to improve coprocessor speed. This design has not been implemented.

A second Prolog unification coprocessor designed by P. Parikh [9] under the supervision of Dr. Karan Watson at Texas A&M University. The architecture is designed at functional level and simulated using a hardware description language

(ISPS). Two data buses are used to expedite the internal data transfer and hence, to reduce the execution time of the unification coprocessor. In fact, it was shown that this coprocessor increases the execution speed by about 12%-16% over the coprocessor designed by R. Gollakota [3]. The controller portion of the UNIFIC is designed at system level but has not been implemented.

CHAPTER II

FUNCTIONAL DESCRIPTION OF THE UNIFIC

The architecture of the UNIFIC was designed by P. Parikh [9] at the functional level. He has shown that his coprocessor significantly reduces the execution time of the "unify" function of the prolog interpreter. For this reason, his architecture has been selected to be implemented by standard cell VLSI design.

A. Data Formats of the UNIFIC

Prolog uses four different types of data: constant, variable, function and list. The UNIFIC uses 32 bit data format. The MSB is bit 31 and the LSB is represented by bit 0. The data described by P. Parikh [9] is divided further into different fields, DTAG, CID, BO, ARITY.

B. Block Diagram of the UNIFIC Execution Unit

The UNIFIC has two internal data buses, input/output registers, data and memory address registers, stack registers and stack pointer, a LIFO stack, micro-control unit, a Content Addressable Memory (CAM) with an attached Random Access Memory (RAM) and an Arithmetic Logic Unit (ALU). The block diagram of the UNIFIC execution unit is shown in Fig. 1. The UNIFIC uses two 32 bit wide internal data buses, A and B, which are used for transfer of data through the unification coprocessor.

Registers

DIN and DOUT

They are 32 bit data registers used for communicating with the outside world. Data input register (DIN) and data output register (DOUT) are connected to the external data bus and internally connected to the data buses A and B, respectively. Data read from the external memory comes to DIN and data to be written to the external memory goes to DOUT.

AO

AO is a 14 bit address register connected to the external data bus. Internally AO is connected to both the buses A and B. Any transfer to the AO register signals a data transfer to the on-chip bus controller for the external bus. This bus controller postpones the execution of the next state of the micro-instruction until the external transfer is complete.

DR1 and DR2

These are each 32 bit wide registers connected to both buses A and B. The micro-control unit has the ability to selectively decode any of the bits in the registers.

DTEMP1 and DTEMP2

The temporary data registers DTEMP1 and DTEMP2 are each 32 bits wide. DTEMP1 and DTEMP2 are connected to both the internal buses A and B. These registers store the data coming into DIN only.

MAR1 and MAR2

These two are memory address registers. They are each 14 bits wide, enabling them to access 16k memory. They are connected to both the internal buses A and B. MAR1 can only read from the bus B. MAR1 has the address of terms in Expression 1 and MAR2 has the address of terms in Expression 2.

ARITY

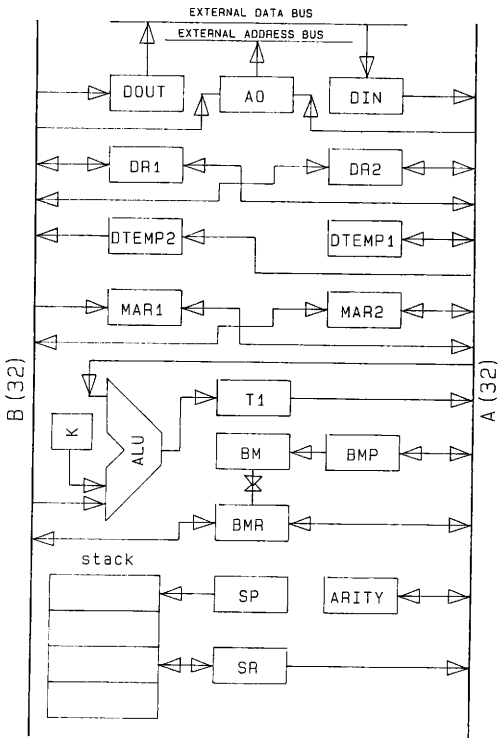


Fig. 1 Block Diagram of the UNIFIC Execution Unit

Arity is a 4 bit register connected to the internal bus A. The first four bits of the pseudo-instruction contains the number of terms in the expressions. These bits are loaded into the ARITY register from DTEMP1 in the initial phase of processing. The arity is decremented after the completion of unification of each term in the expressions. When dealing with functions, the arity of the expression is stored in the stack, and the arity of a function is stored in the ARITY register.

T1

This is a 4 bit buffer connected to the ALU and the internal bus A. The ALU performs a 32 bit logical comparison and the result is stored in T1. If the result of the comparison is false, then the process of unification has failed at one of its steps.

Stack

The stack is organized on a Last In First Out (LIFO) register stack. It is 14 bits wide and has a depth of 32 words. The contents of ARITY, MAR1 and MAR2 are pushed onto the stack whenever two functions are to be unified. The contents are pushed onto the stack only if ARITY is non-zero.

Stack Registers and Stack Pointer

The Stack Register (SR) is a 14 bit buffer connected to both the stack and the internal bus A. The Stack Pointer (SP) is 6 bits wide with a local incrementer/decrementer. It is connected directly to the stack.

Binding Memory

The Binding Memory (BM) is responsible for binding variables. It is 32 bits wide and has a Content Addressable Memory (CAM) and a Random Access Memory

(RAM). The upper 16 bits are part of the CAM and the lower 16 bits is the RAM. The Binding Memory Pointer (BMP) is a 4 bit wide register connected to the BM and the internal bus A. The Binding Memory Register (BMR) is a 32 bit wide register which acts as a buffer between the BM and the data buses A and B. The upper 16 bits of the BMR and the BM are always compared. If the result of comparison is true, then data can be exchanged between that particular BM location and the BMR. Also, if a match occurs, the BMP points to the particular location. This provides a good speed advantage for the UNIFIC in searching for the bindings of variables. The CAM part of the BM, is used to store the variable identifiers. The RAM part can be used to determine if the variable is bound. The uppermost two bits of the RAM indicate bound status, if the variable is bound, the remaining lower 14 bits of the RAM is the pointer to the term to which it is bound.

Arithmetic Logic Unit (ALU)

The ALU performs only a 32 bit logical comparison of two registers ALUA and ALUB. The result of the operation is fed back to the micro-control unit via buffer T1. If the result of the comparison is false, then the process of unification has failed at one of its steps. This terminates further processing in the UNIFIC and send a fail signal to the host processor.

C. Micro-Control Unit

The micro-control unit is the heart of the UNIFIC. It consists of the following parts: microsequencer, array, combinational logic, instruction decode, control store, and micro-instruction register. Fig. 2 depicts the functional diagram of the micro-control unit.

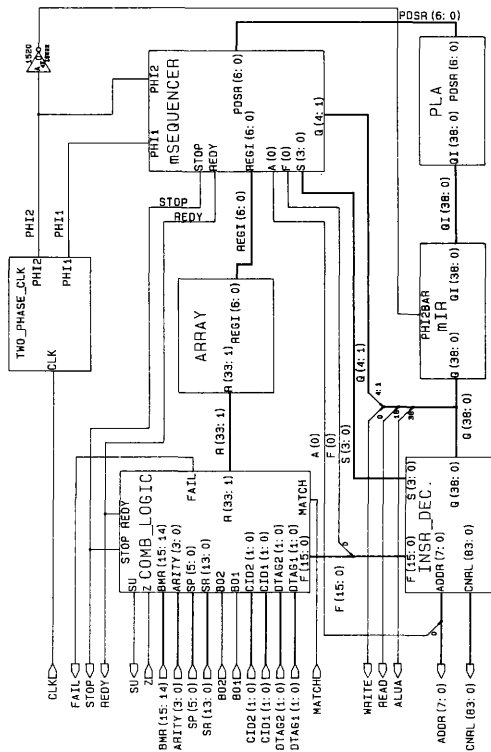


Fig. 2 Top Level Schematic of the Micro-Control Unit

Microsequencer

The microsequencer (mSEQUENCER) is responsible for providing the information about the next instruction to be executed during the next microcycle. It consists of a multiplexer (MUX), a micro-program counter (mPC), the micro-stack (mSTACK), the micro-stack pointer (mSP), the nROM, and three special purpose registers namely mPCO, DATA, and REG. The micro-program counter holds the next micro-instruction address and is incremented by one at each microcycle during sequential operation. It is loaded with parallel input data when branching occurs via "jump" or "call" in the micro-program. The micro-stack stores the address of the micro-instruction when "call" micro-operation is executed. This micro-instruction stored on the micro-stack is executed when "return" micro-operation is executed. The block diagram of the microsequencer is given in Fig. 3.

Array

The array is a mapping-table circuit which generates the starting address of the micro-subroutine or the branch address by decoding the inputs. The address generated by this way is stored in REG. There are two types of branch occurring in the micro-program. One is the conditional branch, which appears in REG, and the other is direct branch. In case of the direct branch, the next address is stored in DATA. Depending upon the STATE field in the micro-instruction, one of these four registers, viz. micro-program counter, micro-stack, REG and DATA, is selected for the next address of the micro-instruction.

Combinational Logic

This routine generates thirty three signals R(33:1) by receiving input from DR1, DR2, and other registers employed in the UNIFIC. These signals are decoded

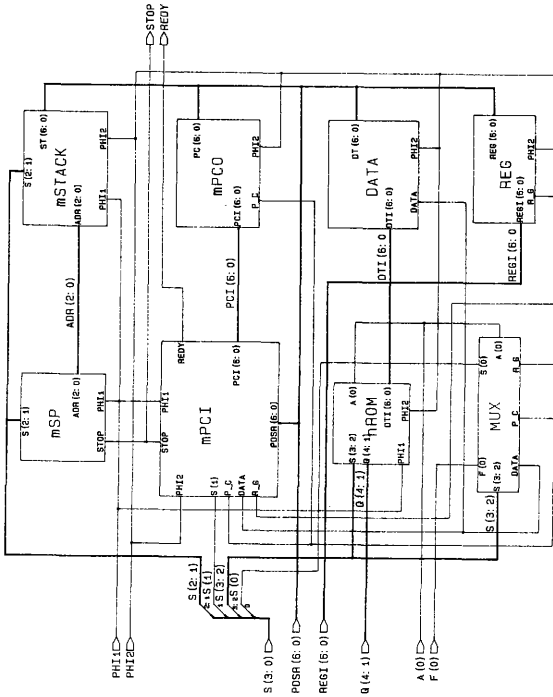


Fig. 3 Microsequencer

by the array at each microcycle. The information regarding all these signals are obtained by P. Parikh [9]. The block diagram of the subroutines describing the combinational logic (COMB_LOGIC) is given in Fig. 4.

Instruction Decode

Fig. 5, depicts the functional decoding scheme of the instruction decode (INSR_DECODE). It essentially decodes different fields from the micro-instruction register. The outputs from the instruction decode are the STATE field, the FLAG field, the ADDRESS field, and the control signals.

Control Store

The control store is a read-only memory that stores the micro-program. It is basically made up of a PLA with 39 bits wide and 70 words long. Each micro-instruction stored in the PLA is divided into 14 fields. Fields 1 through 9 occupy upper 27 bits and control normal micro-operations such as data transfer and arithmetic operation. Field 10 controls the decoding signals and determines the next address of the micro-instruction. Fields 11 and 14 control writing to and reading from the memory respectively. Field 12 determines the next micro-instruction to be executed. It determines whether sequential operation, conditional branch, call to a micro-subroutine, return from a micro-subroutine or a direct jump is to be performed in the micro-program. If a direct jump is required, the branching address in the coded form is stored in the micro-instruction itself. The field holding the coded address is decoded and nROM determines the next address to jump to. All these fields are decoded in parallel.

Micro-Instruction Register

The micro-instruction register (mIR) holds the current micro-instruction being

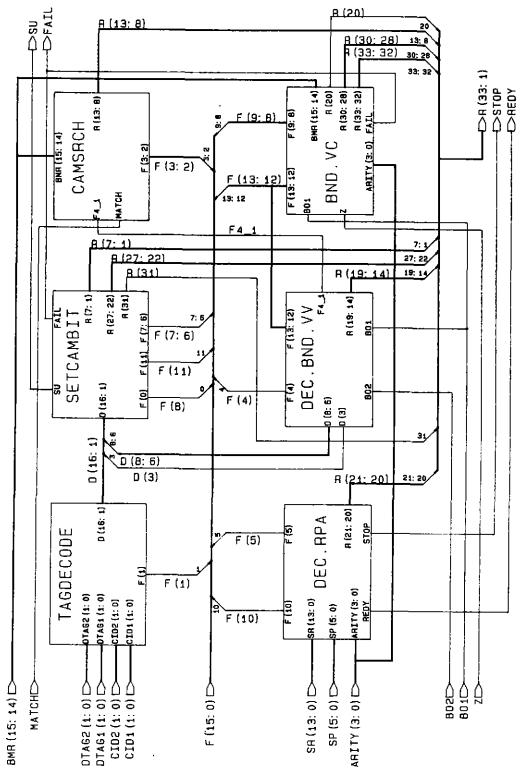


Fig. 4 Combinational Logic

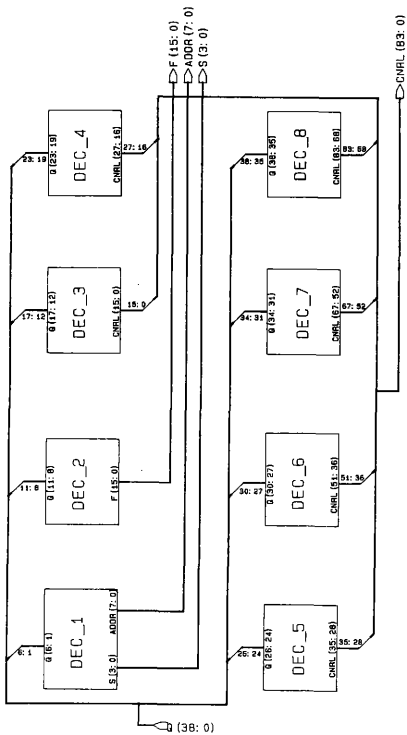


Fig. 5 Instruction Decode

executed. The different fields in the micro-instruction register are decoded simultaneously and the control signals for the micro-operation are generated.

