1

**PM-1 3½"x4" PHOTOGRAPHIC MICROCOPY TARGET
NBS 1010a ANSI/ISO #2 EQUIVALENT**

PRECISION^{SM} RESOLUTION TARGETS

PIONEERS IN METHYLENE BLUE TESTING SINCE 1974

Microlith

13800 COUNTY ROAD 8, BURNINGWELL, WY 82407 USA
TEL: 812 436 7887 FAX: 812 436 7887 TLE: 8124367888

National Library
of Canada

Bibliothèque nationale
du Canada

Canadian Theses Service

Service des thèses canadiennes

Ottawa, Canada
K1A 0N4

NOTICE

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Reproduction in full or in part of this microform is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30, and subsequent amendments.

AVIS

La qualité de cette microforme dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

La reproduction, même partielle, de cette microforme est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30, et ses amendements subséquents.

APPLICATION SPECIFIC INTEGRATED CIRCUIT (ASIC)

HARDWIRED MICROCONTROLLER

By

THAI Q. LE, B. Eng.

This report is a thesis submitted to the
Faculty of Graduate Studies and Research
in partial fulfilment of requirements
for the degree of Master of Engineering

Faculty of Engineering
Department of Electronics
Carleton University

December 1991

Copyright
1991, Thai Q. Le

National Library
of Canada

Bibliothèque nationale
du Canada

Canadian Theses Service   Service des thèses canadiennes

Ottawa, Canada
K1A 0N4

The author has granted an irrevocable non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of his/her thesis by any means and in any form or format, making this thesis available to interested persons.

The author retains ownership of the copyright in his/her thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without his/her permission.

L'auteur a accordé une licence irrévocable et non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de sa thèse de quelque manière et sous quelque forme que ce soit pour mettre des exemplaires de cette thèse à la disposition des personnes intéressées.

L'auteur conserve la propriété du droit d'auteur qui protège sa thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

ISBN 0-315-75978-X

Canada

Thesis contains black & white
~~photographs &/or~~ explanatory tables
which when microfilmed may lose their
significance. The hardcopy of the
thesis is available upon request from
Carleton University Library.

The undersigned recommends to the Faculty of
Graduate Studies and Research
acceptance of the thesis

**APPLICATION SPECIFIC INTEGRATED CIRCUIT
HARDWIRED MICROCONTROLLER**

submitted by Thai Q. Le, B. Eng.
in partial fulfilment of the requirements for
the degree of Master of Engineering

A handwritten signature in dark ink, appearing to read 'John Knight', is written over a horizontal line.

Prof. J.P. Knight

✓ Thesis Supervisor

A handwritten signature in dark ink, appearing to read 'J.S. Wight', is written over a horizontal line.

Prof. J.S. Wight

Chair, Department of Electronics

Carleton University

December 1991

ABSTRACT

This thesis presents a new architecture for Application Specific Integrated Circuit (ASIC) microcontrollers using hardwired microprogram sequencers. The main objectives of this architecture are to improve the speed and to reduce the area of ASIC microcontrollers. Shift registers are employed to drive word lines and bit lines of the microprogram memory instead of the conventional n -to- 2^n address decoders. Wire connections and discrete logic are used to implement microprogram branches and subroutine calls. Address lines and address supporting logic are eliminated. With this architecture, the microinstruction fetching time is significantly reduced. The microinstruction is narrower and the microprogram memory is much smaller.

ACKNOWLEDGEMENTS

I would like to thank Dr. John P. Knight, my supervisor, for his guidance and his contribution to my work, Mr. Yatish Kumar for his valuable suggestions, and my family for their continual encouragement and support

TABLE OF CONTENTS

	<u>Page</u>
1. Introduction	1
1.1 Conventional Microcontroller - An Overview	1
1.2 Hardwired Microcontroller	6
1.2.1 An Overview	6
1.2.2 The Incentives	9
1.3 Thesis Overview	10
 2. Microprogramming Issues	 12
2.1 Microinstructions	12
2.1.1 Non-encoded Format	13
2.1.2 Encoded Format	13
2.1.3 Basic Microinstruction Set	14
2.2 Microinstruction Sequencing	16
2.2.1 Serial/Non-pipelined Microinstruction Sequencing ..	16
2.2.2 Parallel/Pipelined Microinstruction Sequencing	16
2.2.3 Serial-Parallel Microinstruction Sequencing	17
 3. Conventional ROMs	 18
3.1 ROM Basic Architecture	18
3.1.1 Address Decoder	20
3.1.2 Memory Array	21
3.1.3 Precharge Circuit	21
3.1.4 Sense Amplifier	24
3.2 X-Y Memory Array	24
3.3 Virtual Ground Memory Array	26
 4. Hardwired Microcontroller	 28
4.1 Sequential Operation	31
4.2 Dynamic Shift Register Row/Column Decoders	35
4.3 Clock Generator	40
4.4 Branches Implementation	42

TABLE OF CONTENTS

	<u>Page</u>
4.5 Subroutine Calls Implementation	47
4.5.1 Call Block	47
4.5.2 Return Block	51
4.5.3 Subroutine Call Example	53
 5. Hardwired Microcontroller, An Implementation Example	 56
5.1 Design Specifications	56
5.2 Logic Model	58
5.2.1 Microprogram Memory	58
5.2.2 Address Lookup Tables	61
5.2.3 Row/Column Shift Registers with Built-In Sequencing Logic	 64
5.2.4 Hardwired Microcontroller	68
5.3 Layout Example	70
5.3.1 Technology	70
5.3.2 1K X 8 Bit Hardwired Microcontroller	70
5.3.2.1 Hardwired Microcontroller Cells	70
5.3.2.2 Layout of the Hardwired Microcontroller	71
 6. Hardwired Microcontroller, Design Considerations	 76
6.1 Hardwired Microprogram Memory Realization	76
6.2 Length of the Row and Column Shift Registers	76
6.3 Branches versus Subroutine Calls	78
6.4 Encoded Wire Connections	79
6.5 Mask Programming Considerations	80
6.6 Testability	82
6.7 Distributed Hardwired Microcontrollers	84
6.8 Design Flexibility	85
 7. Hardwired Microcontroller vs. Conventional Microcontroller	 86
7.1 Area	86

TABLE OF CONTENTS

	<u>Page</u>
7.1.1 Hardwired Microcontroller	86
7.1.1.1 A Hardwired Microcontroller with 1K of Microinstructions and 6 Output Control Bits	87
7.1.1.2 A Hardwired Microcontroller with 2K of Microinstructions and 18 Output Control Bits	88
7.1.1.3 A Hardwired Microcontroller with N Microinstructions and m Output Control Bits	89
7.1.2 Conventional 1K X 8 Bit Virtual Ground ROM	90
7.1.3 Conventional Microcontroller with Non-Encoded Microinstructions	92
7.1.3.1 Conv. Microcontroller with 1K of Non-encoded Microinstructions and 6 Output Control Bits	93
7.1.3.2 Conv. Microcontroller with 2K of Non-encoded Microinstructions and 18 Output Control Bits	94
7.1.3.3 Conv. Microcontroller with N Non-encoded Microinstructions and m Output Control Bits	94
7.1.4 Conventional Microcontroller with Encoded Microinstructions	95
7.1.4.1 Conv. Microcontroller with 1.1K of Encoded Microinstructions and 6 Output Control Bits	95
7.1.4.2 Conv. Microcontroller with 2.2K of Encoded Microinstructions and 18 Output Control Bits	96
7.1.4.3 Conv. Microcontroller with 1.1N Encoded Microinstructions and m Output Control Bits	97
7.1.5 Area Comparison Summary	98
7.2 Performance	102
7.2.1 Non-pipelined Microcontrollers	102
7.2.2 Pipelined Microcontrollers	106
7.2.3 Performance Comparison Summary	110

TABLE OF CONTENTS

	Page
8. Conclusions and Future Work	111
8.1 Conclusions	111
8.1.1 Thesis Summary	111
8.1.2 Advantages of Hardwired Microcontrollers	111
8.1.3 Disadvantages of Hardwired Microcontrollers	113
8.2 Future Work	114
8.2.1 Testability for Hardwired Microcontrollers	114
8.2.2 Distributed Hardwired Microcontrollers	114
8.2.3 'Gate Array Like' Generic Hardwired Microcontroller	115
8.2.4 Hardwired Microcontroller IC Synthesis	115
8.2.5 Linear Hardwired Microcontrollers	116
9. References	117
 Appendix A Layout Example of a 1K X 8 Bit ROM	 121
Appendix B Layout Example of a 1K X 8 bit Hardwired Microcontroller	147
Appendix C Layout of the 3 um CMC Inverter Standard Cell, and the Scaled Down Inverter, NAND, NOR Standard Cells	 172
Appendix D Area Calculations for Conventional Microcontrollers	178

LIST OF TABLES

<u>Table</u>	<u>Description</u>	<u>Page</u>
2-1	The functional description of the Signetic 8X02 microprogram sequencer instruction set	15
5-1	Hardwired microcontroller, an example microprogram	56
5-2	Column address lookup table	62
5-3	Row address lookup table	63
7-1	Summary of the areas taken by different microcontrollers given the same number of non-encoded microcodes and the same number of output control bits	98
7-2	Areas taken by different microcontrollers given the same number of non-encoded microinstructions, N, and the same number of output control bits, m	99
7-3	Increasing order of the area taken by different microcontrollers	100
7-4	Microinstruction execution sequence of a non-pipelined hardwired microcontroller and a non-pipelined conv. microcontroller	103
7-5	Microinstruction execution sequence of a pipelined hardwired microcontroller and a pipelined conv. microcontroller.....	107

LIST OF FIGURES

Figure Description	Page
1-1 A general block diagram of μ microcontroller	2
1-2 A more detailed block diagram of a conv. microcontroller	4
1-3 Block diagram of a hardwired microcontroller	6
1-4 A more detailed block diagram of a hardwired microcontroller ..	8
2-1 Basic microinstruction format	12
3-1 Block diagram of a linear ROM	19
3-2 An example of an address decoder, 4-to-16 decoder	20
3-3 An example of a static memory array	22
3-4 An example of a dynamic memory array	23
3-5 Block diagram of a dynamic X-Y ROM	25
3-6 Virtual ground memory array	27
4-1 Block diagram of a 1K X 8 bit hardwired microcontroller	28
4-2 A more detailed block diagram of a hardwired microcontroller ..	29
4-3 The simplest form of hardwired microcontrollers	31
4-4 Sequence of execution of a sequential hardwired microcontroller	32
4-5 Sequence of execution of a hardwired microcontroller where the number of stage. of the row decoder is divisible by the number of stages of the column decoder	33
4-6 Dynamic shift register row/column decoder	35
4-7 Timing requirements for a hardwired microcontroller	37
4-8 An alternate circuit for the dynamic shift registers	39
4-9 The clock generator for a hardwired microcontroller	40
4-10 A simple presentation of the branch block	42
4-11 An example of how branches are implemented in a hardwired microcontroller	44
4-12 Hardwired microcontroller branch block, schematic, truth table	46
4-13 A simple presentation of the call block	48
4-14 Hardwired microcontroller call block, schematic and truth table	50
4-15 A simple presentation of the return block	51
4-16 Hardwired microcontroller return block, schematic, truth table.	52

LIST OF FIGURES

<u>Figure Description</u>	<u>Page</u>
4-17 Hardwired microcontroller subroutine calls implementation, an example	54
5-1 A 20 X 16 X 2 bit virtual ground dynamic memory cell array	59
5-2 Block diagram of the virtual ground dynamic memory cell array used for the 1K X 8 bit microprogram memory	60
5-3 16-bit hardwired column shift register	66
5-4 65-bit hardwired row shift register	67
5-5 Block diagram of the 1K X 8 bit hardwired microcontroller	68
5-6 Block diagram of a large hardwired microcontroller	69
5-7 Layout of the hardwired 16-bit column shift register	72
5-8 Layout of the hardwired 65-bit row shift register	73
5-9 Layout of the 1K X 8 bit hardwired microcontroller	74
6-1 Layout of a memory cell array	77
6-2 Wires can be encoded to save space	79
6-3 A blank hardwired microprogram address decoder	81
6-4 A simple built-in self test hardwired microcontroller	83
6-5 A simple example of a distributed hardwired microcontroller ...	84
7-1 Area partition of a hardwired microcontroller	86
7-2 Layout of a 1K X 8 bit conventional ROM	91
7-3 Area partition of a conventional microcontroller	92
7-4 Block diagram of a non-pipelined conventional microcontroller..	102
7-5 Block diagram of a non-pipelined hardwired microcontroller	102
7-6 Block diagram of a pipelined conventional microcontroller	106
7-7 Block diagram of a pipelined hardwired microcontroller	106