CHAPTER III

DESIGN HIERARCHY OF THE MICRO-CONTROL UNIT

This section presents the standard cell design process on the Mentor Graphics workstation. The design of a standard cell consists of three major steps: schematic capture, circuit simulation, and physical layout.

A. Schematic Capture

During schematic capture, the logic of the micro-control unit is defined. The symbols in the logic components library are used to implement the micro-control unit schematic. Logic symbols built on the Mentor Graphics IDEA station contain the information necessary to drive the simulation and physical layout tools. The process flow of the IDEA station is shown in Fig. 6.

NETwork and SYMbol EDitors

NETwork EDitor (NETED) and SYMbol EDitor (SYMED) are employed to capture the circuit schematic of the micro-control unit. The symbols created for different components in SYMED are connected together in NETED to form the logic circuit.

MOSIS_EXPAND_COMP

Due to the hierarchical design capabilities of SYMED and NETED, schematics of the micro-control unit must be flattened (expanded) before netlisting. MOSIS_EXPAND_COMP generates an expanded design file (COMP.EREL) at the component level (not at the gate level).

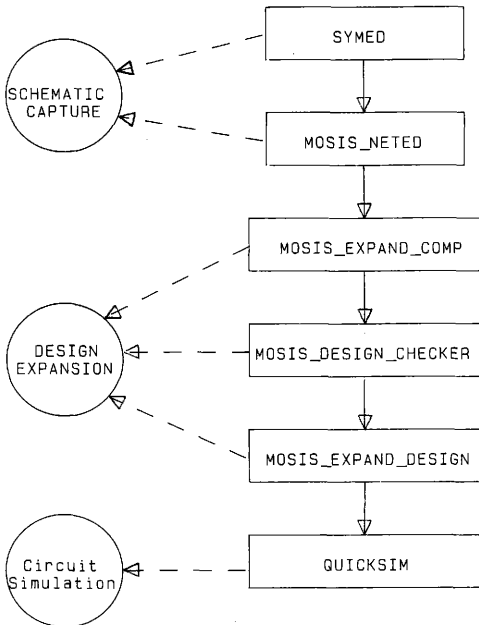


Fig. 6 The IDEA Station Process Flow

MOSIS_DESIGN_CHECKER

This software does a design rule check of the expanded schematic of the micro-control unit based on a set of technology specific rules. These design rules consist of fanout checking, net current (fan in) checking, external net checking, and unused pins checking. The design checker is run after expanding the micro-control unit with MOSIS_EXPAND_COMP.

MOSIS_EXPAND_DESIGN

This software expands the schematic of the micro-control unit at gate level. It also assures that all properties needed for simulation (QUICKSIM) are flattened and included in the database file (DESIGN.EREL).

MOSIS_ADD_DELAY

This software modifies the circuit's delay characteristics based on a number of possible inputs, the most significant of which is actual wire length delays. The performance of a circuit varies considerably due to many factors including physical layout, processing parameters, operating voltage, operating temperature, and die size. It is the function of ADD_DELAY to help evaluate the design's performance based on known relationships between these characteristics and the timing behavior of the technology. After ADD_DELAY is run, the micro-control unit is resimulated using QUICKSIM software to determine the effects.

B. Circuit Simulation

Simulation uses the connectivity data from the schematic and the timing information from the logic library to model the circuit's logical function. It analyzes

the internal gate delays of a circuit and uses the information to predict the circuit's behavior.

QUICKSIM

This software provided with the Mentor Graphics IDEA station is utilized to simulate the micro-control unit. It invokes DESIGN.EREL file and does a logic and timing analysis of the design before any layout is performed.

C. Physical Layout

In physical layout, connectivity data is used to implement a chip's design using the technology design rules to place defined patterns of connectivity (known as physical macros). Then net routing connects the macros of the design. The Mentor Graphics CELL Station, a system of software tools for standard cell chip design, is used to do the layout of the micro-control unit. The placement and routing can be performed either automatically or through interactive graphics. The process flow of the CELL station is given in Fig. 7.

LOGIC.ENTRY

This command extracts the connectivity information from the design file (DESIGN.EREL) generated by the command MOSIS_EXPAND_COMP and creates the physical design file (DESIGN.PRM) used by the other CELL station commands.

CELLFLOOR

Based on the netlist file (DESIGN.PRM) which contains cell library and floorplan parameters, CELLFLOOR automatically generates a minimum area chip floorplan that has enough sites for all the cell instances in the netlist. It provides

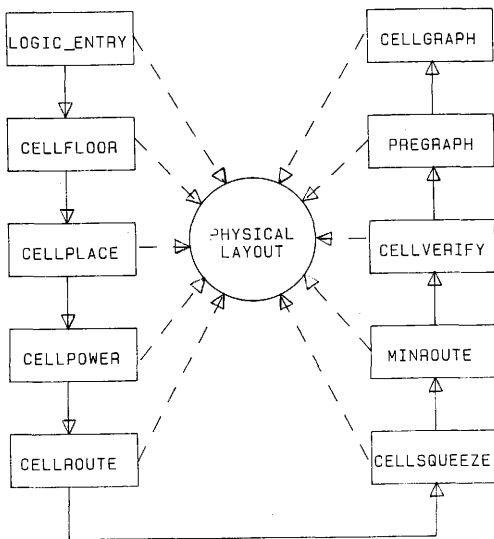


Fig. 7 The CELL Station Process Flow

enough feedthrough tracks and also satisfies constraints imposed by floorplan parameters.

CELLPLACE

Based on the floorplan, CELLPLACE automatically performs a global placement followed by a detailed placement of the cells.

CELLPOWER

The power nets are routed before the signal nets because the former have more stringent layout constraint. CELLPOWER automatically routes the power nets.

CELLROUTE

CELLROUTE performs global routing, which is followed by detailed routing of the signal nets.

CELLSQUEEZE

This command identifies and removes unused horizontal tracks in the channels to minimize the chip area.

MINROUTE

It performs post-routing processing tasks such as minimizing the use of poly in the routes.

CELLVERIFY

CELLVERIFY serves as a final check on the validity of the layout produced by the CELL Station. It also checks design rule violations and the electrical connectivity.

PREGRAPH

It generates a "working file" for editing purposes. This command is needed when running an automatic program.

CELLGRAPH

CELLGRAPH is CELL station's interactive graphics editor. It allows manual editing of the placement and wiring of a physical design. It can be used at any point in the CELL Station processing sequence, after CELLFLOOR, to examine or alter the physical design file data. The output of CELLGRAPH is a final chip layout file. The chip layout file is then converted to the GDSII stream format by GDSII.OUTPUT software. MOSIS uses this file to fabricate the micro-control unit.

D. Chipgraph

This is a graphic editor that supports the physical layout of integrated circuits. CHIPGRAPH is a part of CHIP station, one of the Mentor Graphics workstation. The micro-control unit memory cells are implemented with CHIPGRAPH. The cells are then incorporated into the CELL station database by set of software tools explained below. The process flow of the CHIP station is depicted in Fig. 8.

MCIF READ and MCIF WRITE

MCIF READ converts Caltech Intermediate Format (CIF) file into CHIPGRAPH database (MCIF). This software is needed since CHIPGRAPH has no design rule checker. MCIF READ requires a Process Definition File (PDF) and a standard CIF file. These two technology files have been generated as will be discussed in Chapter VI.

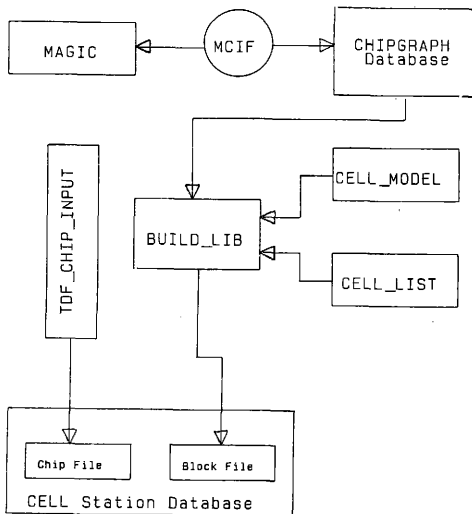


Fig. 8 The CHIP Station Process Flow

MCIF WRITE converts a CHIPGRAPH database to CIF file with the same technology file requirements as MCIF READ software.

BUILD_LIB

This software reads the CHIPGRAPH database and extracts the cell information for the CELL station database. BUILD_LIB reads CELL_LIST file to determine which cells to convert and interprets the data in each cell using a CELL_MODEL file. Both of these files have been written for the micro-control unit memory cells which will be discussed in Chapter VI. BUILD_LIB generates the block file for the CELL station database as shown in Fig. 2.

TDF_CHIP_INPUT

Before running BUILD_LIB to create the block file section of the CELL station database, TDF_CHIP_INPUT program is run to read the ASCII chip file and create a binary file. This file contains the design rule data for the CELL station database. Having obtained both chip file and block file of memory cells, LOGIC.ENTRY is used to incorporate these cells into the standard cell design of the micro-control unit.

CHAPTER IV

OPERATION OF THE MICRO-CONTROL UNIT

In this chapter the operation of the micro-control unit is described. The micro-control unit of the UNIFIC is responsible for the proper functioning of the coprocessor. It retrieves micro-instructions from the control store and generates control signals based on these micro-instructions.

The UNIFIC uses a two-phase clocking scheme [9]. However, the micro-control unit uses a three-phase scheme, Φ_1 , Φ_2 and $\Phi_2\bar{}$ as shown in Fig. 9. The sequence of events occurring during each phase is explained in the following sections.

A. Phase I

- 1) The incremented value of the micro-program counter is loaded into the micro-program counter. It now holds the address of the micro-instruction to be executed in sequence.
- 2) The control fields from the micro-instruction register are decoded and appropriate control signals are generated. Data transfer from registers to data buses takes place during this phase. Control signals to transfer data from data buses to registers are generated simultaneously, however, this data transfer takes place only during the second phase of clock.

The address field of the control word decides the next micro-word to be loaded into the micro-instruction register. The micro-instructions deciding this are 'sequential branch' (SEQ), 'conditional branch' (BC), 'call', 'return' (RTN), and 'jump' (JMP).

- In sequential branching, the next micro-word to the present one in the control ROM is to be loaded into the micro-instruction register. The micro-program

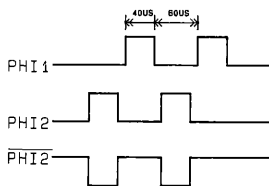


Fig. 9 Clocking Scheme of the Micro-Control Unit

counter holds the address of the next micro-word.

- When 'conditional branch' is executed, the next micro-instruction to be loaded is determined by decoding the FLAG field. The appropriate data is loaded into the register REG.
- When 'call' is encountered, the contents of the micro-program counter are saved on the micro-stack and the micro-stack pointer is incremented by one. The micro-instruction 'call' is of two types: 'conditional call' (BC/CALL), and 'direct call' (CALL). During 'direct call', the register DATA is loaded from the NROM. The ADDRESS field decides which data to be loaded while 'conditional call' is essentially same as 'conditional branch' except for the fact that the micro-program counter contents are saved in the case of 'conditional call'.
- The micro-instruction 'return' decrements the micro-stack pointer by one.
- 'Jump' is similar to 'direct call' but the micro-program counter contents are not saved.

B. Phase II

- 1) During the second phase, the data transfer from registers to data buses takes place. The appropriate control signals are already generated during the first phase.
- 2) The contents of the micro-program counter are loaded into the incrementer and incremented by one.
- 3) The multiplexer selects the data to be loaded into the ROM address register. There are four registers to be selected from. They are DATA, micro-program counter, micro-stack, and REG. The contents of the micro-program counter are loaded in the case of a 'sequential branch'. Execution of the micro-

instruction 'return' initiates the loading of the micro-stack pointed to by the micro-stack pointer. The REG contents are loaded when 'conditional branch' or 'conditional call' is encountered. In the remaining two cases of 'direct call' and 'jump', DATA is loaded into the control store.

C. Phase III

The control word is loaded into the micro-instruction register at the leading edge of $\Phi_{2\text{bar}}$.

CHAPTER V

CIRCUIT DESCRIPTION OF THE MICRO-CONTROL UNIT

In this chapter, the Mentor Graphics component library developed by the MOSIS Engineers was a major initiative to design the micro-control unit at gate level. The NETwork EDitor (NETED) and the SYMBol EDitor (SYMED), a system of software tools on the IDEA Station, have been used to capture the micro-control unit at the functional level and the gate level. The component library contains primitive gates such as AND, OR, NAND, NOR, and etc. Due to lack of the transistor cells in this library, memory cells such as RAM, ROM, and PLA had to be implemented with the CHIPGRAPH software of the CHIP station. It should be noted that the micro-control unit has been designed in a hierarchical fashion. Fig. 10 depicts the root symbol of the micro-control unit.

The different sections of the Micro-control unit are discussed below.

A. Two Phase Clock

A two phase non-overlapping clock is designed for the micro-control unit. This clock can be thought of as a three phase clock by inverting ϕ_2 since the micro-instruction register uses ϕ_2 bar. The circuit is shown in Fig. 11.

B. Combinational Logic

The FLAG fields, F(15:0), in conjunction with the information from the data registers and binding memory register decide the next word to be loaded into REG. There are thirty three signals produced which are R(33:1). The Instruction Set Processor (ISP) description of each routine generating these signals is obtained by P. Parikh [9]. Next, the routines employed in the COMB.LOGIC are described.

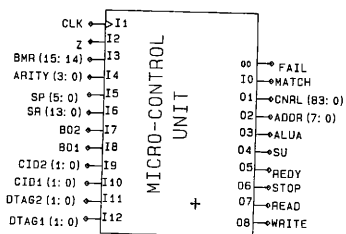


Fig. 10 Root Symbol of the Micro-Control Unit

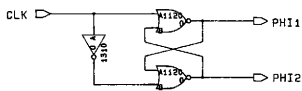


Fig. 11 Two Phase Clock

TAGDECODE

The fields CID1(1:0) and CID2(1:0) are decoded with the enable signal F(1). The signal is enabled when F(1) is active. The fields DTAG1(1:0) and DTAG2(1:0) use decoders shown in Fig. 12. The outputs signals generated are D(16:1).