LIST OF SYMBOLS AND ABBREVIATIONS

*	Denotes an active low signal.
ADDR	Address.
ASIC	Application specific integrated circuit.
B	Branch flag, hardwired connection to the hardwired microcontroller status flag BRANCH.
BIST	Built-in self test.
BLT	8X02 branch to loop microinstruction.
BRANCH	Hardwired microcontroller status flag indicating if the present microinstruction is a conditional branch.
BRT	8X02 branch on test condition true microinstruction.
BSR	8X02 branch to subroutine if test condition is true microinstruction.
CALL	Hardwired microcontroller status flag indicating if the present microinstruction is a subroutine call.
CMC	Canadian Microelectronics Corporation.
CMOS	Complementary metal oxide semiconductor.
CMOS3 DLM	3-micron P-well CMOS process with single-level polysilicon and double-level metal.
CP	Crosspoint
EEPROM	Electrically erasable/programmable read-only memory.
EN	Enable, hardwired connection to the output of the current stage of the 'other' shift register.
EN+2	Hardwired connection to the output of the return stage of the other shift register.
EPROM	Erasable/programmable read-only memory.
FK Scheme	Fujiwara and Kinoshita easily testable PLA scheme.
FSM	Finite state machine.
GICIS	Guide to the Integrated Circuit Implementation Services.
GND	Ground.
HO Scheme	Hong and Ostapko easily testable PLA scheme.
INC	8X02 increment microinstruction.
K Scheme	Khakbaz easily testable PLA scheme.

LIST OF SYMBOLS AND ABBREVIATIONS

LFSR	Linear feedback shift register.
Log_2	Logarithm to base 2.
LSSD	Level sensitive scan design.
LSI	Large scale integrated circuit.
Microcode	Microprogram instruction or microinstruction.
mm	Millimetre.
MOD	Modulus operator.
MOS	Metal oxide semiconductor
MUX	Multiplexer.
N	Hardwired connection to the output of the current shift register stage.
N+1	Hardwired connection to the input of the subsequent shift register stage.
N+2	Hardwired connection to the output of the subsequent shift register stage.
N+B	Hardwired connection to the input of the 'branch destination' shift register stage.
NMOS	N-channel metal oxide semiconductor.
OE	Output enable.
OP-CODE	Operation code.
PLA	Programmable logic array.
PLP	8X02 push for looping microinstruction.
PMOS	P-channel metal oxide semiconductor.
POP	8X02 pop stack microinstruction.
PROM	Programmable read-only memory.
RAM	Random access memory.
RET	Return, hardwired connection to the hardwired microcontroller status flag RETURN.
RET_C*	Hardwired return path for a subroutine.
RETURN	Hardwired microcontroller status flag indicating if the present microinstruction is a subroutine return.
ROM	Read-only memory.

LIST OF SYMBOLS AND ABBREVIATIONS

RST	8X02 reset to zero microinstruction.
SKF Scheme	Saluja, Kinoshita and Fujiwara easily testable PLA scheme.
SR Flip-flop	Set-Reset flip-flop.
TSK	8X02 test and skip microinstruction
TTL	Transistor-transistor logic
um	Micrometer or micron.
Vdd	Positive voltage supply rail.
VLSI	Very large scale integrated circuit.
X-Y ROM	ROM arranged as a two-dimensional memory array.
XROM	ROM with the active area patterns laid out in an 'X' shape .

CHAPTER 1

1. INTRODUCTION

Microcontrollers are a common method for implementing a large finite state machine (FSM). It was first suggested by M. Wilkes in 1951 [TanA76] as an alternative approach to controlling gate sequences for main-frame computers. It allowed a computer to have several sets of machine instructions. Implementing a new set of machine instructions or modifying existing instruction sets could be done easily without modifying the hardware of the computer. Only the microprogram needed to be modified. Later, microcontrollers have been widely used for digitally controlled systems. Examples of such systems are vending machines, elevator controllers, programmable thermostats etc.

The major advantage of microcontrollers is that they are user programmable, easy to implement, easy and inexpensive to modify. However, they are generally much larger and slower than random logic. These are the two major incentives of the hardwired microcontroller architecture to be presented in this thesis: to increase the speed and to reduce the area of microcontrollers.

1.1 CONVENTIONAL MICROCONTROLLER - AN OVERVIEW

Microcontrollers are large finite state machines designed to control complex digital systems. These finite state machines usually have a large number of states and a large number of input and output control signals. Generally, the output control signals are used to control different hardware functions of the digital system. These signals are generated by the microcontroller based on the system input signals and the preprogrammed decoding logic stored in the microprogram.

Figure 1-1 on next page shows a general block diagram of a conventional microcontroller. There are basically four major components in a

microcontroller: the input and output interfaces, the microprogram memory, and the microprogram sequencer.

Figure 1-1 A general block diagram of a microcontroller.

The input and output interfaces usually provide holding registers and necessary conversion mechanisms for the input and output signals. The holding registers provide a means to capture and hold the input signals until the microcontroller is ready to process the input data, and to hold the output data for the controlled system without tying up the microcontroller. The data conversion mechanism is required only when the signals from the controlled system are not compatible with those from the microcontroller.

The microprogram memory is usually implemented using Read Only Memory (ROM), or Programmable Read Only Memory (PROM), or Programmable Logic Array (PLA). A typical microprogram memory size varies between 256 to 4K words of 16 to 64 bits. The microprogram memory is the storage for the microprogram which specifies the logic for the finite state machine. The logic can be divided into two types: 'next state' logic and output logic. Instructions of the microprogram are known as microinstructions. Each microinstruction generally consists of three fields. The first two fields, Operation Code (OP-CODE) and BRANCH ADDRESS, specify the 'next state' logic. The third field OUTPUT CONTROL specifies the output logic. The microprogram can be seen as a large lookup table. For every combination of input and present state, there is a microinstruction specifying the next state and the appropriate output sequence.

The microprogram sequencer sequences the microcontroller according to the preprogrammed logic specified by the microprogram and the system input.

The block diagram of a conventional microcontroller can be broken down further as shown in figure 1-2 on next page. The system input and output interfaces are not shown to simplify the diagram.

Figure 1-2 A more detailed block diagram of a conventional microcontroller with the microprogram sequencer and the microprogram memory expanded. The input and output interfaces are not shown.

The microprogram memory shown is a typical Read Only Memory (ROM) which consists of two address decoders (row and column), a memory cell array, and an output latch (data register). For every address input from the microprogram sequencer, the microprogram memory generates a corresponding microinstruction which specifies the output control signals (SYSTEM OUTPUT), and the OP-CODE and the BRANCH ADDRESS for the microprogram sequencer. The input address for the microprogram memory is generated by the microprogram sequencer based on the system input conditions, SYSTEM FLAGS and DIRECT INPUT ADDRESS, and the 'next state' feedback, OP-CODE and BRANCH ADDRESS, from microprogram memory. SYSTEM FLAGS generally are the controlled system status flags. The DIRECT INPUT ADDRESS, is sometimes used to set the microprogram counter to a certain address, often the reset address.

The microprogram sequencer consists of a multiplexer, an adder, a stack, a control logic block, and an address register. The multiplexer allows selecting an input address from four different sources: the direct input address, the branch address specified in the microinstruction, the adder, or the stack. The multiplexer is controlled by the control logic block according to the SYSTEM FLAGS and the OP-CODE. The adder is used to increment the microprogram address in sequential operations. The stack is used to store the return address of subroutine calls. When a subroutine call is executed, the return address is pushed into the stack and the subroutine address stored in the BRANCH ADDRESS field of the microinstruction is loaded into the address register. When a subroutine return is encountered, the return address stored in the stack is popped out of the stack and loaded into the address register. The address register is used to maintain the address for the microprogram memory until the next address is available.

1.2 HARDWIRED MICROCONTROLLER

1.2.1 AN OVERVIEW

Figure 1-3 Block diagram of a hardwired microcontroller.

The hardwired microcontroller architecture presented in this thesis is a different architecture which supports the same basic microinstruction set and offers the same functions as those of the conventional architecture shown in figure 1-2. The unique features of this architecture are that the microprogram memory cell array is driven by two shift registers, and the microprogram sequencer is implemented by physical wire connections and some sequencing logic blocks.

The block diagram of a hardwired microcontroller is shown in figure 1-3 with the system input and output interfaces omitted. The hardwired connections and the sequencing logic blocks are shown as control logic blocks. The decoding logic block is very similar to that of a conventional microcontroller. It decodes the op-code of the microinstruction and the system status to generate flags that indicate which operation: sequential, branch, call or return, is being carried out.

Upon a reset, the first stage of each shift register is reset to one while all other stages are reset to zero. This effectively activates the first row and the first column of the memory cell array. In the subsequent clock cycles, the shift registers shift the logic one downwards (or to the right) one stage at a time to activate different rows and columns of the memory array. When the shift registers reach their last stages, they loop back to their first stages and then start again.

If the microcontroller is designed only for sequential operation where microinstructions are sequentially executed one after another, the hardwired microcontroller needs only the memory array and the shift registers. If there are branches and subroutine calls, connections between nonadjacent stages of the shift registers are required. The output of the shift register stage where a conditional branch or a subroutine call is initiated, is not only connected to the adjacent stage, but also connected to the branch or subroutine destination stage. Since only one connection can be active at a time, some sequencing logic is required to arbitrate these connections. For example, when the branch condition is true, the branch logic block activates the connection between the current stage and

the branch destination stage, and deactivates the connection between the current stage and adjacent stage to carry out the branch operation. If the branch condition is false, the connection to the branch destination stage is deactivated and the connection to the adjacent stage is activated, thus the subsequent microinstruction is executed just as in the sequential operation.

Figure 1-4 A more detailed block diagram of a hardwired microcontroller which has a subroutine call and a branch microinstruction.

Figure 1-4 shows a more detailed block diagram of a hardwired microcontroller that has a subroutine call and a conditional branch microinstruction. There are basically three types of hardwired sequencing logic blocks: branch, subroutine call, and subroutine return. For every nonsequential microinstruction, a conditional branch, a subroutine call or a subroutine return, a corresponding sequencing logic block must be placed at the appropriate stage of each shift register. Each of these sequencing logic blocks can be seen as a switch that connects the output of the current shift register stage to either the input of the adjacent stage or the input of the 'branch destination' or the 'subroutine address' stage. The subroutine call block not only arbitrates the connections but also has some memory to mark the return address.

All microprogram sequencing operations of a hardwired microcontroller are carried out through hardwired connections and logic.

1.2.2 THE INCENTIVES

There are several potential advantages that the hardwired microcontroller architecture may offer. The most important advantage is the microcontroller speed. With the hardwired microprogram sequencer, the hardwired microcontroller requires almost zero time to feed the next microinstruction regardless the type of the microinstruction. This property makes a hardwired microcontroller much faster than a normal conventional microcontroller. Pipelined conventional microcontrollers may be just as fast as a nonpipelined hardwired microcontroller, however they still require an extra memory cycle for a conditional branch microinstruction to flush the pipeline and to feed the branch address while the hardwired microcontroller does not. The hardwired microcontroller can generate a condition code, test and branch in the same cycle.

A hardwired microcontroller if designed properly may also require less semiconductor space. In general, a hardwired microinstruction is narrower than a conventional microinstruction. Consequently the hardwired

microprogram memory is smaller than the equivalent conventional microprogram memory. A sequential hardwired microprogram sequencer needs only the shift registers for the address decoders. With branches and subroutine calls, the hardwired microprogram sequencer requires sequencing logic blocks and wire connections, but the additional area taken by the hardwired sequencer is likely to be less than the space saved from the microprogram memory.