SETCMBIT

Refer to Fig. 13. According to certain combinations of the decoded signals D(16:1) from TAGDECODE routine, the following signals are generated. The signals R(6:4) and R(27:22) are produced when F(0) and F(7:6) are active, respectively. The signal R(31) is generated by enabling F(11). The signals DTAG2(1:0)=DTAG1(1:0)=11 and CID2(1:0)=CID1(1:0)=11 signifies the operation is successful ; that is, signal S_U is logic 1.

CAMSRCH

The routine CAMSRCH is implemented and the output signals are generated when the following occur. The signals BMR(15:14) from the Binding Memory Register (BMR) and MATCH signal from Content Addressable Memory (CAM) decide which one of the signals (R(13:8)) is generated providing either F(2) or F(3) is enabled. The signal MATCH enables the data transfer between the binding memory and the register BMR. The circuit is shown in Fig. 14. The signal DEC.BOUND.VV, F4_1, is true when F(3) is active and either BMR(15:14)=00 or BMR(15:14)=01.

DECODE.SR.SP.ARTY

Fig. 15 shows the implementation of this routine. The signal F(10) enables the SR input to the 4-input OR gates. When the content of SR(13:0) is zero, the AND gates for generating R(20) and STOP are enabled. If SP(5:0) is also zero, then

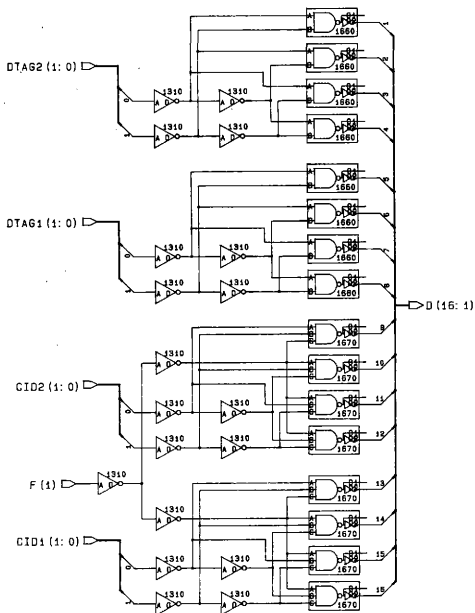


Fig. 12 TAGCODE Routine

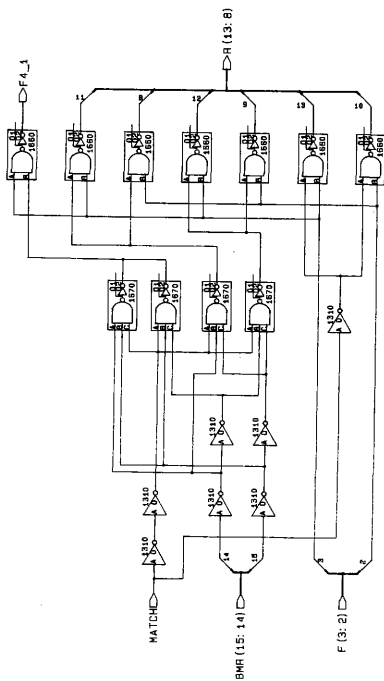


Fig. 14 CAMSRCH Routine

R(20) will be logic 1, otherwise the STOP signal is generated. The signal R(21) is enabled if either SR(13:8) or ARITY(3:0) is non-zero. The signal REDY is enabled if ARITY(3:0) and SP(5:0) both are zero.

DEC.BOUND.VV

This routine is implemented with logic gates shown in Fig. 16. The signals R(19:14) are generated when the signal F4(.1) from CAMSRCH routine is logic 1 and certain combinations of DTAG2(1:0), DTAG1(1:0), B01, and B02 occur.

DEC.BOUND.VC

DEC.BOUND.VC generates the following signals. The signal FAIL is activated if F(9) is enabled and Z flag from ALU is disabled. The signal R(30) is active providing ARITY(3:0) is non-zero and both Z flag and signal F(9) is logic 1. However, if ARITY(3:0) is logic 0, the signal R(20) is generated. If B01 is zero, then R(28) is logic 1, otherwise R(29) signal is activated providing F(8) is enabled. The signals R(32) and R(33) are generated if BMR(15:14) is 10 and both F(12) and F(13) is enabled, respectively. The schematic of this routine is given in Fig. 17.

C. Instruction Decode

The current micro-instruction held in the micro-instruction register is decoded by instruction decode (INSTR_DEC) routine. The outputs ADDR(7:0), S(3:0), and F(15:0) generated from DEC_1, and DEC_2 routines are used in the micro-control unit itself to decide the next instruction to be executed. The gate level description of DEC_1 and DEC_2 are shown in Fig. 18 and Fig. 19, respectively. Whereas, the outputs CNRL(83:0) generated from other six decoders are control signals shown in Figs. 20-25.

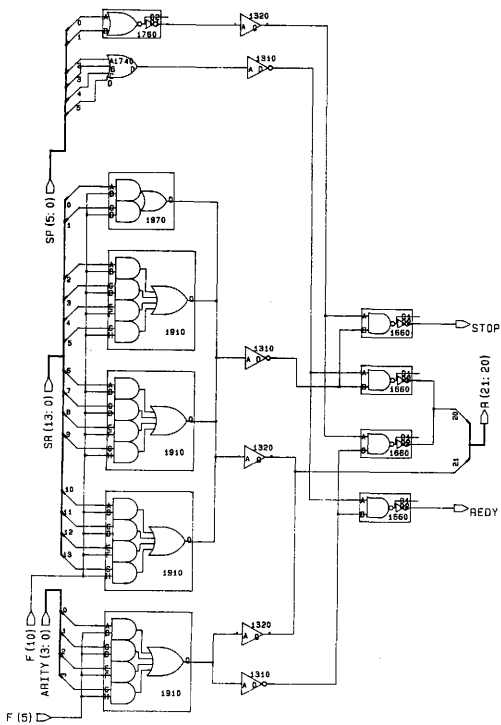


Fig. 15 DECODE.SR.SP.ARITY Routine

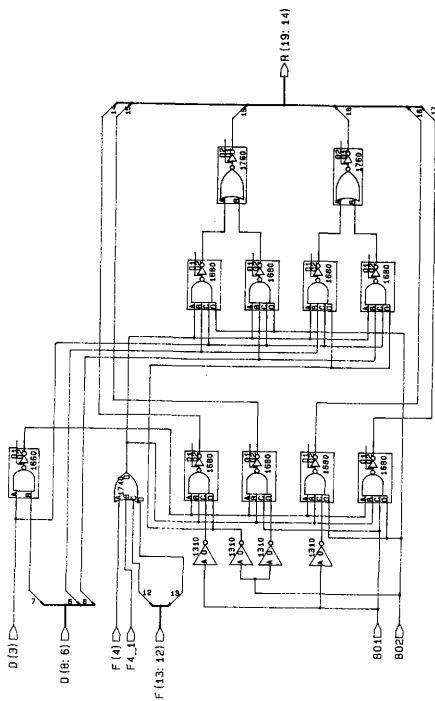


Fig. 16 DEC.BOUND.VV Routine

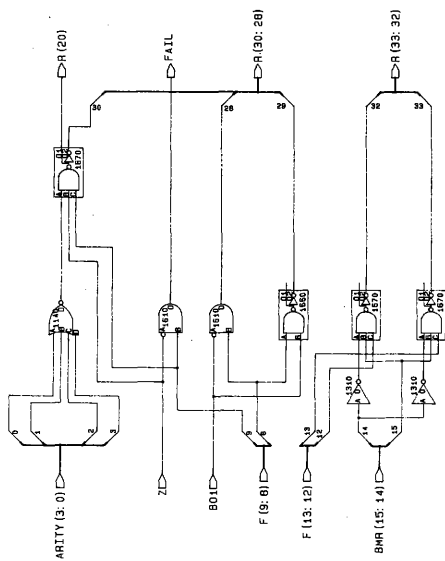


Fig. 17 DEC.BOUND.VC Routine

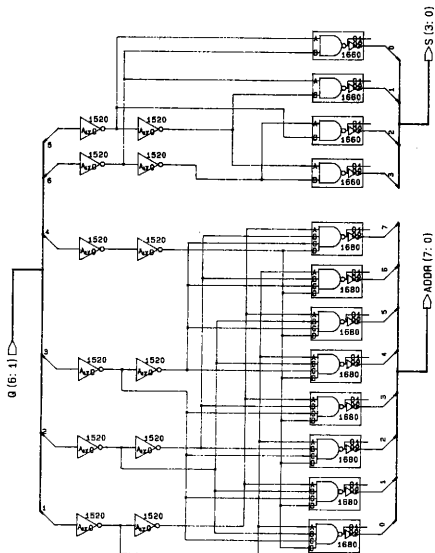


Fig. 18 DEC.1

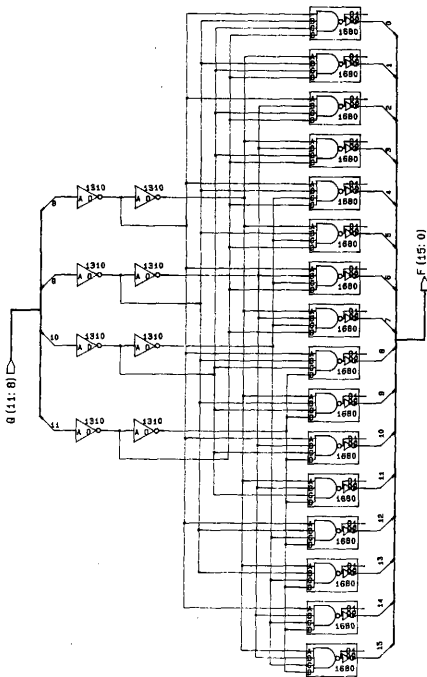


Fig. 19 DEC.2

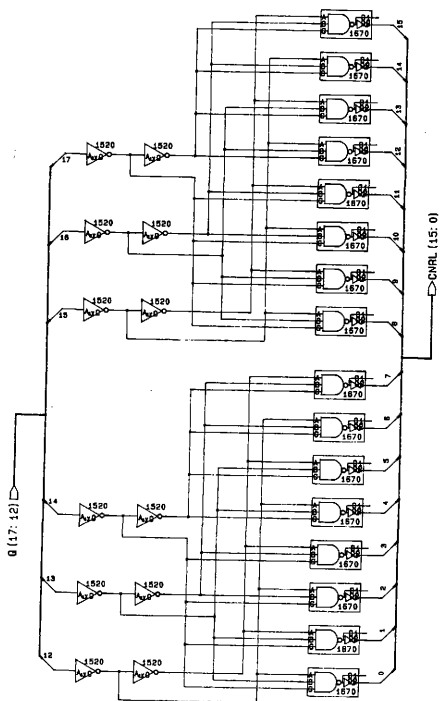


Fig. 20 DEC.3

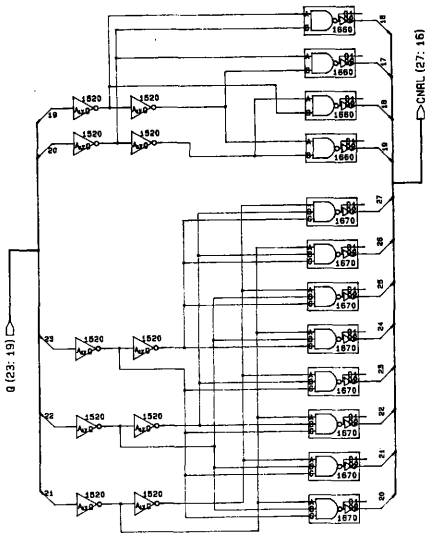


Fig. 21 DEC_4

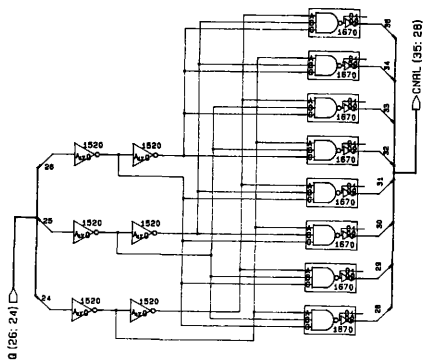


Fig. 22 DEC_5

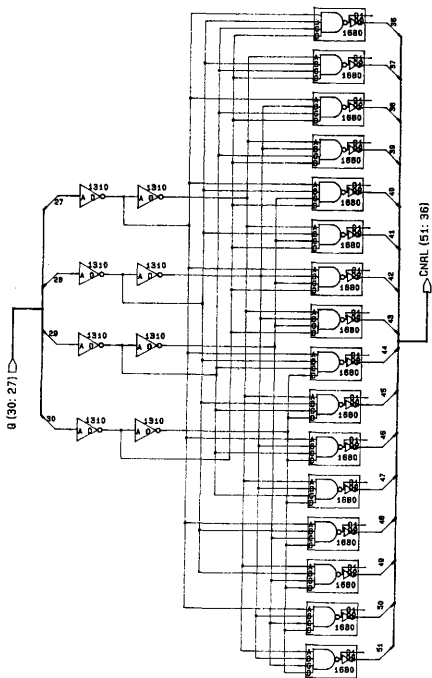


Fig. 23 DEC_6

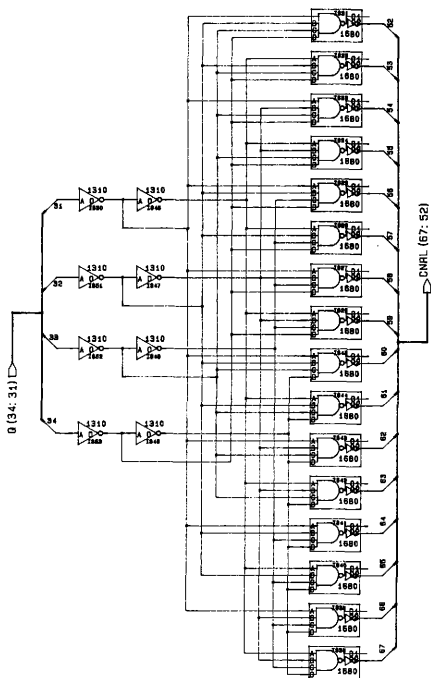


Fig. 24 DEC.7

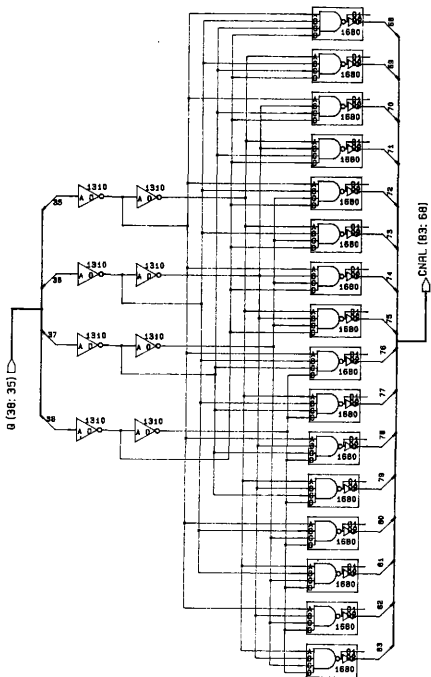


Fig. 25 DEC.8

D. Microsequencer

The microsequencer (mSEQUENCER) decodes different fields like ADDRESS (ADDR(7:0)), FLAG (F(15:0)), and STATE (S(3:0)) and selects the proper data from mSTACK, mPC, REG or DATA. The different sections of mSEQUENCER are implemented as follows.

Micro-Stack Pointer

A 3-bit synchronous up/down binary counter is designed as the micro-stack pointer (mSP). It maintains a fully-independent clock circuit and can be used as a register as well. The input is enabled when the STATE fields, S(1) or S(2), is high. The up/down mode of counting is decided by the inputs S(2:1) as follows.

- S(2:1)=00 disables the counter.
- S(2:1)=10 signifies counting up sequence.
- S(2:1)=01 signifies counting down sequence.
- S(2:1)=11 condition is not possible.

Fig. 26 shows the implementation of the design.

Micro-Stack

A 8x7 Static Random Access Memory (SRAM) is designed and implemented as a micro-stack (mSTACK). The outputs from the micro-stack pointer are connected to the address inputs (ADR(2:0)). The read/write mode depends on the signal S(1) and S(2) as follows.

- S(2:1)=00 signifies power down mode.
- S(2:1)=10 signifies write operation.
- S(2:1)=01 signifies read operation.
- S(2:1)=11 condition is not possible.

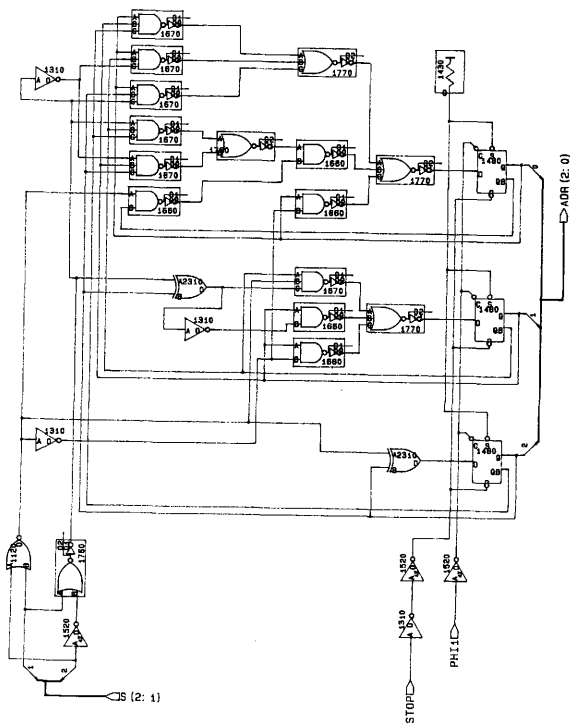


Fig. 26 Micro-Stack Pointer

The micro-stack is enabled when S(1) is active, otherwise it remains in the power down mode. The data inputs and data outputs are bi-directional. They are connected to the outputs from the micro-program counter and are connected to the PLA inputs. A 4:1 design rule was followed for dimensioning of the transistors in the micro-stack design. The following is a listing of the W/L ratios for the transistors throughout the design.

- *Address Buffers:* PMOS-10/5, NMOS-5/10
- *Address Decoders:* PMOS-50/5, NMOS-5/10
- *Data Buffers:* PMOS-10/5, NMOS-5/10
- *Memory Cells:* PMOS-10/5, NMOS-5/10
- *Pass Gates:* PMOS-10/5, NMOS-5/10 at all locations.

As seen in the listing for the address decoders, the PMOS W/L ratio is quite large compared to the other transistor dimensions. This is due to the fact that the decoders are configured as 4-input NOR gates whose channel lengths add to give an effective length of 20. Therefore, a 5:1 design rule was followed in this case. The following is a listing of the transistor counts of the micro-stack.

- *Address Buffers:* 14 (3 total; includes enable inverter).
- *Address Decoders:* 30 (8 total).
- *Data Buffers:* 84 (7 total).
- *Memory Cells:* 336 (56 total).

A schematic of the micro-stack spacing locations of each component, followed by the transistor level design of each component is shown in Figs. 27-30. The physical layout description of the micro-stack will be discussed in Chapter VI.

Micro-Program Counter

A 7-bit binary counter with input registers (mPCI) and seven bit registers

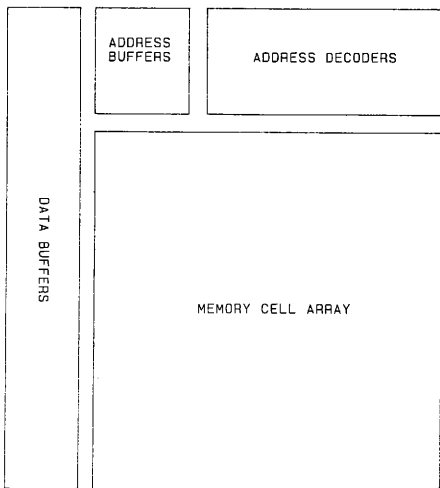
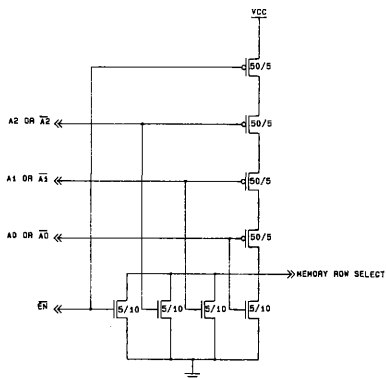
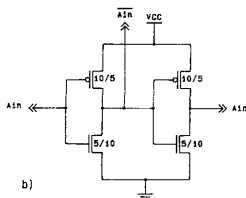


Fig. 27 Layout Spacing of the Micro-Stack



a)



b)

Fig. 28 Micro-Stack

a) Address Decoder b) Address Buffer

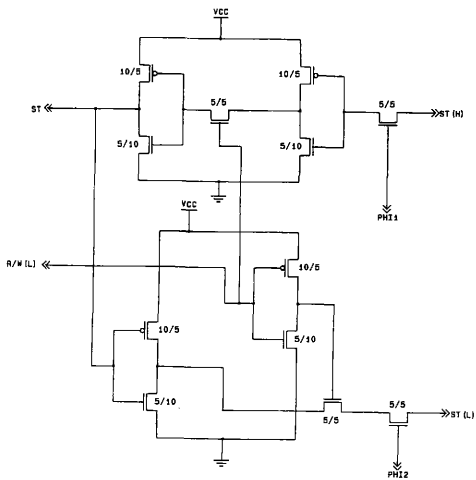


Fig. 30 Data Buffer of the Micro-Stack

(mPCO) are designed to function as a micro-program counter. The counter/register is enabled as soon as the UNIFIC becomes active; that is; the signal REDY is logic 0. The same clock is used for the counter as well as the register. The counter state, hence, will equal the previous contents plus one when the clock arrives. The contents are cleared when the STOP signal is logic 1. The inputs to the register come from the outputs of the micro- stack. The registers are loaded when S(1) is enabled. The outputs of the mPCI go to the inputs of micro-stack and to mPCO. The mPCO register is enabled when the signal P_C is active. The gate level description of mPCI, followed by mPCO are shown in Figs. 31-32.

Register DATA

Fig. 33 shows the design of a 7-bit register implemented as a register DATA. The register data is loaded from nROM when the address field in the micro-instruction contains a non-zero value.

nROM

A Read-Only Memory (ROM) is designed so as to implement DATA. The address decoder of the ROM is designed at the gate level. The memory cell array portion of the ROM (basically an OR-matrix of a PLA) has been generated using the PLA generator available on the Apollo workstation. The circuit is shown in Fig. 34 and the memory array table is given in Appendix C. The four bits of the address field Q(4:1) from the micro-instruction register are connected to the address inputs of the nROM. It is enabled when S(3) is logic 1 or S(2) and A(0) are both high. The outputs of the nROM are connected to the inputs (DTI(6:0)) of register DATA. The register data is selected providing DATA signal is enabled. The implementation of the nROM will be discussed in Chapter VI.

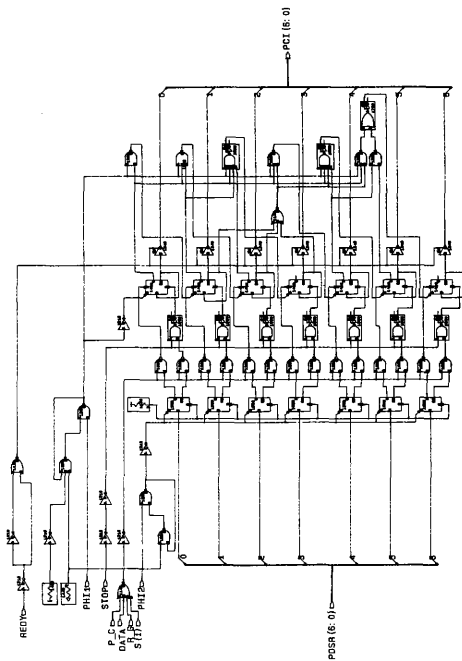


Fig. 31 mPCI



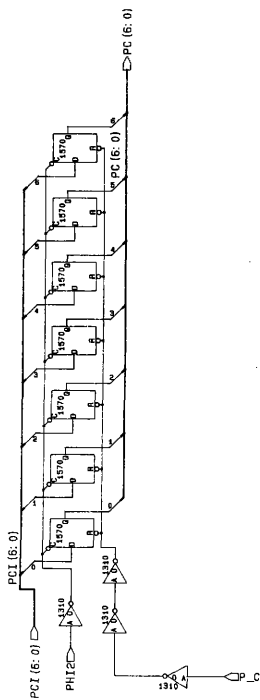


Fig. 32 mPCO

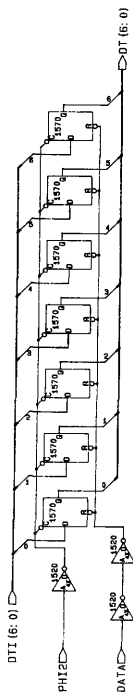


Fig. 33 Register DATA

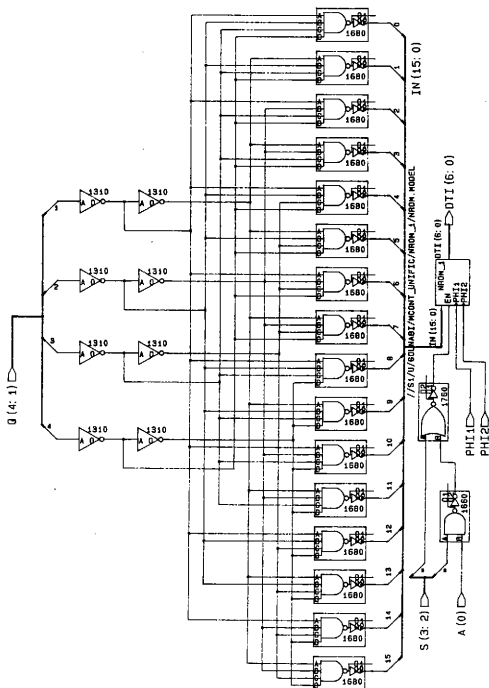


Fig. 34 nROM

Register REG

The signals $F(15:0)$ in conjunction with the information from the data registers decide the next word to be loaded into REG. The register REG is selected when R_G signal is active. The outputs $REG(6:0)$ are fed into the input of micro-program counter, $PDSR(6:0)$. The design of the register REG is shown in Fig. 35.

Multiplexer

This routine generates the DATA, the P_C , and the R_G signals according to the STATE field ($S(3:2)$, $S(0)$), the FLAG field ($F(0)$), and the ADDRESS field ($A(0)$). The circuit is given in Fig. 36.

E. Array

The signals $R(33:1)$ generated by combinational logic routine are fed to the inputs of an array. This array is nothing but a mapping table shown in Appendix C. The instruction set processor (ISP) description of each routine generating these signals is obtained by P. Parikh [9]. This array can also be described as an OR-matrix of a PLA. The PLA generator provided with the Apollo workstation has been employed to implement the array which will be discussed in Chapter VI. The data selected from this array is loaded into the register REG when R_G is active. The outputs $REG(6:0)$ are connected to the inputs of the PLA.

F. Control Store

The control store is designed so as to be implemented by a PLA. It has 7 inputs and 39 outputs. The equation for each output is obtained by P. Parikh [9]. The function $F(0)$ corresponds to the MSB of the control word while $F(38)$ is the LSB.

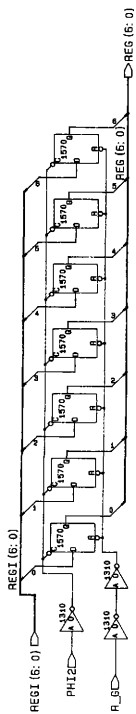


Fig. 35 Register REG

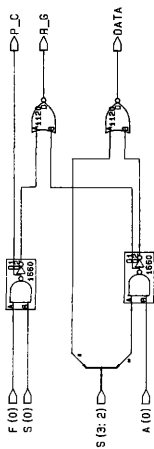


Fig. 36 Multiplexer

The EQNTOTT software on the Apollo workstation has been utilized to convert the output equations into the AND and OR matrix data for the PLA. Then, the ESPRESSO (a boolean minimizer software) has been run to minimize the product terms of the PLA. The result is shown in Appendix C. Note that in this table the AND plane data is given in square brackets while the OR plane data is given in its true form. The final minimized PLA has 7 inputs, 70 product terms, and 39 outputs. The implementation of the PLA will be described in Chapter VI.