Another potential advantage that the hardwired microcontroller architecture has is that the shift registers may be modified and used for built-in test circuits. There are several design for testability techniques, which make use of shift registers to test imbedded memories. Examples of these techniques are the Easily Testable PLA Schemes: FK [FuJH81], HO [HonS80], SKF [SalK83], K [KhaJ83].

The hardwired microcontroller architecture may also be advantageous for distributed microcontrollers. Hardwired distributed microcontrollers are fast and simple to synchronize. One needs only to shift a logic one in or out of the shift registers to start or stop a hardwired microcontroller.

1.3 THESIS OVERVIEW

Chapter 2 examines various microprogramming issues such as microinstruction format, microinstruction set, and microinstruction sequencing. The chapter provides the background on some major factors that determine the size and the speed of a microcontroller.

Chapter 3 provides a brief description of a conventional ROM as a background for the memory array used in a hardwired microcontroller.

Chapter 4 presents the architecture and the theory of operations of hardwired microcontrollers.

Chapter 5 shows an implementation example of the hardwired microcontroller architecture.

Chapter 6 discusses various design issues of hardwired microcontrollers.

Chapter 7 provides a detailed area and speed comparison between a hardwired microcontroller and a conventional microcontroller.

Chapter 8 presents the thesis conclusions and discusses future work.

CHAPTER 2

2. MICROPROGRAMMING ISSUES

The format of the microinstruction, the organization of the microprogram, and the way the microinstructions are sequenced, are the major factors that determine the area of the microprogram memory as well as the speed of the microcontroller.

2.1 MICROINSTRUCTIONS

A microinstruction usually consists of three fields:

The first two fields contain bits that are required to generate the next state for the microcontroller. The first field specifies the type of the microinstruction, which is often referred to as Operation Code or OP-CODE. The second field is normally used to store the 'branch address' for conditional branch or subroutine call microinstructions.

The third field is usually reserved for the output control signals which control the hardware functions of the controlled system.

Figure 2-1 Basic microinstruction format.

There are basically two different formats for microinstructions: non-coded and encoded formats.

2.1.1 NON-ENCODED FORMAT

In the non-encoded format, each bit and each field of the microinstruction is used solely for one unique function of the microcontroller. For example, bits 0-2 are used strictly for the OP-CODE, bits 3-15 are reserved for the branch address regardless whether the microinstruction is a conditional branch or not, and bits 16-63 are tied to the control signals of various hardware functions of the controlled system. This format is also known as unpacked or exploded format. Non-encoded format leads to wide microinstructions, large and inefficient microprogram memory, but high operating speed and high degree of concurrency. Since only a few microinstructions are conditional branches, the bits reserved for the branch address in the microinstructions are wasted most of the time. However, with this format for the microinstructions, the microcontroller requires minimum decoding logic for the microinstructions. Also, it can send a different output control pattern to the controlled system in every microinstruction. Therefore the non-encoded microcontroller can yield very fast throughput.

2.1.2 ENCODED FORMAT

There are several encoding techniques for microinstructions to reduce the size of the microprogram memory.

One technique encodes the field reserved for the output control signals of the microinstruction. For example, assume that 256 control lines are required, in which only one line can be active at a time. Instead of dedicating 256 bits in the microinstruction for 256 separate control lines of the controlled system, one can use just 8 bits to represent 2^8 possibilities and adds an 8-to-256 demultiplexer to decode the 8 bits of the microinstruction to 256 control lines for the controlled system. If the propagation delay of the decoder is much less than the microinstruction cycle (the time required to execute a microinstruction), there will not be any performance loss.

Another technique attempts to reduce the size of the field reserved for the branch address. The microprogram memory is divided into pages. To address a microinstruction, one must give the page reference and the offset reference within the page. The microinstruction normally has only a few bits reserved for the branch address. When a conditional branch is encountered, these bits represent the page reference of the branch location. The subsequent microinstruction will specify the offset reference of the branch address using all the bits reserved for the OP-CODE and the branch address. This technique reduces the branch address field of microinstruction but it requires two microinstruction cycles for a conditional branch, and some extra logic for the address decoders.

A similar technique stores all the branch addresses in a second ROM. The branch address field of the microinstruction specifies only the address of the branch addresses stored in the second ROM. Let m be the number of branches and n be the total number of microinstructions. The branch address field consists of only $\log_2 m$ bits instead of $\log_2 n$ bits. This technique is much faster than the paging technique previously mentioned.

In general, encoded format leads to shorter microinstructions, a smaller microprogram memory with the loss of some concurrency and some speed.

2.1.3 BASIC MICROINSTRUCTION SET

A typical microinstruction set for a microcontroller usually supports 3 modes of operations: sequential, branches and subroutine calls. Table 2-1 on next page shows the instruction set of the Signetic microprogram sequencer series 8X02 as an example [AleN84].

In the sequential operation, the microprogram address is automatically incremented by one after the current microinstruction is executed. Microinstructions are executed sequentially.

<i>Mnemonic</i>	<i>Function description</i>
TSK	AC₂₋₀ = 000: TEST & SKIP Perform test on TEST INPUT LINE. If test is Next Address = Current Address + 1 FALSE(LOW): Stack Pointer unchanged If test is Next Address = Current Address + 2 TRUE(HIGH) (i.e. Skip next microinstruction) Stack Pointer unchanged
INC	AC₂₋₀ = 001: INCREMENT Next Address = Current Address + 1 Stack Pointer unchanged
BLT	AC₂₋₀ = 010: BRANCH TO LOOP IF TEST CONDITION TRUE. Perform test on TEST INPUT LINE. If test is Next Address = Current Address + 1 FALSE(LOW): Stack Pointer decremented by 1 If test is Next Address = Address from Stack TRUE(HIGH): Register File (POP) Stack Pointer decremented by 1
POP	AC₂₋₀ = 011: POP STACK Next Address = Address from Stack Register File (POP) Stack Pointer decremented by 1
BSR	AC₂₋₀ = 100: BRANCH TO SUBROUTINE IF TEST CONDITION TRUE. Perform test on TEST INPUT LINE. If test is Next Address = Current Address + 1 FALSE(LOW) Stack Pointer unchanged If test is Next Address = Branch Address Input (B₀₋₉) TRUE(HIGH): Stack Pointer incremented by 1 PUSH (write) Current Address + 1 - Stack Register File
PLP	AC₂₋₀ = 101: PUSH FOR LOOPING Next Address = Current Address + 1 Stack Pointer incremented by 1 PUSH (write) Current Address - Stack Register File
BRT	AC₂₋₀ = 110: BRANCH ON TEST CONDITION TRUE Perform test on TEST INPUT LINE. If test is Next Address = Current Address + 1 FALSE(LOW): Stack Pointer unchanged If test is Next Address = Branch Address Input (B₀₋₉) TRUE(HIGH) Stack Pointer unchanged
RST	AC₂₋₀ = 111: RESET TO ZERO Next Address = 0 Stack Pointer unchanged

Table 2-1   The Functional Description of the Signetic 8X02 Microprogram Sequencer Instruction Set

In the branch operation, the branch address specified in the current microinstruction may be conditionally or unconditionally loaded into the address register, which forces the microprogram to branch to a specified memory location after the current microinstruction is executed.

Subroutine calls are similar to branches except that the current address is incremented by one and pushed into the stack before the microprogram branches to the subroutine. Upon completion of the subroutine, the return address stored in the stack is popped out and loaded into the address register to carry out a subroutine return.

2.2 MICROINSTRUCTION SEQUENCING

The two major aspects of microinstruction sequencing are the timing of the microinstruction fetches (serial or parallel, depending upon the non-overlap or complete overlap between the execution of the current microinstruction and the fetching of the next microinstruction), and the method to take care of conditional branches in the microprogram [AleN84].

2.2.1 SERIAL OR NON-PIPELINED MICROINSTRUCTION SEQUENCING

Serial or non-pipelined microinstruction sequencing does not allow fetching the address of the next microinstruction until the execution of the current microinstruction is completed. The microprogram sequencer of this scheme is generally simple and slow.

2.2.2 PARALLEL OR PIPELINED MICROINSTRUCTION SEQUENCING

Parallel or pipelined sequencing allows the next microinstruction to be fetched concurrently with the execution of the current microinstruction. This scheme of sequencing is generally twice as fast as serial sequencing, but it presents some problems in handling conditional branches where the next microinstruction is not known until the end of the execution of the current microinstruction.

2.2.3 SERIAL-PARALLEL MICROINSTRUCTION SEQUENCING

Serial-parallel sequencing is the combination of both of the sequencing schemes previously described. Fetching of the next microinstruction is done concurrently with the execution of the microinstruction only if the microinstruction is not a conditional branch. If the current microinstruction is a conditional branch, the next microinstruction address will not be fetched until the execution of the current microinstruction is completed.

CHAPTER 3

3. CONVENTIONAL ROMs

Many microprogram memory implementations use ROM (Read Only Memory) because it is non-volatile, that is the information stored in a ROM will not be lost even when the power supply to the ROM is cut off.

The simplest form of ROM is mask programmed ROM. Mask programmability can be achieved via contact programming: presence or absence of a transistor, or an implant to turn a memory cell transistor permanently on or off. Transistor programming is preferred in microprogrammed systems since they are fast and require the least dynamic power dissipation. Contact or metal programming is preferable in more generic applications. Mask programmed ROMs are widely used in high volume productions since they are fast and inexpensive.

Programmable ROMs (PROMs) are preferable for low volume production systems. PROMs are slower and more expensive than ROMs but they can be programmed by the user and the cost difference may not be as much as the overhead cost to manufacture the ROM masks.

Erasable PROMs (EPROMs) are more suitable for prototypes and preproduction systems where the microprogram or firmware may be changed. EPROMs are much slower and more expensive but their flexibility make them the most popular among ROMs.

3.1 ROM BASIC ARCHITECTURE

Figure 3-1 on next page shows the block diagram of a linear ROM. It generally consists of an address decoder, a memory cell array, a precharge circuit, and a sense amplifier circuit.

Figure 3-1 Block diagram of a 1Kx8 linear ROM.

3.1.1 ADDRESS DECODER

The address decoder can be seen as an n -to- 2^n demultiplexer with n is the number of input address lines. The decoder interprets the n input address lines to select one of 2^n control lines of the memory array to be active. Figure 3-2 shows a 4-to-16 address decoder as an example.

Figure 3-2 An example of an address decoder, 4-to-16 decoder.

3.1.2 MEMORY ARRAY

The memory array is an array of memory cells, in which each cell holds a unique digital logic state. Memory cells are implemented with one transistor per bit of storage as shown in figure 3-3. The horizontal lines are often referred to as word lines, and the vertical lines are referred to as bit lines. The example memory array is known as a Linear ROM where the number of word lines corresponds to the number of memory locations, and the number of bit lines corresponds to the number of output bits.

The memory array shown in figure 3-3 is also known as a static memory array where the bit lines are statically precharged. When a word line is active, the memory cell transistors controlled by that word line are activated to pull the corresponding bit lines to ground to output a logic 0. Bit lines which are not grounded output a logic 1.

Memory arrays are also implemented dynamically as shown in figure 3-4, to increase the memory speed. The memory array incorporates a dynamic precharge and discharge mechanism for the bit lines using a two-phase non-overlapping clock. During phase one, all bit lines are pulled high or precharged. During phase two, those bit lines which have a path to ground, via a memory cell transistor activated by the relevant word line, are conditionally discharged ie. output a logic 0. Bit lines in which no transistors have been programmed for that specific word, remain in the precharge state ie. output a logic 1. Since no active pullup operation is involved during the read cycle, the speed of operation is high.

3.1.3 PRECHARGE CIRCUIT

The precharge circuit is used to precharge the bit lines of the memory array as previously described. The precharge circuit for a static memory array usually consists of a series of transistors operating as pullup resistors as shown in figure 3-3. The precharge circuit for a dynamic memory array consists of a series of transistors which dynamically connect the bit lines to Vdd, based on the level of a PRECHARGE control signal (θ ,

in the example shown in figure 3-4).