G. Micro-Instruction Register

A 39 bit register is designed to function as a micro-instruction register (mIR). It is always enabled and is loaded at the rising edge of PHI1. The circuit diagram of the micro-instruction register is depicted in Fig. 37.

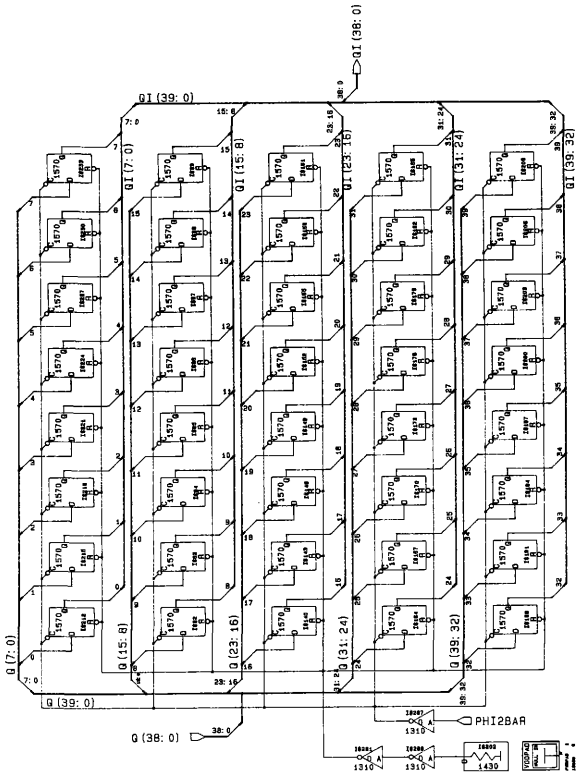


Fig. 37 Micro-Instruction Register

CHAPTER VI

LAYOUT DESCRIPTION OF THE MICRO-CONTROL UNIT

In this chapter, the layout description of the micro-control unit is given. The component cell library developed by the MOSIS Engineers was a major initiative to design the micro-control unit at the gate level. The Mentor Graphics CELL Station has been utilized to implement the physical layout of standard cells. Since the component library contains only primitive gates, memory cells have been implemented with CHIPGRAPH (a part of the CHIP station). These cells have then been incorporated into the CELL station by a set of software tools within the Mentor Graphics.

A. Cell Station Process Flow

The CELL station consists of consists of three sections: inputs, processing modules and output.

Inputs

The CELL station requires two inputs: process design rules, and logic input.

Process Design Rules

The process design rules define chip technology. It sets constraints such as grid and track spacing, power voltage and wire style. This input is contained in the CELL station technology directory.

Logic Input

As mentioned earlier in Chapter III, a logical design file (DESIGN.EREL) was created by expanding the micro-control unit. This file contains all the necessary information to be used for the layout process.

Processing Modules

These modules are a set of automatic programs used for the layout process of the micro-control unit. Typically, these programs handle most of the work. However, CELLGRAPH has utilized for interactive editing. The physical layout tools employed for the implementation of the standard cells of the design are described below:

LOGIC_ENTRY

This is the first step in the layout process of a standard cell design of the micro-control unit. The physical design file was created from an expanded logical design file by LOGIC_ENTRY software. This file was used by standard cell layout applications which collect output from all subsequent layout operations performed on the micro-control unit. Completion of LOGIC_ENTRY sets up the next step, generation of the floorplan for the design.

CELLFLOOR

The floorplan of the circuit was automatically generated by CELLFLOOR program. EDIT_PARMs (an interactive program) was used to modify the floorplan parameters.

CELLPLACE

The placed standard cells of the micro-control unit has been generated by CELLPLACE program. The macros in the micro-control unit were placed automatically.

CELLPOWER

The power nets of the micro-control unit were routed by CELLPOWER program (an automatic program).

CELLROUTE

The placed standard cells of the micro-control unit was globally routed and followed by detailed routing of the signal nets.

CELLSQUEEZE

It is the function of CELLSQUEEZE to remove excess tracks of the routing channels of the micro-control unit.

MINROUTE

It minimizes the amount of poly used in the micro-control unit by post-routing.

Fig. 38 illustrates the micro-control unit after MINROUTE.

CELLVERIFY

It does a final check on the validity of the layout produced by CELL station.

Such as, design rule violations and the electrical connectivity. CELLGRAPH was used for interactive editing of the micro-control unit whenever it was needed.

Output

At the end of the layout process, the chip file was converted into Calma GDSII format by GDSII.OUTPUT program for fabrication.

B. Chip Station Process Flow

As mentioned earlier in Chapter III, The micro-control unit memory cells have been implemented with the CHIP station using CHIPGRAPH layout editor. Due to the lack of CHIPGRAPH's design rule checker, the cells have been checked using MAGIC layout editor. To carry out this task, a process definition file is written for the MCIF program to convert a CIF file to CHIPGRAPH database or vice versa. Since MAGIC's CIF file had some layer conflicts with the standard CIF file, a technology file is written so that it is compatible with the standard CIF required for

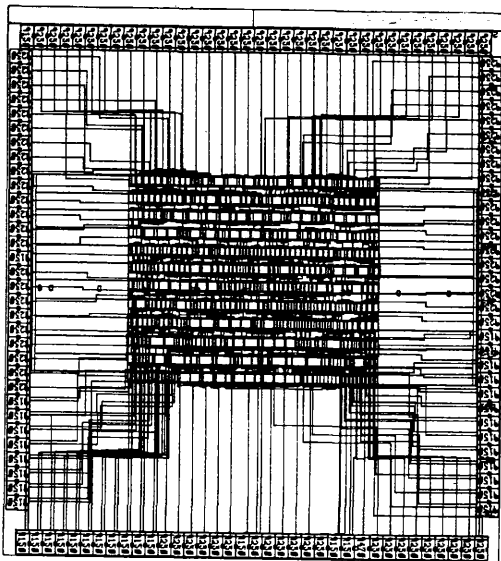


Fig. 38 Standard Cell Design of the Micro-Control Unit

the MCIF program. Having implemented all the memory cells, a CELL_MODEL file and a CELL_LIST file are written for BUILD_LIB and TDF_CHIP_INPUT softwares to convert cells in CHIPGRAPH database into the cells in the CELL station database. The CELL station's LOGIC_ENTRY software is utilized to incorporate these cells into the micro-control unit standard cells. APPENDIX A Contains all the mentioned technology files. Next, The different sections of the memory cells are discussed.

Array

This array is basically an OR-matrix of a PLA. The PLA generator on the Apollo workstation has been employed to implement the array. The layout has been checked with MAGIC's design rule checker and converted to CHIPGRAPH database by the above mentioned technology files. The layout of the array is given in Fig. 39.

Micro-Stack

Micro-stack was designed at transistor level and implemented using CHIPGRAPH. The layout of the micro-stack is shown in Fig. 40. The micro-stack layout was converted from CHIPGRAPH to MAGIC by MCIF program for design rule check. After completion, the CIF file was converted back to the CHIPGRAPH format.

nROM

Fig. 41 shows the implementation of the PLA. As mentioned earlier, the address decoder of the nROM was designed at the gate level. The memory cell array of the nROM was implemented using the PLA generator (MPLA) software. After checking the layout by MAGIC, the file is converted to CHIPGRAPH database as before.

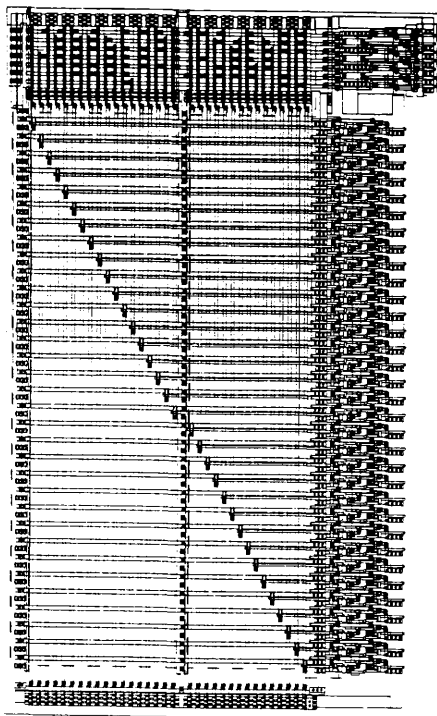


Fig. 39 The Array Layout

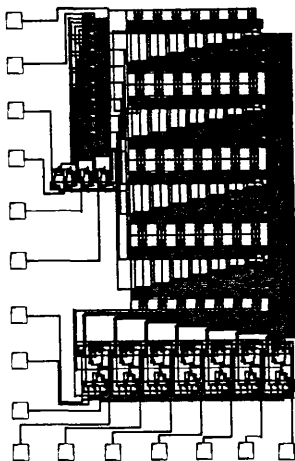


Fig. 40 The Micro-Stack Layout

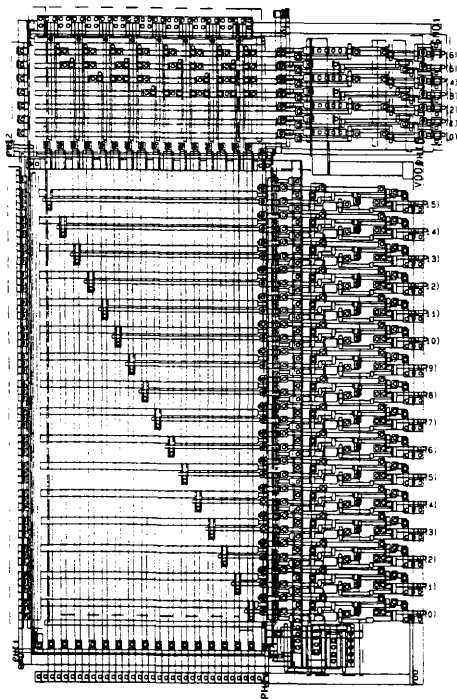


Fig. 41 The nROM Layout

PLA

The EQUNTOTT software of the Apollo workstation has been utilized to convert the output equations into the AND and OR matrix data for the PLA. The ESPRESSO (a boolean minimizer) program has been run to minimize the product terms of the PLA. The PLA layout has then been generated by the MPLA program. The layout was fully checked for design rule violations and is given in Fig. 42.

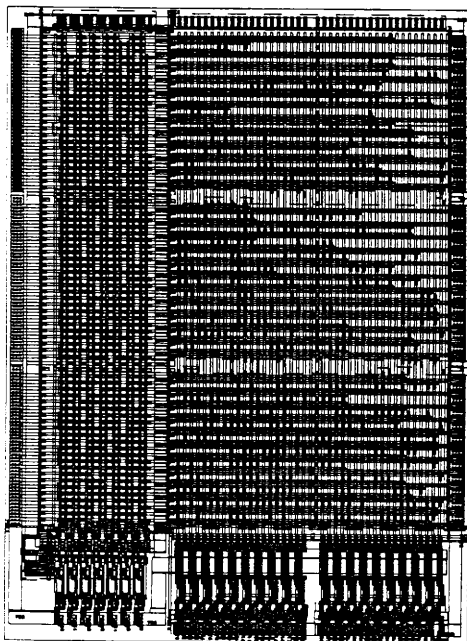


Fig. 42 The PLA Layout

CHAPTER VII

SIMULATION RESULTS AND CONCLUSIONS

In this chapter, the simulation results of the micro-control unit are presented. The performance of the micro-control unit is compared with Parikh's functional simulation [9]. The steps accomplished to facilitate a more comprehensive digital design capability by mixing both standard cell and custom design are summarized. Finally suggestions for future work is given.

A. Simulation Results

Due to fact that the MOSIS component library does not provide transistor cells, the micro-control unit was designed in a hierarchical fashion described by the following two simulation tools:

- 1) IDEA station's QUICKSIM software for gate level sections.
- 2) SPICE for micro-stack, array, nROM, and PLA.

As a result, the micro-control unit was not simulated as a complete system. Table I shows the results obtained for two expressions with an increasing number of terms. These results agree with Parikh's functional simulation. Because the execution unit of the UNIFIC has not been layed out, the simulated results of unification of terms with increasing number of nested function arguments still rely on the functional assumptions made in Parikh's work. All of the gate level simulations of the micro-control unit are given in Appendix B. The information from the simulations shows that the actually UNIFIC chip, in a CMOS layout, should easily perform as well as functional simulations have predicted.

TABLE I.
Simulation Results for Two Expressions
with Increasing Number of Terms

Arity	Expression 1	Expression 2	Time (μ s)
1	x_1	A_1	1.9
2	x_1, x_2	A_1, A_2	3.65
3	x_1, \dots, x_3	A_1, \dots, A_3	5.5
4	x_1, \dots, x_4	A_1, \dots, A_4	7.3
5	x_1, \dots, x_5	A_1, \dots, A_5	9.1
6	x_1, \dots, x_6	A_1, \dots, A_6	10.9
7	x_1, \dots, x_7	A_1, \dots, A_7	12.7
8	x_1, \dots, x_8	A_1, \dots, A_8	14.5
9	x_1, \dots, x_9	A_1, \dots, A_9	16.3
10	x_1, \dots, x_9 y_1	A_1, \dots, A_9 B_1	18.1
11	x_1, \dots, x_9 y_1, y_2	A_1, \dots, A_9 B_1, B_2	19.9
12	x_1, \dots, x_9 y_1, \dots, y_3	A_1, \dots, A_9 B_1, \dots, B_3	21.7
13	x_1, \dots, x_9 y_1, \dots, y_4	A_1, \dots, A_9 B_1, \dots, B_4	23.5
14	x_1, \dots, x_9 y_1, \dots, y_5	A_1, \dots, A_9 B_1, \dots, B_5	25.3
15	x_1, \dots, x_9 y_1, \dots, y_6	A_1, \dots, A_9 B_1, \dots, B_6	27.1
16	x_1, \dots, x_9 y_1, \dots, y_7	A_1, \dots, A_9 B_1, \dots, B_7	28.9

B. Conclusions

The goal of this work was to design the micro-control unit using Mentor Graphics software package, and to interface custom layout facilities with standard cell design tools. The results obtained for the unification process strongly agrees with Parikh's work. As a result of designing the UNIFIC micro-control unit tools to incorporate custom layout into the standard cell design on the Apollo workstations was accomplished. Employing the technology files, CELL_LIST and CELL_MODEL files, given in Appendix A, allow the merger of standard cell tools from the Mentor Graphics with custom design tools, either the Mentor Graphics or magic, for comprehensive system designs. The following extensions are suggested for future work:

- 1) Design the execution unit of the UNIFIC.
- 2) Implement the UNIFIC by interfacing the existing micro-control unit with the execution unit using standard cell layout.
- 3) Enhance the simulation tools for more convenient merger of standard cell and customized designs.
- 4) Provide user Friendly interfaces for guidance between the standard cell functions and custom functions.

REFERENCES

- [1] J. Crammond, "A Comparative Study of Unification Algorithms for OR-Parallel Execution of Logic Languages," *IEEE Transactions On Computers*, C-34, 911-917, October 1985.
- [2] T.P. Dobry, A.M. Despain, Y.N. Patt, "Performance Studies of a Prolog Machine Architecture," in *Proceedings of the 12th Annual International Symposium on Computer Architecture*, Boston, Massachusetts, June 17-19, 1985, pp. 180-190.
- [3] R. Gollakota, "Design, Simulation and Analysis of A Coprocessor for the Unification Algorithm," *Research Report*, Master's Thesis, Department of Electrical Engineering, Texas A&M University, May 1985.
- [4] T. Moto-aka, H. Tanaka, H. Aida, K. Hirata, T. Maruyama, "The Architecture of a Parallel Inference Engine-PIE," in *Proceedings of the International Conference on Fifth Generation Computer Systems*, Tokyo, Japan, November 6-9, 1984, pp. 479-488.
- [5] N.J. Nilsson, *Principles of Artificial Intelligence*. Palo Alto, CA: Tioga Publishing Company, 1980.
- [6] R. Nakazaki, et al., "Design of a High-speed Prolog Machine (HPM)," in *Proceedings of the 12th Annual International Symposium on Computer Architecture*, Boston, Massachusetts, June 17-19, 1985, pp. 191-197.
- [7] J.V. Oldfield, "Logic Programs and an Experimental Architecture For Their Execution," *IEE Proceedings*, 133, 163-167, May 1986.
- [8] C.A. Papachristou, R. Rashid, S.B. Gambhir, "VLSI Design of a PLA Based Microcontrol Scheme," in *Proceedings of IEEE International Conference on Computer Design: VLSI in Computers*, Port Chester, New York, October 8-11, 1984, pp. 771-777.
- [9] P. Parikh, "VLSI design of the controller for the UNIFIC, A Prolog unification coprocessor," *Research Report*, Master's Thesis, Department of Electrical Engineering, Texas A&M University, May 1987.
- [10] C.G. Ponder, Y.N. Patt, "Alternative Proposals for Implementing Prolog Concurrently and Implications Regarding Their Respective Microarchitecture," in *Proceedings of the Seventeenth Annual Microprogramming Workshop*, New Orleans, Louisiana, 1984, pp. 192-203.
- [11] J. Robinson, "A Machine-Oriented Logic Based on the Resolution Principle," *Journal of ACM*, 10, no. 1, 23-41, January 1965.

- [12] P. Robinson, "The SUM: an AI coprocessor," *Byte*, 10 (6), 169-180, 1985.
- [13] S. Shobatake, H. Aiso, "A Unification Processor Based on a Uniformly Structured Cellular Hardware," in *Proceedings of the 13th Annual International Symposium on Computer Architecture*, Tokyo, Japan, June 2-5, 1986, pp.140-148.
- [14] K. Taki, et al, "Hardware Design and Implementation of the Personal Sequential Inference Machine (PSI)," in *Proceedings of the International Conference on Fifth Generation Computer Systems*, Tokyo, Japan, November 6-9, 1984, pp. 398-409.
- [15] N. Tamura, K. Wada, H. Matsuda, Y. Kaneda, S. Maekawa, "Sequential Prolog Machine PEK," in *Proceedings of the International Conference on Fifth Generation Computer Systems*, Tokyo, Japan, November 6-9, 1984, pp. 542-550.
- [16] E. Tick, "Sequential Prolog Machine: Image and Host Architectures," in *Proceedings of the Seventeenth Annual Microprogramming Workshop*, New Orleans, Louisiana, February 1984, pp. 204-206.
- [17] E. Tick, H.D.Warren, "Towards a Pipelined PROLOG Processor," in *Proceedings of the International Symposium on Logic Programming*, Atlantic City, NJ, 1984, pp. 29-40.
- [18] S. Uchida, T. Yokoi, "Sequential Inference Machine: SIM-Progress Report," in *Proceedings of the International Conference on Fifth Generation Computer Systems*, Tokyo, Japan, November 6-9, 1984, pp.58-69.
- [19] N.S. Woo, "A Hardware Unification Unit: Design and Analysis," in *Proceedings of the 12th Annual International Symposium on Computer Architecture*, Boston, Massachusetts, June 17-19, 1985, pp. 198-205.
- [20] M. Yokota, et al., "A Microprogrammed Interpreter for the Personal Sequential Inference Machine," in *Proceedings of the International Conference on Fifth Generation Computer Systems*, Tokyo, Japan, November 6-9, 1984, pp. 410-418.
- [21] P. R. Cohen and E. D. Feigenbaum, *The Handbook of Artificial Intelligence*. Vol.3 Stanford: William Kaufmann, Inc., 1982.

APPENDIX A

THE TECHNOLOGY, CELL_MODEL, AND CELL_LIST FILES

```

*
** .....
** This technology file is written to generate a
** standard CIF file from the MAGIC environment
** for the MCIF conversion program. Cifoutput
** and Cifinput routines are combined with the
** scmos.tech20 file to accomplish this task.
** .....
*
tech
  kludge
end
*
** Cifoutput routine....
*
cifoutput
style lambda=1.5(chipgraph)
  scalefactor 150 25
  layer CWF pwell
bloat -or ndiff,ndc,nfet * 750
  bloat -or psc,ppd * 450
grow 450
shrink 450
calma 41 1
  layer CMS m2,m2c/m2,pad/m2
labels m2
calma 51 1
  layer CMF pad
grow 150
or m1,m2c/m1,pc/m1,ndc/m1,pdc/m1,ppcont/m1,nncont/m1,pad/m1
labels m1,m2c/m1,pc/m1,ndc/m1,pdc/m1,ppcont/m1,nncont/m1,pad/m1
calma 49 1
  layer CPG poly,pc/active,nfet,pfet
labels poly,nfet,pfet
calma 46 1
  layer CAN ndiff,nfet,ndc,nncont
  labels ndiff
  calma 42 1
  layer CAP pdiff,pfet,pdc,ppcont
  labels pdiff
  calma 43 1
  layer CVA pad
shrink 450
calma 50 1
  layer CVA m2c
squares 150 300 450

```

```
shrink 600
or glass
calma 52 1
  layer XP pad
end
*
** Cifinput routine....
*
cifinput
style lambda=1.5(chipgraph)
  scalefactor 150
  layer pwell CWP
labels CWP
  layer m2 CMS
labels CMS
  layer m1 CMF
labels CMF
  layer poly CPG
labels CPG
  layer ndiff CAN
labels CAN
  layer pdiff CAP
labels CAP
  layer pfet CPG
and CAP
  layer nfet CAN
and CPG
and CWP
  layer m2c CVA
grow 225
shrink 75
and CMS
and CMF
  layer ncont CC
grow 150
and CAN
and CMF
  layer pdc CC
grow 150
and CAP
and CMF
  layer ndc CC
grow 150
and CAN
and CWP
and CMF
  layer ppeont CC
grow 150
and CAP
```



```
and CWP
and CMF
  layer pc CC
grow 150
and CPG
and CMF
  layer glass COG
  layer pad CMF
shrink 150
and CMS
shrink 450
and CVA
shrink 150
and COG
grow 600
and XP
  calma CWP 41 *
  calma CAN 42 *
  calma CAP 43 *
  calma CPG 46 *
  calma CC 47 *
  calma CMF 49 *
  calma CVA 50 *
  calma CMS 51 *
  calma COG 52 *
end
```

```

#
## -----
## This is a Process Definition File (PDF) written for converting
## standard CIF file into CHIPGRAPH database.
## -----
#
# ATTRIBUTE      DOMAIN
# -----
#
# Transparency   ( transparent,opaque,xor,no_border )
# Pattern        ( none,solid,'b'..'z' )
# Fill_color     ( red,green,blue,yellow,magenta,cyan,black,white,
# Fill_color     purple,gray,'light blue','yellow green',pink,beige )
# Line_color     ( red,green,blue,yellow,magenta,cyan,black,white,
# Line_color     purple,gray,'light blue','yellow green',pink,beige )
# Line_style     ( solid,dotted,short_dash,long_dash )
# Line_width     1..32
# Text_color     ( red,green,blue,yellow,magenta,cyan,black,white,
# Text_color     purple,gray,'light blue','yellow green',pink,beige )
#
transcribing on
# Loading double metal CMOS process definition ...
# Default minimum resolution : 0.001 microns
transcribing off
define process KLUDGE 0.001 micron
#
#
define layer name cwp      1  -shape -path -instance -pin
define layer name cwp.ext 41  -perimeter -port
define layer name cap      3  -shape -path -instance -pin
define layer name cap.ext 43  -perimeter -port
define layer name can      4  -shape -path -instance -pin
define layer name can.ext 44  -perimeter -port
define layer name cp      6  -shape -path -instance -pin
define layer name cp.ext 46  -perimeter -port
define layer name cc      7  -shape -path -instance -pin
define layer name cc.ext 47  -perimeter -port
define layer name cmf      6  -shape -path -instance -pin
define layer name cmf.ext 48  -perimeter -port
define layer name cms      9  -shape -path -instance -pin
define layer name cms.ext 49  -perimeter -port
#
define alias cwp          1
define alias cap          3
define alias can          4
define alias cp           6
define alias cc           7
define alias cmf          8
define alias cms          9

```

```

#
# some useful layer groups
#
define layer group trans  cwp cap can cc cpg
define layer group int   cmf cc cms
#
# physical layer groups
#
define physical layer group PW cwp cwp.ext
define physical layer group ND can can.ext
define physical layer group PO cpg cpg.ext
define physical layer group PD cap cap.ext
define physical layer group CO cc cc.ext
define physical layer group M1 cmf cmf.ext
define physical layer group M2 cms cms.ext
#
# Define Minimum Layer Widths:
#
minimum width PW 1
minimum width PD 4
minimum width PO 3
minimum width ND 4
minimum width CO 4
minimum width M1 3
minimum width M2 4
#
# Define Spacing Rules:
#
minimum spacing  cpg  cpg  3.0
minimum spacing  cpg  cap  4.0
minimum spacing  cpg  can  4.0
minimum spacing  cmf  cmf  3.0
minimum spacing  cms  cms  5.0
minimum spacing  can  can  4.0
minimum spacing  cap  cap  4.0
minimum spacing  cap  cwp  4.0
#
# attributes for layers
#
define layer attributes normal  cwp  fill_color yellow
    line_color yellow line_style long_dash line_width 1 pattern none
define layer attributes normal  cpg  fill_color red
    line_color red line_style solid line_width 1 pattern solid
define layer attributes normal  can  fill_color green
    line_color green line_style solid line_width 1 pattern solid
define layer attributes normal  cap  fill_color magenta
    line_color magenta line_style solid line_width 1 pattern solid
define layer attributes normal  cc  fill_color tan
    line_color tan line_style solid line_width 1 pattern solid

```

```

define layer attributes normal cmf fill_color blue
  line_color blue line_style solid line_width 1 pattern o
define layer attributes normal cms fill_color purple
  line_color purple line_style solid line_width 1 pattern m
#
define layer attributes normal cwp.ext fill_color yellow
  line_color yellow line_style long_dash line_width 1 pattern none
define layer attributes normal cpg.ext fill_color red
  line_color red line_style solid line_width 1 pattern none
define layer attributes normal can.ext fill_color green
  line_color green line_style solid line_width 1 pattern none
define layer attributes normal cap.ext fill_color magenta
  line_color magenta line_style solid line_width 1 pattern none
define layer attributes normal cc.ext fill_color tan
  line_color tan line_style solid line_width 1 pattern none
define layer attributes normal cmf.ext fill_color blue
  line_color blue line_style solid line_width 1 pattern none
define layer attributes normal cms.ext fill_color purple
  line_color purple line_style solid line_width 1 pattern m
#
define layer attributes selected cwp fill_color yellow
  line_color yellow line_style long_dash line_width 3
define layer attributes selected cpg fill_color red
  line_color red line_style solid line_width 3
define layer attributes selected can fill_color green
  line_color green line_style solid line_width 3
define layer attributes selected cap fill_color magenta
  line_color magenta line_style solid line_width 3
define layer attributes selected cc fill_color tan
  line_color tan line_style solid line_width 3
define layer attributes selected cmf fill_color blue
  line_color blue line_style solid line_width 3
define layer attributes selected cms fill_color purple
  line_color purple line_style solid line_width 3
#
define layer attributes selected cwp.ext fill_color yellow
  line_color yellow line_style long_dash line_width 3
define layer attributes selected cpg.ext fill_color red
  line_color red line_style solid line_width 3
define layer attributes selected can.ext fill_color green
  line_color green line_style solid line_width 3
define layer attributes selected cap.ext fill_color magenta
  line_color magenta line_style solid line_width 3
define layer attributes selected cc.ext fill_color tan
  line_color tan line_style solid
define layer attributes selected cmf.ext fill_color blue
  line_color blue line_style solid line_width 3
define layer attributes selected cms.ext fill_color purple
  line_color purple line_style solid line_width 3