Figure 3-3 An example of a static memory array.

Figure 3-4 An example of a dynamic memory array - Bit lines are dynamically precharged and discharged.

3.1.4 SENSE AMPLIFIER

The output voltage level of the bit lines is usually much lower than the standard TTL or CMOS level due to the large capacitive load on the bit line. To convert the output voltage level of bit lines to the proper voltage level, sense amplifiers are used to sense the logic level of the bit lines and then amplify it to the proper output voltage level. A sense amplifier can be as simple as a non-inverting buffer with a low input voltage threshold.

3.2 X-Y MEMORY ARRAY

The memory arrays shown so far are classified as linear ROMs where each bit line is corresponding to an output bit of the ROM. One major set back of linear ROMs is that the bit lines are generally very long, which can present problems such as driving capability and speed. The longer the bit lines are, the more current each bit line draws, and the longer it takes to switch the state of a bit line.

Another property of linear ROMs is that the word lines are usually much shorter than the bit lines. This property is taken advantage of in the X-Y memory array implementation to shorten the bit lines. Generally, an X-Y memory array is arranged as a two dimensional array of memory cells as shown in figure 3-5 on next page. There are two address decoders: a row (X) address decoder and a column (Y) address decoder. The row address decoder is used to drive the word lines. The column address decoder is used to multiplex several bit lines to a single output bit. The number of bit lines is increased, however the number of word lines is significantly reduced. Consequently, the word lines are slightly longer but the bit lines are much shorter than those of the linear ROM. The number of bit lines is usually kept low enough so that the load of the word lines and the propagation delay of the bit line multiplexing circuit will not increase enough to affect the performance of the memory array.

Figure 3-5 Block diagram of a dynamic X-Y ROM.

3.3 VIRTUAL GROUND MEMORY ARRAY

Considering the dynamic memory array shown in figure 3-4, there are one and a half metal lines for every bit line in the memory array: one for the bit line itself and a half for the shared ground line. This implementation is not very efficient. These unnecessary ground lines may be eliminated by using the concept of virtual ground [GlaL88].

Considering the virtual ground memory array shown in figure 3-6, there are only nine metal lines for every eight bit lines. The memory cell transistors are addressed by raising a word line and an appropriate column line to select the correct bit output. For example, to address bits 0, 1, 8 and 9 on word line 7, row 7 and column 0 are raised. When the output enable OE is high, the virtual ground lines between bit 0 and bit 1, bit 8 and bit 9, are pulled low. The memory cell transistors at these bit lines which are activated by word line 7, consequently connect all these bit lines to ground to output a logic 0 at Z0, Z1, Z2 and Z3. If row 7 and column 3 are raised and the output is enabled, the ROM will output 0011 at Z0, Z1, Z2 and Z3. Note that this memory array by definition is also an X-Y memory array.

One major issue in virtual ground memory arrays is the sneak paths. For example, raising row 4 and column 3 will cause all bit lines between bit 8 and bit 15 to discharge. The multiplexing transistors controlled by COL0-COL3 insure that only the appropriate virtual ground and bit lines within a group are activated. However, bit lines from another group, b0-b7, also share the same multiplexing control signals. If there had been no gap between group (b8-b15) and its neighbouring group (b0-b7), bit 8 would have pulled bit 7 down and caused an incorrect bit output at Z1. One method to control this problem is to have a gap between every two groups of bit lines which share the same set of control lines (columns). The addition of these gaps also establishes the limit on the maximum load on a single bit line (the total load of all bit lines in the group).

Figure 3-6 Virtual ground memory array.

CHAPTER 4

4. HARDWIRED MICROCONTROLLER

The structure of a hardwired microcontroller is very similar to a conventional ROM. The memory cell array, the precharge circuitry, and the bit line sensing and multiplexing circuitry are exactly the same. Figure 4-1 shows the block diagram of a 1K X 8 bit hardwired microcontroller with an X-Y microprogram memory array.

Figure 4-1 Block diagram of a 1K X 8 bit hardwired microcontroller. The memory cell array, the memory cells precharge circuitry, and the bit line sensing and multiplexing are exactly the same as those of a conventional ROM.

Figure 4-2 shows a more detailed block diagram of the hardwired microcontroller.

Figure 4-2 A more detailed block diagram of a hardwired microcontroller. For every non-sequential microinstruction, branch, call or return, two sequencing logic blocks are required, one for each shift register.

The address decoders are replaced by two customized shift registers called shift register row and column decoders, or simply row and column shift registers. Each stage of the row decoder is connected to a word line (row) of the memory array and each stage of the column decoder is connected to a control line (column) of the bit line multiplexing circuit. Since only one row and one column can be active at a time, only one stage of the shift registers can be active at any given time. A logic 1 output from a stage of the shift register activates the corresponding row or column line. All stages of the shift register decoders are linked together in a closed loop.

For sequential operations, the output of every stage of the shift register is connected to the input of the adjacent stage to allow the active bit to sequentially pass through the shift register.

For conditional branches, the output of the stage where the branch microinstruction resides not only has a connection to the adjacent stage but also has an alternate connection to the branch destination stage. These connections are arbitrated by a control logic block called BRANCH block.

Subroutine call implementation is similar but more complex than branches. Connections between the calling microinstruction and the subroutine address, and between the end of the subroutine and the return addresses are required. Control logic is required not only to arbitrate different connections but also to keep track of the return address.

Since the shift registers are independent of each other, a unique combination of wire connections and some arbitration logic is required for each shift register for every non-sequential microinstruction.

All clock signals and control signals for the row and column shift registers, the precharge circuit and the output control circuit are generated by the clock generator.

4.1 SEQUENTIAL OPERATION

Figure 4-3 shows the simplest form of a microcontroller where microinstructions are executed sequentially one after another. There are no conditional branches and no subroutine calls in the microprogram. All stages of the row and column shift registers are linked together with the output of one stage connected to the input of the adjacent stage. The clock generator is not shown to simplify the diagram.

Figure 4-3 The simplest form of hardwired microcontrollers which allows only sequential operations.

Upon a reset condition, the first stages of the shift registers, row 0 and column 0, are reset to an active logic state, say logic 1, while all other stages are reset to logic 0. This activates the memory cell controlled by row 0 and column 0. In the next clock cycle, the logic 1 is shifted downwards by one stage, the second stages are active while all others are passive. This time the memory cell controlled by row 1 and column 1 will be activated. When the active state reaches the last stage of a shift register, it loops back to the first stage.

Figure 4-4 Sequence of execution of a sequential hardwired microcontroller. Memory cells are sequentially activated one after another starting from cell 0 then cell 1, cell 2, .. cell 39, then back to cell 0.

Figure 4-4 illustrates the sequence of execution of the hardwired microcontroller shown in figure 4-3. One by one all memory cells are sequentially activated with cell 0 executed first then cell 1, 2, .. 39, then back to cell 0. Since the shift registers are running independently

of each other, the numbers of stages of the shift register decoders must not be divisible by each other.

Figure 4-5 illustrates an example where the number of stages of the row shift register is divisible by the number of stages of the column shift register. The maximum number of memory cells that can be addressed in this case is only the number of stages of the larger shift register instead of the product of the numbers of stages.

Figure 4-5 Sequence of execution of a hardwired controller where the number of stages of the row shift register is divisible by the number of stages of the column shift register. The maximum number of memory cells that can be addressed is only the number of stages of the larger shift register, 12, instead of the product of the numbers of stages, 12×4 .

This restriction introduces the first rule of the hardwired microcontroller architecture.

Rule 1   The number of stages of the shift registers must not be divisible by each other.

Note that the circuit shown in figure 4-5 will work properly if the column decoder runs at $1/3$ of the speed of the row decoder. However, this requires more complex clock and control circuitries.

4.2 DYNAMIC SHIFT REGISTER ROW/COLUMN DECODERS

Figure 4-6 Dynamic shift register row/column decoder. Each stage of the shift register controls a row or a column of the memory array.

Figure 4-6 shows an example of a 4-stage shift register that may be used in place of the conventional 2-to-4 address decoder. Each stage of the shift register is used to drive a row or a column of the microprogram memory array. An M-rows by N-columns memory array generally requires an M-stage shift register for the row decoder and an N-stage shift register for the column decoder.

Each stage of the shift register can be divided into two phases: input and output, with each phase consisting of a transmission gate and an inverter.

Phases are separated from each other by the transmission gates which are driven by two non-overlapping clocks θ_1 and θ_2 . The input phase allows shifting in the input logic state when clock θ_1 is active. The output phase shifts out the latched logic state when clock θ_2 is active. The inverter of the output phase should be large enough to drive a row or a column of the memory array and the input phase of one or more shift register stages. The inverter of the input phase needs only be large enough to drive the output phase. Every stage of the shift register also has a reset transistor at the output phase to allow an asynchronous reset. The first stage of a shift register has the reset transistor grounding the input of the output inverter to output a logic 1. This shift register stage is called a RESET-TO-HIGH stage. All subsequent stages of the shift register have the reset transistor pulling the input of the output inverter to Vdd to output a logic 0. These stages are called RESET-TO-LOW stages.

Figure 4-7 on next page shows the timing requirements for a hardwired microcontroller.

t_1 is the minimum time required between the non-overlapping clock phases θ_1 and θ_2 . In general, t_1 can be as low as 0 provided that the worst case overlapping time between θ_1 and θ_2 is significantly less than the propagation delay of both phases of the shift register. The overlap will not cause the shift registers to function incorrectly.

t_2 is the minimum setup time required for the output enable OE. This time interval is necessary to ensure that the virtual ground is activated only when all rows and columns are valid. If any rows or columns are still in transition and OE goes active, some bit lines may be incorrectly discharged which will result in an incorrect output word.

t_3 is the time delay required to evaluate a branch condition. t_4 is the setup time for clock phase θ_1 . This setup time is required to ensure that a new 'address' is not latched until the branch condition is evaluated.

The width of θ_1 is determined by the propagation delay of the shift register input phase and the required precharge time for the memory cells.

The width of θ_2 is determined by the propagation delay of the shift register output phase, the time required to read and send the output word to the controlled system, and the time to evaluate a branch condition.

TIMING REQUIREMENTS FOR A HARDWIRED MICROCONTROLLERS

Figure 4-7 Timing requirements for a hardwired microcontroller.

To reset the shift register, the system reset flag, RESET, must be held high for at least the width of clock θ_1 . This is to ensure that all shift register input phases are reset properly. The first stage of each shift register is reset to output a logic 1 and all other stages are reset to output a logic 0. This is similar to resetting the microprogram counter to zero.

To start normal operations, the system reset flag RESET must be lowered before clock phase ϕ_2 goes high. The shift registers will start shifting the logic 1 downwards as follows

Output clock phase: clock ϕ_1 goes low and clock ϕ_2 goes high.

The output transmission gate is activated and the input transmission gate is deactivated. The output of the input inverter is passed through to the output inverter. The output inverter outputs the logic level of the previous shift register stage that was shifted in, in the input clock phase. Meanwhile, the input transmission gate is off which prevents the new output of the previous shift register stage from getting through to the input inverter. In effect, the input circuitry maintains the previously latched logic level and the output circuitry shifts out the latched logic.

Input clock phase: clock ϕ_1 goes high and clock ϕ_2 goes low

The input transmission gate is activated to pass the output of the previous stage through to the input inverter. The output transmission gate is deactivated to prevent the latched logic from getting through to the output inverter. The output inverter outputs the inverse of the logic state dynamically stored at the gate of the output inverter. Effectively, the input logic is shifted in and the output logic remains the same.

All clock phases must be kept short enough so that the charges stored at the gates of the inverters will not leak enough to switch the inverter output.

Figure 4-8 shows an alternate circuit for the dynamic shift register row/column decoder. This circuit does not use transmission gates which generally can cause problems for in-circuit testers. However to keep the circuit simple, the circuit shown in figure 4-6 will be used through out the remainder of this thesis.

Figure 4-8 An alternate circuit for the dynamic shift register row/column decoder that is more testable.

4.3 CLOCK GENERATOR

CLOCK GENERATOR

TRUTH TABLE

Q_{n-1}	$\overline{Q_{n-1}}$	CLK	Q_n	$\overline{Q_n}$	OE	B1	B2
0	1	0	0	1	1	0	0
0	1	1	1	0	0	1	0
1	0	0	1	0	0	0	1
1	0	1	0	1	1	0	0

TIMING

Figure 4-9 The clock generator for a hardwired microcontroller - Schematic diagram, truth table and timing diagram.

The main purposes of the clock generator are to provide two non-overlapping clocks for the shift registers, and to synchronize these clocks with the PRECHARGE*/OE signal of the virtual ground dynamic memory cell array.

From figure 4-7, there are three main constraints for the clock generator:

First, clock phases θ_1 and θ_2 must be non-overlapping or the overlap must be significantly less than the propagation delay of both phases of the shift register.

Second, the output enable OE which activates the virtual ground of the memory array must not go active until all rows and columns are valid.

Third, clock phase θ_1 which latches in the new 'address' must not go active until all rows and columns are valid and the branch condition has been evaluated. This constraint allows the hardwired microcontroller to generate a condition code, test and branch in the same microinstruction cycle.

A simple circuit of a hardwired microcontroller clock generator, that satisfies these constraints, is shown in figure 4-9. There are basically three clock phases.

Clock phase 1: θ_1 goes high while θ_2 and OE stay low

Clock θ_1 goes high to shift in the new 'address' as described in the previous section. All rows and columns are valid and stable, θ_2 is low. Branch conditions must have already been evaluated since OE is low. The memory array is precharged.

Clock phase 2: θ_2 goes high, θ_1 goes low and OE remains low

Clock θ_2 goes high to output the 'latched address'. Both of the row and column shift registers are shifted by one. The output enable OE can not go active yet since the rows and columns are still in transition. The memory array is still precharged.

Clock phase 3: OE goes high, θ_1 and θ_2 are low

OE goes high to activate the virtual ground and enable the memory output. At this point, all rows and columns are stable since θ_2 is low. θ_1 can not go high yet because the branch conditions are being evaluated in this

phase.

The width of θ_1 should be larger than the propagation delay of the input phase of a shift register stage. The width of θ_2 should be larger than the propagation delay of the output phase of a shift register stage. A microinstruction cycle should be larger than the total of the precharge time and the time required to read and output the microinstruction and to evaluate the branch conditions .

4.4 BRANCH IMPLEMENTATION

- N is the output of the current stage of the shift register.
- N+1 is the input of the adjacent stage.
- N+B is the input of the branch destination stage.
- B is the branch conditional flag.
- EN is the output of the current stage of the other shift register.

Figure 4-10 A simple representation of a branch block.

Branch operations may be implemented in a hardwired microcontroller using the branch logic block shown in figure 4-10 and some wire connections between stages of the shift register decoders. Two branch blocks are required for every branch microinstruction in the microprogram with one block for the row shift register and one block for the column shift register.

'N' of the branch block is connected to the output of the current shift register stage. 'N+1' is connected to the input of the adjacent (next) stage, and 'N+B' is connected to the input of the branch destination stage.

The branch block arbitrates different wire connections based on the branch conditional flag 'B' and the branch enable flag 'EN'. 'B' is the system flag that signifies if a branch operation is to be carried out. This flag is derived from the op-code of the microinstruction together with external branch conditions. 'EN' is the branch enable which is used when there are more than one branch blocks at the same stage of the shift register. To ensure that only the proper branch block is activated when the branch flag 'B' is active, the 'EN' input of each block should be connected to the output of the current stage of the other shift register. If there is only one branch instruction at a stage of the shift register, the branch enable 'EN' should be connected to the system branch flag 'B'.

Figure 4-11 on next page shows an example of how branch blocks are used to implement branches in the microprogram. There are two branches at column 02, one from address (12,02) and the other from address (25,02), and another branch at column 08. The branch conditional flag 'B' of all branch blocks are connected to the system flag 'BRANCH'. There is only one branch block at column 08 so the branch enable 'EN' of this block can be connected to the branch flag, 'B'. At column 02, there are two branch blocks so the branch enable 'EN' of each block must be connected to the appropriate row address.

When the branch condition is false, all branch blocks direct the output of the current stage 'N' to the input of the adjacent stage 'N+1'. As the result, the microprogram sequencer will execute the subsequent microinstruction.

Figure 4-11 An example of how branches are implemented in a hardwired microcontroller. Only the column decoder is shown.

When both of the branch conditional flag 'B' and the branch enable 'EN' are true, the branch block will direct the output of the current stage to the branch destination stage. The 'branch destination' microinstruction will be executed instead of the subsequent microinstruction. Note that the branch conditional flag 'BRANCH' is only true when the current microinstruction is a branch microinstruction and the branch conditions are true.

Suppose that the current address is (25, 2) and the branch conditions are true, the system branch conditional flag BRANCH will be true. The top branch block has the branch enable flag 'EN' false so it will output a logic 1 at 'N+1' and a logic 0 at 'N+B'. The middle branch block has the branch enable flag true so it will output a logic 0 at 'N+1' and a logic 1 at 'N+B'. Note that the 'N+1' of these two branch blocks are ANDed together before being fed to the input of the adjacent stage. This insures that when there is a branch taking place from one of the blocks at the same address, the adjacent stage will not be activated. The adjacent stage is activated only when all branches from the current address are disabled. In this case, a logic 0 will be directed to column 3 and a logic 1 to column 8. A branch between column 2 and column 8 will be carried out.

The bottom branch block is enabled but its input 'N' is at logic 0 so only a logic 0 is directed to its branch destination stage. This block does not carry out a branch operation.

When the current stage is at column 8 and all branch conditions are true, the bottom branch block will carry out a branch to column 12.

Note that the 'N+B' output of the branch blocks must be ORed together with the input of the destination stage to insure that this stage can be activated either sequentially or by a conditional branch.

Two more rules for the hardwired microcontroller architecture have been derived from this example.

Rule 2 All outputs 'N+1' of branch blocks at the same row or column address must be ANDed together before being fed into the adjacent address.

Rule 3 All outputs 'N+B' of the branch blocks must be ORed together with the other inputs to the destination address.

Figure 4-12 shows the schematic diagram and the truth table of a branch sequencing logic block.

BRANCH BLOCK

NOTATION:

- B: BRANCH CONDITION FLAG
- EN: BRANCH ENABLE
- N: OUTPUT FROM THE CURRENT SHIFT-REGISTER STAGE
- N+1: INPUT TO THE NEXT SHIFT-REGISTER STAGE
- N+B: INPUT TO THE 'BRANCH' DESTINATION STAGE

CIRCUIT LAYOUT

TRUTH TABLE

EN	B	N	N+1	N+B
0	0	0	0	0
0	0	1	1	0
0	1	0	0	0
0	1	1	1	0
1	0	0	0	0
1	0	1	1	0
1	1	0	0	0
1	1	1	1	1

Figure 4-12 Hardwired microcontroller branch block, schematic, floor planning, and truth table.

4.5 SUBROUTINE CALL IMPLEMENTATION

Subroutine calls can also be implemented in a hardwired microcontroller. The only restriction on hardwired subroutine calls is that they must not be recursive since each subroutine call is implemented by some wire connections and two call blocks, one for the row address and the other for the column address. A recursive call would require impossible wire connections and an infinite number of call logic blocks. Nested subroutine calls are allowed in a hardwired microcontroller. The maximum number of nesting levels is dependent on how complex the microcontroller layout can get.

In general, two call blocks are required for every subroutine call and two return blocks are required for every subroutine return with one block for the row shift register and the other for the column shift register.

4.5.1 CALL BLOCK

Figure 4-13 on next page shows a simple presentation of the CALL block. A call block can be functionally divided into three logic blocks: the call logic, the stack and the return logic. Note that the return logic block described herein is not the same as the return block used to implement a subroutine return.

The call logic block decodes the system flag 'CALL' together with the output of the current shift register stage N, and the call enable flag 'EN' to direct the output of the current stage to the input of the adjacent stage, 'N+1', or to the input of the subroutine stage, 'N+C'. The enable flag, 'EN', is similar to that of the branch block. When there are more than one subroutine calls at the same stage of a shift register decoder, 'EN' should be connected to the output of the current stage of the other shift register to insure that only the proper subroutine call is executed. When there is only one subroutine call at the current shift register stage, EN should be connected to the system flag CALL. This system flag is usually decoded from the microinstruction op-code to

indicate if a subroutine call is to be carried out.

- N Output of the current shift register stage (current address).
- $N+C$ Input of the subroutine address stage.
- $N+1$ Input of the adjacent stage.
- $N+2$ Clear stack flag, output of the adjacent stage.
- $EN+2$ Clear stack enable, output of the adjacent stage of the other shift register.
- EN Call enable, output of the current stage of the other shift register.
- $CALL$ System flag indicating if a subroutine call is to take place.
- RET_C^* Common return path of a subroutine.

Figure 4-13 A simple presentation of a call logic block.

The stack is a set-reset flip-flop that is set whenever the subroutine is called ('N+C' is active), and reset after the subroutine has returned (both 'N+2' and 'EN+2' are active). 'N+2' and 'EN+2' are the output of the adjacent stage of the shift registers. These signals represent the address of the microprogram after it has returned successfully from the subroutine call. The stack output indicates if the subroutine is to return to this subroutine call.

The return logic block decodes the output of the stack r_{en*} and the subroutine return flag 'RET_C*' to generate the appropriate input for the return address stage of the shift register. When the current microinstruction is a subroutine return ('RET_C*' is active), and the stack output ' r_{en*} ' is active, the return logic will direct the return signal, 'RET_C*', to the 'return address' stage to carry out a return operation to this location.

Note that the return signal 'RET_C*' is connected to all subroutine call blocks of the same subroutine. Different subroutines must have different return signals.

The schematic diagram and the truth table of the call block is shown in figure 4-14 on next page.

CALL BLOCK

NOTATION

- EN+2: CLEAR STACK ENABLE
- N+2: CLEAR STACK FLAG
- RET_C: RETURN PATH FOR SUBROUTINE C
- N+1: INPUT TO THE NEXT SHIFT-REGISTER STAGE
- N+C: INPUT TO THE 'CALL' SHIFT-REGISTER STAGE
- CALL: CALL STATUS FLAG
- EN: CALL ENABLE
- N: OUTPUT FROM THE CURRENT STAGE

RETURN LOGIC		
RET_C	F+EN	N+1
0	0	0
0	1	1
1	0	1
1	1	1

STACK LOGIC			
R		S	
EN+2	N+2	N+C	F+EN
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1
1	1	1	N/A

CALL LOGIC				
CALL	EN	N	N+1	N+C
0	0	0	1	0
0	0	1	0	0
0	1	0	1	0
0	1	1	0	0
1	0	0	1	0
1	0	1	0	0
1	1	0	1	0
1	1	1	1	1

Figure 4-14 Hardwired microcontroller call block, schematic diagram and truth tables.

4.5.2 RETURN BLOCK

- N   Output of the current shift register stage.
- EN   Enable, output of the current stage of the other shift register.
- N+1   Input to the adjacent stage.
- RET_C*   Return path for all subroutine calls of the same subroutine.

Figure 4-15 A simple presentation of the return block.

Similar to the branch block and the call block, two return blocks are required for every subroutine return microinstruction, one for the row address and one for the column address. 'N' is the output of the current stage of the shift register. 'EN' is the return enable which is usually connected to the output of the current stage of the other shift register. 'RET' is the system flag which is decoded from the microinstruction opcode to indicate if a subroutine return is about to take place. 'N+1' is the input to the adjacent stage of the shift register. 'RET_C*' is the return path that is connected to all subroutine call blocks of the same subroutine.

When there is only one return block at the current shift register stage, 'EN' should be connected to 'RET'. When there are more than one return blocks at a shift register stage, all 'EN' signals must be connected to the appropriate addresses of the other shift register so that only one return block can be activated at a time.