```

```
#
# attributes for item classes
#
define item attributes normal instance    line_style solid pattern none
define item attributes normal pin        line_style solid pattern none
define item attributes normal port       line_style solid pattern none
define item attributes normal perimeter  line_style solid pattern none
define item attributes selected instance line_style solid pattern none
define item attributes selected pin       line_style solid pattern none
define item attributes selected port      line_style solid pattern none
define item attributes selected perimeter line_style solid pattern none
transcribing on
# Process definition loaded.
transcribing off
#read menu /idea/unrlsd/cds.emos_menu
```

```

#
## -----
## The CELL_MODEL file contains the models of PLA, ARRAY,
## nROM, and RAM which BUILD_UP uses to convert the shapes
## in CHIPGRAPH database to pins and blockages in the CELL
## station database. The model file is a set of statements
## which give layers and location data for the pins.
## -----
#
## Next, the Process Definition File (kludge2.bin) is
## loaded for PLA, ARRAY, and nROM models.
#
MACMODEL PROCESS=//s1/u/golnabi/memory_cells/kludge2.bin
#
## PLA model...
#
BEGIN MODEL NAME=CELLPLA
  BEGIN POWER NAME=VDD
    DEFINE LEVEL=1 LAYERS=28 FORM=PORTS
    PHYPROP=POWER.PIN DIRPROP=PINTYPE
  END POWER
  BEGIN POWER NAME=VSS
    DEFINE LEVEL=1 LAYERS=28 FORM=PORTS &
    PHYPROP=POWER.PIN DIRPROP=PINTYPE
  END POWER
  BEGIN SIGNAL
    DEFINE LEVEL=1 LAYERS=28 FORM=PORTS
    DEFINE LEVEL=2 LAYERS=32 FORM=PORTS &
    PHYPROP=PHY.PIN LOGPROP=PIN DIRPROP=PINTYPE
  END SIGNAL
  BEGIN BLOCKAGE
    DEFINE LEVEL=1 LAYERS=28 FORM=EACH_SHAPE
    DEFINE LEVEL=2 LAYERS=32 FORM=EACH_SHAPE
  END BLOCKAGE
END MODEL
#

```

```

## ARRAY model...
#
BEGIN MODEL NAME=CELLARRAY
  BEGIN POWER NAME=VDD
    DEFINE LEVEL=1 LAYERS=28 FORM=PORTS
    PHYPROP=POWER.PIN DIRPROP=PINTYPE
  END POWER
  BEGIN POWER NAME=VSS
    DEFINE LEVEL=1 LAYERS=28 FORM=PORTS &
    PHYPROP=POWER.PIN DIRPROP=PINTYPE
  END POWER
  BEGIN SIGNAL
    DEFINE LEVEL=1 LAYERS=28 FORM=PORTS
    DEFINE LEVEL=2 LAYERS=32 FORM=PORTS &
    PHYPROP=PHY.PIN LOGPROP=PIN DIRPROP=PINTYPE
  END SIGNAL
  BEGIN BLOCKAGE
    DEFINE LEVEL=1 LAYERS=28 FORM=EACH_SHAPE
    DEFINE LEVEL=2 LAYERS=32 FORM=EACH_SHAPE
  END BLOCKAGE
END MODEL
#
## nROM model...
#
BEGIN MODEL NAME=CELLNROM
  BEGIN POWER NAME=VDD
    DEFINE LEVEL=1 LAYERS=28 FORM=PORTS
    PHYPROP=POWER.PIN DIRPROP=PINTYPE
  END POWER
  BEGIN POWER NAME=VSS
    DEFINE LEVEL=1 LAYERS=28 FORM=PORTS &
    PHYPROP=POWER.PIN DIRPROP=PINTYPE
  END POWER
  BEGIN SIGNAL
    DEFINE LEVEL=1 LAYERS=28 FORM=PORTS
    DEFINE LEVEL=2 LAYERS=32 FORM=PORTS &
    PHYPROP=PHY.PIN LOGPROP=PIN DIRPROP=PINTYPE
  END SIGNAL
  BEGIN BLOCKAGE
    DEFINE LEVEL=1 LAYERS=28 FORM=EACH_SHAPE
    DEFINE LEVEL=2 LAYERS=32 FORM=EACH_SHAPE
  END BLOCKAGE
END MODEL
END MACMODEL
#
## The PDF file is loaded for RAM model.
#

```

```
MACMODEL PROCESS=//s1/u/golnabi/memory_cells/cmos.bin
BEGIN MODEL NAME=CELLRAM
BEGIN POWER NAME=VDD
  DEFINE LEVEL=1 LAYERS=28 FORM=PORTS
  PHYPROP=POWER.PIN DIRPROP=PINTYPE
END POWER
BEGIN POWER NAME=VSS
  DEFINE LEVEL=1 LAYERS=28 FORM=PORTS &
  PHYPROP=POWER.PIN DIRPROP=PINTYPE
END POWER
BEGIN SIGNAL
  DEFINE LEVEL=1 LAYERS=28 FORM=PORTS
  DEFINE LEVEL=2 LAYERS=32 FORM=PORTS &
  PHYPROP=PHY.PIN LOGPROP=PIN DIRPROP=PINTYPE
END SIGNAL
BEGIN BLOCKAGE
  DEFINE LEVEL=1 LAYERS=28 FORM=EACH_SHAPE
  DEFINE LEVEL=2 LAYERS=32 FORM=EACH_SHAPE
END BLOCKAGE
END MODEL
END MACMODEL
```



```

#
## .....
## The CELLLIST file contains a list of PLA, ARRAY, nROM,
## and RAM cells which BUILD_UP program converts from their
## CHIPGRAPH representation to the CELL station format.
## .....
#
CELLLIST
BEGIN PHYLIB LIBRARY=cell version=1 &
  SEARCH=/s1/u/golnabi/pla/memory_cells
  DEFINE CELLNAME=((PLA, 0.0, 0.0, N)) MODEL=CELLPLA &
    NAME=&PHY.COMP LOGNAME=&COMP &
    CLASS=&COMPTYPE PLACETYP=&PLACETYPE
  DEFINE CELLNAME=((ARRAY, 0.0, 0.0, N)) MODEL=CELLARRAY &
    NAME=&PHY.COMP LOGNAME=&COMP &
    CLASS=&COMPTYPE PLACETYP=&PLACETYPE
  DEFINE CELLNAME=((NROM, 0.0, 0.0, N)) MODEL=CELLNROM &
    NAME=&PHY.COMP LOGNAME=&COMP &
    CLASS=&COMPTYPE PLACETYP=&PLACETYPE
  DEFINE CELLNAME=((RAM, 0.0, 0.0, N)) MODEL=CELLRAM &
    NAME=&PHY.COMP LOGNAME=&COMP &
    CLASS=&COMPTYPE PLACETYP=&PLACETYPE
END PHYLIB
END CELLLIST

```

APPENDIX B

GATE LEVEL SIMULATION OF THE MICRO-CONTROL UNIT

```

# -----
# SIM Force statements for COMB_LOGIC Routine.
# -----
#
# Set clock period for 100NS, and look
# for a change at the output signals R(33:1)...
#
clock period 100
force PHI2 0 0 -R
force PHI2 1 50 -R
#
## Set inputs for running simulation...
#
check -nospike
force F 0002 0 ## Enable F(1) signal....
#
## Generate R(1) signal....
#
force CID1 0 0
force CID2 0 0
force DTAG1 3 0
force DTAG2 3 0
#
## Generate R(2) signal....
#
force CID1 1 100
force CID2 1 100
#
## Generate R(3) signal....
#
force CID1 2 200
force CID2 2 200
#
## Generate R(3) signal....
#
force F 0202 300 ## Enable F(9) signal...
force Z 0 300 ## Disable Z flag...
#
## The operation is succeeded....
#
force CID1 3 300
force CID2 3 300
#
## The operation is failed....
#
force DTAG2 1 400
force DTAG1 1 500
force DTAG2 3 500
#

```

```
force F 0001 600   ## Enable F(0) signal...
force DTAG1 2 600
force DTAG2 2 600
#
## Generate R(5) signal...
#
force DTAG2 3 700
#
## Generate R(6) signal...
#
force DTAG1 3 800
force DTAG2 2 800
#
## Generate R(5) signal...
#
force DTAG1 2 900
force DTAG2 1 900
#
## Generate R(6) signal...
#
force DTAG1 1 1000
force DTAG2 2 1000
#
## Generate R(7) signal...
#
force F 0000 1100  ## Initialize F(15:0) signals...
force DTAG2 1 1100
force DTAG2 0 1200
force F 0004 1200  ## Disable all signals but F(4) signal...
#
## Generate R(8) signal...
#
force MATCH 1 1200
force BMR 2 1200
#
## Generate R(9) signal...
#
force BMR 3 1300
force F 0008 1400  ## F(8) signal is active...
force MATCH 1 1400
#
## Generate R(12) signal...
#
force BMR 3 1400
#
## Generate R(11) signal...
#
force BMR 2 1500
force MATCH 0 1600
```

```
#
## Generate R(13) signal...
#
force F 0008 1600
#
## Generate R(10) signal...
#
force F 0004 1700   ## Enable F(6) signal...
#
## Generate R(14) signal...
#
force F 0010 1800   ## F(4) is logic 1
force DTAG1 2 1800
force DTAG2 2 1800
force BO1 0 1800
force BO2 0 1800
#
## Generate R(15) signal...
#
force BO1 1 1900
#
## Generate R(16) signal...
#
force BO1 0 2000
force BO2 1 2000
#
## Generate R(17) signal...
#
force BO1 1 2100
#
## Generate R(18) signal...
#
force DTAG1 3 2200
force BO2 0 2200
#
## Generate R(19) signal...
#
force BO2 1 2300
force DTAG1 1 2400
force BO2 0 2400   ## R(18) is generated...
force BO2 1 2500   ## R(19) is generated...
#
## REDY signal is at logic 1 and STOP signal is zero...
#
force ARITY 0 2600
force F 0420 2600   ## F(5) and F(10) are both active...
force SP 0 2600
#
## Generate R(20) signal...
```

```
#
force SR 0 2700
force SP 1 2600
#
## Generate R(21) signal...
#
force ARITY 1 2900
force SR 1 2900
#
## Generate R(22) signal...
#
force F 0040 3000  ## All the signals are disabled but F(6)...
force DTAG1 3 3000
force DTAG2 2 3000
#
## Generate R(23) signal...
#
force DTAG1 2 3100
#
## Generate R(24) signal...
#
force DTAG1 1 3200
#
## Generate R(25) signal...
#
force F 0080 3300  ## Enable F(7) signal...
force DTAG1 2 3300
force DTAG2 3 3300
#
## Generate R(26) signal...
#
force DTAG2 2 3400
#
## Generate R(27) signal...
#
force DTAG2 1 3500
force F 0200 3600
force ARITY 0 3600  ## The signal FAIL
force Z 1 3600  ## is activated...
#
## Generate R(30) signal...
#
force Z 0 3700
#
## Generate R(28) signal...
#
force F 0100 3800  ## F(8) is at logic 1
force BO1 0 3800
#
```

```

## Generate R(29) signal...
#
force BO1 1 3900
#
## R(32) and R(33) signals are generated...
#
force F 1000 4000
force BMR 2 4000
force F 2000 4100
#
## Specify the RUN time...
#
run 4300
#
## The quicksim_traces file contains the waveform traces...
#
plot trace comb_logic/quicksim_traces -replace
#
## The output file (quicksim_list) contains the logic information...
#
write list comb_logic/quicksim_list -replace
#
## Upon finish, Exit the QUICKSIM environment...
#
exit
#
#
#
#
# .....
# SIM Force statements for m1R and Insr_decode Routine.
# .....
#
## Set clock period for 100NS, and look
## for a change at the output signals F(15:0),
## ADDR(7:0), S(3:0), and CNRL(83:0)...
#
clock period 100
force PHI2BAR 0 0 -R
force PHI2BAR 1 50 -R
#
## Note that all the signals are represented
## as buses with their HEX values to make the
## the simulation task easier, i.e., Q1 is a 39 bit wide...
## The values of Q1 used here are the outputs from the PLA...
#
## Set inputs for running simulation...
#
check -nospike
force Q1 0 0

```

```

force QI 1678608000 100  ## PDSR(6:0)=01
force QI 0001164001 200  ## PDSR(6:0)=02
force QI 6000168000 300  ## PDSR(6:0)=03
force QI 7007000000 400  ## PDSR(6:0)=04
force QI 500220D001 500  ## PDSR(6:0)=05
force QI 0500008000 600  ## PDSR(6:0)=06
force QI 000000100 700  ## PDSR(6:0)=09
force QI 0800160062 800  ## PDSR(6:0)=0B
force QI 0100160062 900  ## PDSR(6:0)=0C
force QI 1980160062 1000 ## PDSR(6:0)=0E
force QI 3000004001 1100 ## PDSR(6:0)=0F
force QI 02AE00D055 1200 ## PDSR(6:0)=11
force QI 0028C08016 1300 ## PDSR(6:0)=12
force QI 3040C38000 1400 ## PDSR(6:0)=13
force QI 0012000020 1500 ## PDSR(6:0)=14
force QI 487041C0A0 1600 ## PDSR(6:0)=15
force QI 0020607500 1700 ## PDSR(6:0)=16
force QI 487041C0E4 1800 ## PDSR(6:0)=18
force QI 002D000740 1900 ## PDSR(6:0)=1A
force QI 026901C080 2000 ## PDSR(6:0)=1B
force QI 0022007500 2100 ## PDSR(6:0)=1C
force QI 5505D80062 2200 ## PDSR(6:0)=1D
force QI 3000004001 2300 ## PDSR(6:0)=1E
force QI 002E008054 2400 ## PDSR(6:0)=1F
force QI 0000000800 2500 ## PDSR(6:0)=20
force QI 582D200740 2600 ## PDSR(6:0)=21
force QI 482101CBE6 2700 ## PDSR(6:0)=22
force QI 5205160062 2800 ## PDSR(6:0)=23
force QI 0280005001 2900 ## PDSR(6:0)=24
force QI 0026E08016 3000 ## PDSR(6:0)=25
force QI 0001DB8640 3100 ## PDSR(6:0)=26
force QI 0001DB8640 3200 ## PDSR(6:0)=27
force QI 0001DB8640 3300 ## PDSR(6:0)=28
force QI 020041C0E4 3400 ## PDSR(6:0)=2A
force QI 2D00C40062 3500 ## PDSR(6:0)=2B
force QI 2380160000 3600 ## PDSR(6:0)=2C
force QI 0000000900 3700 ## PDSR(6:0)=2D
force QI 6880000050 3800 ## PDSR(6:0)=2E
force QI 0061600050 3900 ## PDSR(6:0)=2F
force QI 0060400050 4000 ## PDSR(6:0)=30
force QI 3B040E8000 4100 ## PDSR(6:0)=31
force QI 7307A00068 4200 ## PDSR(6:0)=32
force QI 0750000000 4300 ## PDSR(6:0)=33
force QI 0058800072 4400 ## PDSR(6:0)=34
force QI 005B000072 4500 ## PDSR(6:0)=35
force QI 7058000A40 4600 ## PDSR(6:0)=36
force QI 0002600500 4700 ## PDSR(6:0)=37
force QI 003820004A 4800 ## PDSR(6:0)=38
force QI 023800004A 4900 ## PDSR(6:0)=3A