When the current microinstruction is a subroutine return, 'RET' will be active, and of course N and EN will also be active. The return block will direct the output of the current stage to the return path, 'RET_C*', then through the appropriate calling call block, to the return address stage. When either the return flag 'RET' or the return enable 'EN' is false, the output of the current stage, 'N', will be directed to the input of the adjacent stage, 'N+1'.

RETURN BLOCK

NOTATION:

- RET: RETURN FLAG
- EN: RETURN ENABLE
- N: OUTPUT FROM THE CURRENT SHIFT REGISTER STAGE
- N+1: INPUT TO THE NEXT SHIFT REGISTER STAGE
- $\overline{\text{RET\_C}}$: RETURN PATH FOR SUBROUTINE C

TRUTH TABLE

RET	EN	N	N+1	$\overline{\text{RET\_C}}$
0	0	0	0	1
0	0	1	1	1
0	1	0	0	1
0	1	1	1	1
1	0	0	0	1
1	0	1	1	1
1	1	0	0	1
1	1	1	0	0

Figure 4-16 Hardwired microcontroller return block, schematic and truth table.

4.5.3 SUBROUTINE CALL EXAMPLE

Figure 4-17 on next page illustrates an example of how subroutine calls are implemented in the hardwired microcontroller architecture.

'RET' and 'CALL' are the system flags decoded from the microinstruction op-code.

There are three subroutine calls, one call to subroutine A and two calls to subroutine B. Suppose that subroutine A is called first at column 00 and while inside subroutine A, subroutine B is called at column 02. After the return from subroutine B and subroutine A, subroutine B is called again from outside of subroutine A at column 11.

When subroutine A is called, 'N' and 'CALL' to block 1 of SUB A are active so the output 'N+C' to column 03 (SUB A) will be active and the output 'N+1' to the adjacent stage, column 01, will be passive. The call to subroutine A is carried out. The stack of block 1 of SUB A is set.

Similarly when sub B is called from column 02, the stack of call block 1 of SUB B is also set and the active bit is sent to subroutine B, column 10.

Upon the return of subroutine B, column 07, the 'RET' signal is active but the enable lines connected to ROW19 and ROW09 insure that only the return block of subroutine B is active. The return block of subroutine A which resides at the same column remains inactive. The return path 'RET_C*' of subroutine B will be active. There are two places where subroutine B can return to but only the stack of block 1 is set and this insures that subroutine B returns to block 1 instead of block 2. After subroutine B has returned, the outputs of column 03 and row 03 will be active which will clear the stack of block 1 of SUB B.

Figure 4-17 Hardwired microcontroller subroutine calls implementation, an example. Only the column decoder is shown.

Note that the 'N+1' output of the two return blocks at column 07 are ANDed together. Also the 'N+C' output signals of the cell blocks are ORed with the other inputs to the same subroutine address. The reasons for these are similar to those stated for rules 2 and 3 described for branch blocks. Rules 2 and 3 can be restated to apply to all logic blocks as follows

Rule 2'   The N+1 output of all logic blocks at the same row/column address must be ANDed together.

Rule 3'   The N+B or N+C output of logic blocks to a shift register stage must be ORed together with the sequential input to the same shift register stage.

CHAPTER 5

5. HARDWIRED MICROCONTROLLER, AN IMPLEMENTATION EXAMPLE

5.1 DESIGN SPECIFICATIONS

Suppose that a microcontroller which requires 1K eight-bit words for its microprogram memory is to be designed using the hardwired microcontroller architecture. The microprogram sequence is summarized in table 5-1.

Microprogram Sequence

LOGICAL ADDRESS	ROW/COL ADDRESS	OPERATION	DESTINATION LOGICAL ADDRESS	DESTINATION ROW/COLUMN
0 BRN0	0/0			
22	22/6	BRANCH BRN1	100	35/4
27	27/11	BRANCH BRN2	700	50/12
63	63/15	CALL SUBA	800	20/0
80	15/0	CALL SUBB	900	55/4
81	16/1	BRANCH BRN0	0	0/0
100 BRN1	35/4			
166	36/6	CALL SUBA	800	20/0
167	37/7	BRANCH BRN0	0	0/0
700 BRN2	50/12			
715	0/11	CALL SUBB	900	55/4
788	8/4	BRANCH BRN0	0	0/0
800 SUBA	20/0			
825	45/9	BRANCH BRN3	859	14/11
859 BRN3	14/11	RET		
900 SUBB	55/4			
934	24/6	BRANCH BRN4	983	8/7
949	39/5	BRANCH BRN4	983	8/7
982	7/6	BRANCH SUBB	900	55/4
983 BRN4	8/7	RET		

Table 5-1 Hardwired microcontroller, an example microprogram.

Column 1 of the table shows the logical address of the non-sequential microinstructions such as branches, subroutine calls and subroutine returns. Column 2 shows the corresponding row and column addresses. Column 3 shows the type of the microinstructions. Column 4 shows the destination row and column addresses of the non-sequential microinstructions.

The row and column addresses are calculated as follows

Row_address = Logical_address MOD Number_of_rows.

Column_address = Logical_address MOD Number_of_columns.

The number of rows and the number of columns used to calculate the row and column addresses in table 5-1 are 65 and 16 respectively. The derivation of these numbers will be explained in the next section.

The input and output interfaces, and the microcontroller decoding logic, which drives the system status flags such as BRANCH, CALL and RETURN, are system dependent. These circuits vary from one controller to another hence will not be discussed here as they are not part of the thesis. It is assumed that the microcontroller to be designed already has the input and output interfaces, and a decoding logic that generates the BRANCH, CALL and RETURN signals. The BRANCH signal is active only when the current microinstruction is a conditional branch and all branch conditions are true. The CALL signal is active only when the current microinstruction is a subroutine call. The RETURN signal is active only when the microinstruction is a subroutine return.

5.2 LOGIC MODEL

5.2.1 MICROPROGRAM MEMORY

Suppose that a virtual ground dynamic memory cell array as shown in figure 5-1 is to be used for the microprogram memory. The memory array shown consists of twenty rows and sixteen columns which can provide a memory storage of 20 X 16 two-bit words. The memory array can be expanded horizontally to increase the number of output bits by having several blocks of memory cells placed side by side with all blocks sharing the same sixteen multiplexing control lines. For example, if the memory array is to have eight output bits, four blocks will be placed side by side to form a memory array of 20 X 16 eight-bit words. The memory array can also be expanded vertically by increasing the number of word lines. To have a memory space of 1K eight-bit words for the required microprogram memory, one would need four blocks of memory arrays, each of 64 X 16 two-bit words. To satisfy hardwired microcontroller rule 1, the number of word lines is increased to 65 to make the number of rows not divisible by the number of columns, 16. The block diagram of the virtual ground dynamic memory cell array used for a 1K X 8 bit microprogram memory is shown in figure 5-2.

Figure 5-1 A 20 X 16 X 2 bit virtual ground dynamic memory cell array.

Figure 5-2 Block diagram of the virtual ground dynamic memory cell array used for a 1K x 8 bit microprogram memory.

5.2.2 ADDRESS LOOKUP TABLES

From the microprogram sequencing logic specified in table 5-1, two address lookup tables are generated to specify the sequencing logic for the row and column shift registers. Table 5-2 shows the address lookup table for the column shift register. Table 5-3 shows the address lookup table for the row shift register.

Both tables have the same format. The first and second columns of an address lookup table specify the address of the microinstructions involved in non-sequential operations such as conditional branches, subroutine calls or subroutine returns. Column 1 specifies the row/column address. Column 2 specifies the logical address of the non-sequential microinstructions that have the same row/column address shown in column 1.

Columns 3 and 4 list the addresses of the 'branching' or 'calling' microinstructions that non-sequentially converge to the address specified in columns 1 and 2. These addresses are called source addresses. Column 3 specifies the source row/column addresses. Column 4 specifies the source logical addresses.

Columns 5 and 6 list the 'branch destination' or subroutine addresses that the microinstructions reside at the addresses specified by columns 1 and 2, diverge to. These addresses are called destination addresses. Column 5 specifies the destination row/column address. Column 6 specifies the destination logical address.

Column 7 specifies the type of the microinstruction ie. branch, subroutine call or subroutine return.

Table 5-2 Column address lookup table for the microprogram shown in table 5-1.

COLUMN ADDRESS	LOGICAL ADDRESS	SOURCE COL. ADDRESS	SOURCE LOG. ADDRESS	DESTINATION COL. ADDRESS	DESTINATION COL. ADDRESS	OPERATION
0	80			4	900	CALL SUBB
	800	15	63			CALL SUBA
	0	1	81			BRANCH BRN0
	800	6	166			CALL SUBA
	0	7	167			BRANCH BRN0
	0	4	788			BRANCH BRN0
1	81			0	0	BRANCH BRN0
4	788			0	0	BRANCH BRN0
	100	6	22			BRANCH BRN1
	900	11	715			CALL SUBB
	900	6	982			BRANCH SUBB
	900	0	80			CALL SUBB
5	949			7	983	BRANCH BRN4
6	22			4	100	BRANCH BRN1
	166			0	800	CALL SUBA
	934			7	983	BRANCH BRN4
	982			4	900	BRANCH SUBB
7	167			0	0	BRANCH BRN0
	983			RET SUBB		RET SUBB
	983	6	934			BRANCH BRN4
	983	5	949			BRANCH BRN4
9	825			11	859	BRANCH BRN3
11	27			12	700	BRANCH BRN2
	715			4	900	CALL SUBB
	859			RET SUBA		RET SUBA
	859	9	825			BRANCH BRN3
12	700	11	27			BRANCH BRN2
15	63			0	800	CALL SUBA

Table 5-3 Row address lookup table for the microprogram shown in table 5-1.

ROW ADDRESS	LOGICAL ADDRESS	SOURCE ROW ADDRESS	SOURCE LOG ADDRESS	DEST. ROW ADDRESS	DEST. LOG ADDRESS	OPERATION
0	715			55	900	CALL SUBB
	0	16	81			BRANCH BRN0
	0	37	167			BRANCH BRN0
	0	8	788			BRANCH BRN0
7	982			55	900	BRANCH SUBB
8	788			0	0	BRANCH BRN0
	983			RET SUBB		RET SUBB
	983	24	934			BRANCH BRN4
	983	39	949			BRANCH BRN4
14	859			RET SUBA		RET SUBA
	859	45	825			BRANCH BRN3
15	80			55	900	CALL SUBB
16	81			0	0	BRANCH BRN0
20	800	63	63			CALL SUBA
	800	36	166			CALL SUBA
22	22			35	100	BRANCH BRN1
24	934			8	983	BRANCH BRN4
27	27			50	700	BRANCH BRN2
35	100	22	22			BRANCH BRN1
36	166			20	800	CALL SUBA
37	167			0	0	BRANCH BRN0
39	949			8	983	BRANCH BRN4
45	825			14	859	BRANCH BRN3
50	700	27	27			BRANCH BRN2
55	900	15	80			CALL SUBB
	900	0	715			CALL SUBB
	900	7	982			BRANCH SUBB
63	63			20	800	CALL SUBA

For example, consider the first entry the column address lookup table (table 5-2). There is a subroutine call (CALL SUBB) at this column address. The logical address of the subroutine call is 80. There are also five microinstructions that non-sequentially converge to this address. The first one is CALL SUBA at column 15, logical address 63, and the logical address of the destination is 800. The second microinstruction is BRANCH BRN0 at column 1, logical address 81, and the destination is at logical address 0. The third microinstruction is CALL SUBA at column 6, logical address 166, and the destination is at logical address 800. The fourth and fifth microinstructions are BRANCH BRN0 from column 7, logical address 167, and column 4, logical address 788. Note that all non-sequential destination and source addresses of this column are identified.

5.2.3 ROW/COLUMN SHIFT REGISTERS WITH BUILT-IN SEQUENCING LOGIC

The row and column shift registers can be easily constructed using the hardwired microcontroller sequencing logic blocks and the address lookup tables. Each shift register can be constructed as follows

- Step 1**   An N-stage shift register is created with all stages disconnected from each other. N is the number of rows or columns that the shift register will drive. The first stage of the shift register is a RESET-TO-HIGH stage, all other stages are RESET-TO-LOW.
- Step 2**   A branch logic block, a call logic block, or a return logic block is placed wherever there is a branch microinstruction, a subroutine call, or a subroutine return.
- Step 3**   The input signals N and N+2 of all logic blocks are connected to the output of the appropriate shift register stages.
- Step 4**   For every branch block, every call block and every return block: if there is no other block of the same type at the same

address, connect the input signal EN to the appropriate system flag, BRANCH, CALL, or RETURN.

Step 5   The RET_C* input signals of all call blocks of each subroutine are connected to the RET_C* output signal of the return block of the same subroutine. Different subroutines must have different RET_C* lines.

Step 6   Where there are more than one logic blocks at an address, the N+1 signals output from these logic blocks must be ANDed as specified in rule 2'.

Step 7   All N+1 (already ANDed) , N+B, and N+C signals output from the logic blocks are connected to the appropriate stages of the shift register. If there are more than one input signals converging to a stage of the shift register, they must be ORed together as stated in rule 3'.

Step 8   For the remaining sequential stages, where there are no logic blocks connected to the them, connect the output of every stage to the input of the adjacent stage. If there is a signal already connected to the input of the shift register stage, the new input signal must be ORed with the existing one.

The clock signals, the reset signal, the system flags BRANCH, CALL, RETURN, and the EN+2 signals and the EN signals that have not been connected, are to be connected at the microcontroller level.

Figure 5-3 shows the logic model of the column shift register specified by the column address lookup table shown in table 5-2.

Figure 5-4 shows the logic model of the row shift register specified by the row address lookup table shown in table 5-3.

Figure 5-4 65-bit hardwired row shift register specified by the address
lookup table shown in table 5-3.

5.2.4 HARDWIRED MICROCONTROLLER

Once the microprogram memory and the row and column shift registers have been designed, the complete hardwired microcontroller can be constructed as shown in figure 5-5.

Figure 5-5 Block diagram of a 1K X 8-bit hardwired microcontroller.

For larger microcontrollers where word lines, bit lines and control lines may be too long to drive, driving buffers and sense amplifiers may be added as shown in figure 5-6.

Figure 5-6 Block diagram of a large hardwired microcontroller. Buffers and sense amplifiers are added to boost the word lines, bit lines and hardwired connections.

5.3 LAYOUT EXAMPLE

5.3.1 TECHNOLOGY

All layout examples shown in this thesis were designed using the 3 μ m technology specified in 'Guide to the Integrated Circuit Implementation Services (GICIS) of the Canadian Microelectronics Corporation (CMC), 3 micron single polysilicon, double metal P-well CMOS process (CMOS3 DLM)'.

Unless specified otherwise, the transistor sizes are as follows

		<u>Width (W)</u>	<u>Length (L)</u>
Non-driving Transistors	NMOS	6 μ m	3 μ m
	PMOS	12 μ m	3 μ m
Driving Transistor	NMOS	12 μ m	3 μ m
	PMOS	24 μ m	3 μ m

The layout of a 3 μ m CMC inverter standard cell with scale and annotations is shown in appendix C for references.

5.3.2 1K X 8 BIT HARDWIRED MICROCONTROLLER

5.3.2.1 HARDWIRED MICROCONTROLLER CELLS

A library of hardwired microcontroller cells: shift register stages, clock generator, branch block, call block and return block were created in order to layout the example 1K X 8 bit hardwired microcontroller. The layout of these cells and the hardwired microcontroller are shown in appendix B.

The shift register stages must be laid out in such a way that they can fit between pitches of the ROM, and sizing the transistors will not increase the height of the cell. To satisfy these constraints, transistors are laid out horizontally and two shift register stages are intertwined together to

form a shift register cell. There are two shift register cells: Reset-to-high, and Reset-to-low. The only difference between these two cells is that the first shift register stage of the Reset-to-high cell is a reset-to-high stage. All other shift register stages are reset-to-low.

The other hardwired microcontroller cells are built from scaled-down CMC inverter, NAND, NOR standard cells. These hardwired microcontroller cells are laid out so that they all have the same height, which makes it easier to run power lines. The height of these cells is not as critical as the height of the shift registers, however, it should be kept as short as possible, and independent with the transistor sizing. The layout of the scaled down CMC standard cells are shown in appendix C.

5.3.2.2 LAYOUT OF THE 1K X 8 BIT HARDWIRED MICROCONTROLLER

Figure 5-7 shows the layout of the column shift register specified by the column address lookup table shown in table 5-2. Only metal 1 and metal 2 are shown. The shift register is on the left with the column outputs going to the left. Next to the shift register are the OR gates and AND gates driving the inputs of the shift register. To the right of these gates are the sequencing logic blocks, BRANCHes, CALLs and RETURNs. There are seven EN and EN+2 signals to be connected to the appropriate rows: R16, R22, ..R64. The system flags BRANCH, CALL and RET are shown at the bottom right.

Figure 5-8 shows the layout of the row shift register specified by the row address lookup table shown in table 5-3. Also, only metal 1 and metal 2 are shown. The shift register is on the right with the row outputs going to the right. Next to the shift register are the OR gates and AND gates feeding the inputs of the shift register. To the left of these gates are the sequencing logic blocks, BRANCHes, CALLs and RETURNs. There are four EN and EN+2 signals to be connected to the appropriate column addresses: C0, C7, C1 and C12.

Figure 5-7 Layout of the column shift register specified by the column address lookup table shown in table 5-2. Only metal 1 and metal 2 are shown.

Figure 5-8 Layout of the row shift register specified by the row address lookup table shown in table 5-3. Only metal 1 and metal 2 are shown.

65 BIT ROW DECODER

1K x 8 BIT ASIC HARDWIRED MICROCONTROLLER

74

Figure 5-9 Layout of the 1K x 8-bit hardwired microcontroller specified by the microprogram shown in table 5-1. Only metal 1 and metal 2 are shown.

Figure 5-9 shows the complete layout of the 1K X 8 bit hardwired microcontroller with the microprogram summarized in table 5-1. The row decoder is located on the left, then the microprogram memory, then the column decoder. The EN and EN+2 signals of the column decoder are connected to the appropriate rows coming through the right side of the memory array. Similarly, the EN and EN+2 signals of the row decoder are connected to the appropriate columns coming through the left side of the bit-line multiplexing circuit. The clock generator is located on top of the row shift register. Driving buffers for the clock, and the reset signals going to the column shift register are placed on top of the shift register. Also a buffer is placed on the left of the bit-line multiplexing circuit to drive the output enable line OE.

This hardwired microcontroller was laid out and routed manually. It is not as compact as it can be, however, it is still a fair sample for the area comparison shown in chapter 7. The hardwired microcontroller will be compared against data taken from a conventional ROM, which was manually laid out and routed using the same constraints and parameters. The layout of this ROM is shown in appendix A.

CHAPTER 6

6. HARDWIRED MICROCONTROLLER, DESIGN CONSIDERATIONS

6.1 HARDWIRED MICROPROGRAM MEMORY REALIZATION

The hardwired microcontroller architecture is built on the assumption that only one row and one column can be active at a time. This property requires that the memory cell array is of the same structure as a ROM or a RAM memory cell array where only one word line and one bit multiplexing control line can be active at a time. PLA and PAL structures where more than one input address can be active at a time, are not suitable for realizing the microprogram memory for hardwired microcontrollers.

6.2 LENGTH OF THE ROW AND COLUMN SHIFT REGISTERS

The length of the row shift register and the column shift register directly dictate the length of the bit lines and the word lines. The longer the row shift register is, the longer the bit lines are. Similarly, the longer the column shift register is, the longer the word lines are. With the size of the memory array fixed, increasing the number of word lines will reduce the number of bit lines. The bit lines will be lengthened, and the word lines will be shortened. Since word lines are made of polysilicon which has much higher resistance and capacitance than metal, they should be kept as short as possible. To do so, the column shift register is usually kept much shorter than the row shift register. Keeping the column shift register short also keeps the number of bit lines low, and consequently keeps the bit-line multiplexing circuit simple. A complex bit-line multiplexing circuit requires more gates and more decoding time.

In addition to keeping the word lines short, the impedance of the word lines can be reduced by running metal lines in parallel with the polysilicon word lines and having these polysilicon lines and the metal

2

**PM-1 3½"x4" PHOTOGRAPHIC MICROCOPY TARGET
NBS 1010a ANSI/ISO #2 EQUIVALENT**

PRECISION^{SM} RESOLUTION TARGETS

PIONEERS IN METHYLENE BLUE TESTING SINCE 1974

1980 COUNTY ROAD 5, BURNINGWELL, MN 55337 USA
TEL 612 435 7867 FAX 612 435 7867 TLR 5102221000

lines connected together between groups of bit lines as shown in figure 6-1.

Figure 6-1 Layout of a memory cell array. Bit lines shown vertically are made of metal 1. Word lines shown non shaded and horizontally are made of polysilicon. Metal 2 lines shown shaded and horizontally are laid in parallel with the word lines to reduce the impedance of the word lines.

6.3 BRANCHES VERSUS SUBROUTINE CALLS

The microprogram structure and the programming style can significantly influence the complexity of a hardwired microprogram sequencer and the area it takes. Every branch instruction requires a BRANCH block and five wire connections. Every subroutine call requires a CALL block that is twice the size of the BRANCH block, and eight wire connections. A subroutine return microinstruction requires a RETURN block of almost the same size as the BRANCH block and five wire connections. With these constraints to work with, a hardwired microprogram should be designed with a minimum number of subroutine calls and branches. Not only the area but also the propagation delay of the hardwired microprogram sequencer could be reduced.

Also, for the purpose of reducing the area and increasing the speed for the microcontroller, branches are preferred over subroutine calls. However to make it easier for 'high level programming' minded programmers who love to use subroutine calls, one can design a smart hardwired microprogram compiler that transforms subroutine calls into sequential codes with branches. Particularly if the subroutines are very short or they are only called once or twice. This transformation may result in a larger microprogram, but it can significantly reduce the complexity and the area taken by the hardwired microprogram sequencer. It also increases the speed of the microprogram sequencer.

6.4 ENCODED WIRE CONNECTIONS

As shown in the layout example for the 1K X 8 bit hardwired microcontroller, the wire connections of the hardwired microprogram sequencer can take a significantly large space, approximately 50% of the total area taken by the hardwired microprogram sequencer. To reduce the space taken by these wire connections, long wires can be encoded and decoded as shown in figure 6-2. Note that in a hardwired microcontroller, only one row and one column can be active at a time. This guarantees that only one wire, branch or call or return, will be active at any given time. With this property, branch, call and return wires can be encoded to reduce the number of wires spanning across a long distance. Let M be the number of wires, branch or call or return, to be encoded, the number of encoded wires required is the integer number higher or equal to $\log_2(M)$.

Figure 6-2 Wires can be encoded to save space. Only $\log_2(M)$ wires instead of M wires span over a long distance.

6.5 MASK PROGRAMMING CONSIDERATIONS

In the layout example for the 1K X 8 bit hardwired microcontroller, the masks for all layers of the microcontroller are fully customized. Gates are moved around to produce the shortest combination of wire connections and the smallest space. Memory cell transistors are placed only as needed to keep the total power consumption and the propagation delay at minimum. This implementation gives the most area efficient design but it can be very expensive. The overhead cost to generate the masks is normally very high.

The number of masks may be reduced by designing a somewhat general purpose hardwired microcontroller using the same principles used in gate arrays as follows

First, a memory array is laid out with all memory cell transistors are placed but not connected to the word and bit lines. This is similar to a contact programmable mask ROM where the programming is done by placing contacts that connect the memory cell transistor to the corresponding word and bit lines. To implement a logic 0 at a memory location, three contacts are placed to connect the transistor at that location to the corresponding word and bit lines.

Once the memory array has been constructed, the shift registers and the clock generator can be placed. The outputs of the shift registers are connected to the appropriate rows and columns of memory array. The inputs of the shift registers are left disconnected. Also a fixed number of BRANCH, CALL and RETURN blocks, AND and OR gates are placed without any connections along each shift register as shown in figure 6-3. This configuration is similar to a blank gate array.