```



```

force QI 0438000000 5000 ## PDSR(6:0)=3B
force QI 0010600020 5100 ## PDSR(6:0)=3C
force QI 003100006C 5200 ## PDSR(6:0)=3D
force QI 303000004C 5300 ## PDSR(6:0)=3E
force QI 4038000000 5400 ## PDSR(6:0)=3F
force QI 0012000020 5500 ## PDSR(6:0)=40
force QI 1F98000020 5600 ## PDSR(6:0)=41
force QI 1818E0D001 5700 ## PDSR(6:0)=42
force QI 002E008000 5800 ## PDSR(6:0)=43
force QI 0000030C00 5900 ## PDSR(6:0)=44
force QI 00080EA000 6000 ## PDSR(6:0)=45
force QI 0007003020 6100 ## PDSR(6:0)=46
force QI 0007003400 6200 ## PDSR(6:0)=47
force QI 0498E0D001 6300 ## PDSR(6:0)=48
force QI 0028E08000 6400 ## PDSR(6:0)=4A
force QI 00080EA000 6500 ## PDSR(6:0)=4B
force QI 0007003400 6600 ## PDSR(6:0)=4C
force QI 0640000020 6700 ## PDSR(6:0)=4D
force QI 0750000020 6800 ## PDSR(6:0)=4E
force QI 0080030300 6900 ## PDSR(6:0)=4F
#
## Specify the RUN time...
#
run 7000NS
#
## The waveform traces are located on quicksim_traces file...
#
plot trace comb_logic/quicksim_traces -replace
#
## The output file containing the logic information...
#
write list comb_logic/quicksim_list -replace
#
## Upon finish, Exit the QUICKSIM environment...
#
exit
#
#-----
# SIM Force statements for mPCI and mPCO Routine.
#-----
#
## Set clock period for 100NS.
## PH11 is used for counting whereas PH12
## is used for loading the register contents
## into mPCO...
#
clock period 100
force PH11 1 0 -REPEAT
force PH11 0 40 -REPEAT

```

```

force PHI2 0 0 -REPEAT
force PHI2 1 50 -REPEAT
force PHI2 0 90 -REPEAT
#
## Set inputs for running simulation...
#
check -nospike
force REDY 0 0    ## set up the loading sequence...
force STOP 0 0   ## Initialize the register contents...
force STOP 1 30  ##
#
## The registers are loaded if one of
## the four signals is enabled...
#
force R_G 0
force S(1) 1     ## the registers are loaded
force DATA 0
force P_C 0
#
## stimulate the inputs PDSR(6:0) and look
## for a change at the outputs PC(6:0)....
#
force PDSR 02 0
force PDSR 1E 100
force PDSR 20 200
force PDSR 2C 300
force PDSR 3B 400
#
## set up the counting sequence
#
force REDY 1 100
#
## The counter state is now equal
## to the previous register contents plus one...
#
force P_C 1 400 ## the new register contents are loaded into mPCO...
#
## Specify the RUN time...
#
run 500
#
## The quicksim_traces file contains the waveform traces....
#
plot trace mPCI/quicksim_traces -replace
#
## The output file (quicksim_list) contains the logic information...
#
write list mPCI/quicksim_list -replace
#

```

```

## Upon finish, Exit the QUICKSIM environment...
#
exit
#
# -----
# SIM Force statements for mSP Routine.
# -----
#
## Set clock period for 100NS, and look
## for counting sequences of the outputs ADDR(2:0)
#
clock period 100
force PHI 0 0 -R
force PHI 1 50 -R
#
## Set inputs for running simulation...
## S(2:1)=1 will enable the counter...
check -nospike
force STOP 0 0 ## initialize the counter....
force STOP 1 30 ## set up the counting condition....
#
## Counting down sequence...
#
force S 1 100
force S 1 200 ## ADDR(2:0)=7
force S 1 300
force S 1 400
force S 1 500
force S 1 600
force S 1 700
force S 1 800
force S 1 900 ## ADDR(2:0)=0
#
## Counting up sequence...
#
force S 2 1000 ## ADDR(2:0)=1
force S 2 1100
force S 2 1200
force S 2 1300
force S 2 1400
force S 2 1500
force S 2 1600 ## ADDR(2:0)=7
#
## Specify the RUN time...
#
run 1700NS
#
## The waveform traces are located on quicksim_traces file....
#

```

```

plot trace mSP/quickstim.traces -replace
#
## The output file containing the logic information...
#
write list mSP/quickstim.list -replace
#
## Upon finish, Exit the QUICKSIM environment...
#
exit
#
# -----
# SIM Force statements for Address Decoder
# portion of the nROM.
# -----
#
# Set clock period for 100NS, and look
# for a change at the output signals IN(15:0)...
#
clock period 100
force PH11 0 0 -R
force PH11 1 50 -R
#
## Set inputs for running simulation.
## Inputs are Q(4:1)...
check -nospike
force Q 0 0    ## IN(15:0)=0001
force Q 1 30   ## IN(15:0)=0002
force Q 2 60   ## IN(15:0)=0004
force Q 3 90   ## IN(15:0)=0008
force Q 4 120  ## IN(15:0)=0010
force Q 5 150  ## IN(15:0)=0020
force Q 6 180  ## IN(15:0)=0040
force Q 7 210  ## IN(15:0)=0080
force Q 8 240  ## IN(15:0)=0100
force Q 9 270  ## IN(15:0)=0200
force Q A 300  ## IN(15:0)=0400
force Q B 330  ## IN(15:0)=0800
force Q C 360  ## IN(15:0)=1000
force Q D 390  ## IN(15:0)=2000
force Q E 420  ## IN(15:0)=4000
force Q F 450  ## IN(15:0)=8000
#
## Specify the RUN time...
#
run 460
#
## The quickstim.traces file contains the waveform traces....
#
plot trace mIR/quickstim.traces -replace

```

```

#
## The output file (quicksim.list) contains the logic information...
#
write list mIR/quicksim.list -replace
#
## Upon finish, Exit the QUICKSIM environment...
#
exit
#
# -----
# SIM Force statements for register DATA.
# -----
#
# Set clock period for 100NS, and look
# for a change at the output signals DT(6:0)...
#
clock period 100
force PHI 0 0 -R
force PHI 1 50 -R
#
## Set inputs for running simulation.
## Inputs are DTI(6:0)...
#
check -nospike
force DATA 0 0 ## Enable the register DATA...
force DTI 00 0 ## DT(6:0)=00
force DTI 0E 100 ## DT(6:0)=0E
force DTI 23 200 ## DT(6:0)=23
#
## Specify the RUN time...
#
run 250NS
#
## The quicksim.traces file contains the waveform traces....
#
plot trace DATA/quicksim.traces -replace
#
## The output file (quicksim.list) contains the logic information...
#
write list DATA/quicksim.list -replace
#
## Upon finish, Exit the QUICKSIM environment...
#
exit
#
# -----
# SIM Force statements for register REG.
# -----

```

```

#
# Set clock period for 100NS, and look
# for a change at the output signals REG(6:0)...
#
clock period 100
force PHI1 0 0 -R
force PHI1 1 50 -R
#
## Set inputs for running simulation.
## Inputs are REGI(6:0)...
#
check -nospike
force R_G 0 0    ## Enable the register REG...
force REGI 10 0  ## REG(6:0)=10
force REGI 2B 100 ## REG(6:0)=2B
force REGI 3C 200 ## REG(6:0)=3C
force REGI 43 300 ## REG(6:0)=43
#
## Specify the RUN time...
#
run 350NS
#
## The quicksim.traces file contains the waveform traces....
#
plot trace REG/quicksim.traces -replace
#
## The output file (quicksim.list) contains the logic information...
#
write list REG/quicksim.list -replace
#
## Upon finish, Exit the QUICKSIM environment...
#
exit
#
#
# -----
# SIM Force statements for MUX.
# -----
#
## set inputs for simulation.
## The S(3:2), S(0), A(0), and F(0) signals are the inputs.
## The outputs from the MUX are P.C, R.G, and DATA signals....
#
check -nospike
force F(0) 0 20
force S(0) 0 20  ## P.C signal is active...
force A(0) 1 50
force S(3:2) 3 50 ## R.G signal is active...
force S(3:2) 2 80 ## DATA signal is active...

```

```
#
## Specify the RUN time...
#
run 100NS
#
## The quicksim.traces file contains the waveform traces....
#
plot trace MUX/quicksim.traces -replace
#
## The output file (quicksim.list) contains the logic information...
#
write list MUX/quicksim.list -replace
#
## Upon finish, Exit the QUICKSIM environment...
#
exit
```

APPENDIX C

THE NROM, ARRAY, AND PLA MEMORY ARRAYS

The Memory Cell Array of the nROM

```
[ 01] DTI(6:0)=01
      0 0 0 0 0 0 1
[ 02] DTI(6:0)=02
      0 0 0 0 0 1 0
[ 03] DTI(6:0)=03
      0 0 0 0 0 1 1
[ 04] DTI(6:0)=04
      0 0 0 0 1 0 0
[ 05] DTI(6:0)=05
      0 0 0 0 1 0 1
[ 06] DTI(6:0)=06
      0 0 0 0 1 1 0
[ 07] DTI(6:0)=07
      0 0 0 0 1 1 1
[ 08] DTI(6:0)=08
      0 0 0 0 1 1 1
[ 09] DTI(6:0)=09
      0 0 0 1 0 0 1
[ 0A] DTI(6:0)=10
      0 0 0 1 0 1 0
[ 0B] DTI(6:0)=11
      0 0 0 1 0 1 1
[ 0C] DTI(6:0)=12
      0 0 0 1 1 0 0
[ 0D] DTI(6:0)=13
      0 0 0 1 1 0 1
[ 0E] DTI(6:0)=14
      0 0 0 1 1 1 0
[ 0F] DTI(6:0)=15
      0 0 0 1 1 1 0
[ 10] DTI(6:0)=16
      0 0 0 1 1 1 1
```

The Mapping Table of the Array

```
[ 01] reg=07
      0 0 0 0 1 1 1
[ 02] reg=08
      0 0 0 1 0 0 0
[ 03] reg=09
      0 0 0 1 0 0 1
[ 04] reg=10
      0 0 0 1 0 1 0
[ 05] reg=23
      0 0 1 0 1 1 1
[ 06] reg=29
      0 0 1 1 1 0 1
[ 07] reg=34
      0 1 0 0 0 1 0
[ 08] reg=54
      0 1 1 0 1 1 0
[ 09] reg=55
      0 1 1 0 1 1 1
[ 0A] reg=58
      0 1 1 1 0 1 0
[ 0B] reg=60
      0 1 1 1 1 0 0
[ 0C] reg=61
      0 1 1 1 1 0 1
[ 0D] reg=64
      1 0 0 0 0 0 0
[ 0E] reg=13
      0 0 0 1 1 0 1
[ 0F] reg=17
      0 0 1 0 0 0 1
[ 10] reg=19
      0 0 1 0 0 1 1
[ 11] reg=22
      0 0 1 0 1 1 0
[ 12] reg=31
      0 0 1 1 1 1 1
[ 13] reg=33
      0 1 0 0 0 0 1
[ 14] reg=41
      0 1 0 1 0 0 1
```

The Mapping Table of the Array (continued)

```
[ 15] reg=01
      0 0 0 0 0 0 1
[ 16] reg=50
      0 1 1 0 0 1 0
[ 17] reg=51
      0 1 1 0 0 1 1
[ 18] reg=52
      0 1 1 0 1 0 0
[ 19] reg=46
      0 0 1 1 0 1 0
[ 1A] reg=47
      0 1 0 1 1 1 1
[ 1B] reg=48
      0 1 1 0 0 0 0
[ 1C] reg=26
      0 0 1 1 0 1 0
[ 1D] reg=28
      0 0 1 1 1 0 0
[ 1E] reg=39
      0 1 0 0 1 1 1
[ 1F] reg=66
      1 0 0 0 0 0 1
[ 20] reg=67
      1 0 0 0 0 1 1
[ 21] reg=71
      1 0 0 0 1 1 1
```


The Control Store Using PLA (continued)

```

[ 2A] pop2
0 0 0 0 0 0 0 0 1 0 1 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 1 1

[ 2B] pop3
0 0 0 0 0 0 0 0 1 0 1 1 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 1 1

[ 2C] pop4
1 1 1 0 0 0 0 0 1 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 1 0 0 0 0 0 1

[ 2D] incmar1
0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 1 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 1

[ 2E] setcambit11
0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 0 1 1

[ 2F] setcambit12
0 0 0 0 0 1 0 1 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 0 1 1

[ 30] setcambit13
0 0 0 0 1 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1

[ 31] setcambit14
0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1

[ 32] setcambit21
0 0 0 0 0 0 0 0 0 1 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 1 0 1

[ 33] setcambit22
0 1 1 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 1 0 1

[ 34] setcambit23
1 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1

[ 35] setcambit24
0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1

[ 36] prebnd1
1 0 1 1 1 1 1 1 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1

[ 37] tempbnd11
1 0 1 1 0 0 0 0 0 0 1 1 0 0 0 1 1 1 0 0 0 0 0 1 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0

[ 38] tempbnd12
0 0 0 0 0 0 0 0 0 1 0 1 1 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1

[ 39] tempbound13
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 1 0 0 0 0 1 0 0 0 0 1

[ 3A] addent11
0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 1 1 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1

[ 3B] addent12
0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 1 0 0 0 0 1

[ 3C] prebound2
0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 1 1 0 1 0 0 0 0 0 0 0 0 0 1

[ 3D] tempbound21
0 0 0 0 1 0 0 1 0 0 1 1 0 0 0 1 1 1 0 0 0 0 0 1 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0

[ 3E] tempbound22
0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 1 1 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1

```


VITA

The author of this thesis, Habibollah Golnabi, was born on September 16, 1959 in Tehran, Iran. He received the B.S. and M.S. degrees in Electrical Engineering from Texas A&M University in 1985 and 1988, respectively. His areas of interest include the microprocessor system and memory designs. The author can be contacted at 11006 Listi Drive, Dallas, Texas 75238.