Assume that the preplanted general purposed hardwired microcontroller has a sufficiently large memory array, and enough sequencing logic blocks, and AND and OR gates. To customize the blank hardwired microcontroller for a specific microprogram, contacts are planted on the memory array to realize

the microcodes. Also metal lines are placed to connect the sequencing logic blocks to the appropriate stages of the shift registers. Only the masks for metal 1 and metal 2 are required. Obviously this implementation is not as area efficient as the fully customized implementation but the overhead cost is much lower. It also allows the microprogram to be revised for a relatively low cost.

Figure 6-3 A blank general purposed hardwired microprogram sequencer. Sequencing logic blocks, AND and OR gates are preplanted and left disconnected.

6.6 **TESTABILITY**

As VLSI circuits get more and more complex, testability has become one of the major issues in circuit design. Several design for testability techniques make use of shift registers to improve the controllability and observability of the circuit [WilT86]. Shift registers are employed in several self test techniques for imbedded memories. Examples of these techniques are the Easily Testable PLA Schemes: FK [FujH81], HO [HonS80], SKF [SalK83], K [KhaJ83]. The shift registers used as the row and column decoders for the hardwired microcontroller can also be used for testing as illustrated in Figure 6-4.

Assume that a linear static memory array is used for the hardwired microcontroller and there is only one shift register driving the word lines. An extra bit line is added to the memory array to indicate the parity of each word line. For each word line, a transistor is added to the extra bit line if the number of transistors on that line is even. For example, on word line 2, the number of transistors is 4, so a transistor is added to the extra bit line to make the number of transistors on this word line odd. All bit lines are XORed together to calculate the parity of all bit lines. All single cross point faults (the unintentional presence or absence of a transistor) can be detected by this circuit. To test the entire circuit, a logic 1 is shifted down from word 0 to word 15 and the test output is monitored. If there is a transistor not properly connected on a word line, or an additional transistor unintentionally connected to the word line, the parity for that word line will be different from the fault-free word line. All single stuck-at faults are also covered by this scheme. Note that this scheme can only detect single cross-point faults. If there are two or any even number of cross-point faults on the same word line, the parity of the word line will be the same as the fault free parity, and the faults will not be detected.

This built-in self test (BIST) hardwired microcontroller has two modes of testing, on-line testing and off-line testing. In the off-line mode, the hardwired microcontroller is forced into the sequential mode where a logic

1 is shifted through all stages of the shift register to verify the parity of all word lines. In the on-line mode, the microcontroller executes in its normal operation. However, the test circuit still monitors the parity of the word lines but only those executed during the normal operation.

Figure 6-4 A simple built-in self test hardwired microcontroller.

6.7 DISTRIBUTED HARDWIRED MICROCONTROLLERS

Figure 6-5 A simple example of a distributed hardwired microcontroller. Each slave controller has a SR flip flop at the last stage of the shift register to signal if the slave task is completed.

Synchronizing a distributed hardwired microcontroller can be very simple. All controllers must run from the same system clocks. A controller starts its operations when a logic 1 is shifted into the its shift registers, and goes idle when the logic 1 is shifted out of the shift registers. Figure 6-5 illustrates a simple example of a distributed hardwired microcontroller.

The distributed hardwired microcontroller consists of one master controller and three slave controllers. All controllers can have branches and subroutine calls but to simplify the example, only the sequential operation is shown. Each slave controller has a (Set/Reset) SR flip flop at the last stage of the shift register to signal if the slave task has been completed. The master controller starts all slave controllers then goes idle when it reaches stage 4. The output of stage 4 of the master controller is connected to the input of stage 0 of the slave controllers. When a slave controller finishes its task, it shifts out the logic 1 to the corresponding flip flop to signal that its task has been completed. Also when the logic 1 is shifted out of the shift register, the slave controller goes back to idle state. When all slave controllers finish their tasks, all flip flops output a logic 1 which restarts the master controller. When master controller has resumed its operations, all slave controllers' flip flops are reset.

6.8 DESIGN FLEXIBILITY

This is the main limitation of the hardwired microcontroller architecture and most custom-made integrated circuits. The hardwired microcontroller architecture is only for high volume production and only when the design has been well proven and is not expected to be changed. Changing in the microprogram would require changes in the microprogram memory and the hardwired microprogram sequencer which would require a whole new set of masks. However this costly problem may be avoided by designing the prototype using the conventional architecture with the exact same microprogram. Once the design has been proven, it can be easily transformed into a hardwired design for mass production.

CHAPTER 7

7. HARDWIRED MICROCONTROLLER VS. CONVENTIONAL MICROCONTROLLERS

7.1 AREA

7.1.1 HARDWIRED MICROCONTROLLER

Figure 7-1 Area partition of a hardwired microcontroller. There are three main partitions: memory array, row shift register and column shift register.

7.1.1.1 A HARDWIRED MICROCONTROLLER WITH 1K OF MICROINSTRUCTIONS AND 6 OUTPUT CONTROL BITS

Assume that two bits of the microinstruction are used for the opcode to indicate the type of the microinstruction: sequential, branch, subroutine call or subroutine return. With six bits required for the output control field, the width of the hardwired microinstruction or the number of memory output bits is

$$\begin{aligned} M &= \text{OP-CODE bits} + \text{Output control bits} \\ &= 2 \text{ bits} + 6 \text{ bits} = 8 \text{ bits.} \end{aligned}$$

From the layout example of the 1K X 8 bit hardwired microcontroller shown in chapter 5, the overall area of the hardwired microcontroller can be broken down into three main parts as shown in figure 7-1: the memory array, the row shift register, and the column shift register.

The memory array area is the total area of the precharge circuit, the memory cell array, and the bit-line multiplexing and sensing circuit. This area is linearly dependent on the number of microinstructions, N .

The memory array area is $1.59 \text{ mm} \times 2.70 \text{ mm} = 4.29 \text{ mm}^2$

The area of the row shift register is the total area of the shift register and its corresponding logic blocks and hardwired connections. Vertically, this area is linearly dependent on the number of rows, X . Horizontally, the row shift register is dependent on the complexity of the microprogram. The more complex the microprogram is, the more logic blocks are required and the more complicated the wire connections get. The width of the row shift register may be assumed to expand proportionally to $\log_2 N$.

The row shift register area is $1.59 \text{ mm} \times 1.20 \text{ mm} = 1.91 \text{ mm}^2$.

Similarly, the area of the column shift register is the total area of the shift register and its corresponding logic blocks and hardwired connections. This area expands vertically proportionally to the number of

columns, Y, and horizontally proportionally to $\log_2 N$.

The column shift register area is $0.96 \text{ mm} \times 1.20 \text{ mm} = 1.15 \text{ mm}^2$.

In real life applications, the microprogram is usually more complex than the example presented. Assume that the row shift register can be tightly packed into the same area, and the column shift register takes approximately the same area as the row shift register. The total area of the shift registers can be assumed to increase vertically proportionally to $(X+Y)$ and horizontally proportionally to $\log_2 N$.

The total area of the 1K X 8 bit hardwired microcontroller is

$$\begin{aligned} \text{Area} &= \text{Area (1K X 8 bit mem array)} + \text{Area (shift registers)} \\ &= 4.29 \text{ mm}^2 + 2 \times 1.91 \text{ mm}^2 \\ \text{Area (1K X 8 bit hardwired microcontroller)} &= 8.11 \text{ mm}^2 \end{aligned}$$

7.1.1.2 A HARDWIRED MICROCONTROLLER WITH 2K OF MICROINSTRUCTIONS AND 18 OUTPUT CONTROL BITS

Assuming that the number of columns remains the same, 16 columns. The number of rows is increased to $64 \times 2 + 1 = 129$ rows. Note that the number of rows is increased to 129 instead of 128 to make it indivisible by the number of columns.

The number of memory array output bits, M, is increased to 20 bits, 2 bits for the opcode and 18 bits for the output control signals.

The total area for a 2K X 20 bit hardwired microcontroller is

$$\begin{aligned} \text{Area} &= \text{Area(mem. array)} + \text{Area(shift registers)} \\ &= 4.29 \times ((129 \times 16) / (65 \times 16)) \times (20/8) + \\ &\quad 3.82 \times ((129+16) / (65+16)) \times (\log_2 2048 / \log_2 1024) \\ &= 21.29 + 7.52 = 28.81 \\ \text{Area (2K X 20 bit hardwired microcontroller)} &= 28.81 \text{ mm}^2 \end{aligned}$$

7.1.1.3 A HARDWIRED MICROCONTROLLER WITH N MICROINSTRUCTIONS AND m OUTPUT CONTROL BITS

Let Y be the number of columns and X+1 be the number of rows such that

$$X * Y = N$$

The number of memory array output bits is $M = m + 2$ with 2 bits for the opcode and m bits for the output control signals.

The total area of the hardwired microcontroller is

$$\begin{aligned} \text{Area} &= \text{Area}(\text{mem. array}) + \text{Area}(\text{shift registers}) \\ &= 4.29 * ((X+1) * Y / 1040) * ((m+2) / 8) + \\ &\quad 3.82 * ((X+1+Y) / 81) * (\log_2 N / \log_2 1024) [\text{mm}^2] \end{aligned}$$

The area of the memory array is increased in the order of $N * m$. The area of the shift registers is increased in the order of $(X+Y) * \log_2 N$ or $(X+Y) * \log_2 X + (X+Y) * \log_2 Y$ since $N = X * Y$.

7.1.2 CONVENTIONAL 1K X 8 BIT VIRTUAL GROUND ROM

The layout of a 1K X 8 bit conventional ROM using the same technology, the same parameters and the same constraints used for the 1K X 8 bit hardwired microcontroller is shown in figure 7-2 with only metal 1 and metal 2 are shown. Detailed layout drawings of this ROM are shown in appendix A.

The overall area is 1.59 mm X 3.09 mm = 4.91 mm². This area can be divided into three parts: the memory array, the row address decoder and the column address decoder.

$$\begin{aligned} \text{Area (memory array)} &= 1.59 \text{ mm} \times 2.70 \text{ mm} = 4.29 \text{ mm}^2 \\ \text{Area (row address decoder)} &= 1.22 \text{ mm} \times 0.39 \text{ mm} = 0.48 \text{ mm}^2 \\ \text{Area (column address decoder)} &= 0.37 \text{ mm} \times 0.39 \text{ mm} = 0.14 \text{ mm}^2. \end{aligned}$$

Let X be the number of rows and Y be the number of columns. The area of the row address decoder varies proportionally to $X \cdot \log_2 X$ and the area of the column address decoder varies proportionally to $Y \cdot \log_2 Y$. The area of the memory array increases proportionally to N*M, with N is the number of words and M is the number of the memory array output bits.

1K x 8 BIT CONVENTIONAL

Figure 7-2 Layout for a 1K X 8 bit conventional ROM.

9

7.1.3 CONVENTIONAL MICROCONTROLLER WITH NON-ENCODED MICROCODES

Figure 7-3 Area partition of a conventional microcontroller. There are four main partitions: memory array, row address decoder, column address decoder, and address supporting logic.

7.1.3.1 CONVENTIONAL MICROCONTROLLER WITH 1K OF NON-ENCODED MICROINSTRUCTIONS AND 6 OUTPUT CONTROL BITS

Assume that a conventional microcontroller with 1K of non-encoded microinstructions and 6 output control bits is to be designed. This microcontroller is equivalent to the 1K X 8 bit hardwired microcontroller.

Assume that the 8X02 microinstruction set shown in chapter 1 is used for this microcontroller. The opcode field consists of three bits to represent eight different types of microinstructions. The branch address field of microinstruction consists of ten bits to sufficiently address 1K of non-encoded microinstructions. The output control field of the microinstruction consists of six bits as required. The total number of bits in a microinstruction is

$M = 3 + 10 + 6 = 19$. The size of the required microprogram memory is 1K 19-bit words.

The area of the conventional microcontroller can be partitioned as shown in figure 7-3. There are four main partitions: the memory array, the row address decoder, the column address decoder and the address supporting logic.

The number of rows and the number of columns should be exactly the same as those of the 1K X 8 bit conventional ROM. Therefore the area of the address decoders should be exactly the same.

- The area of the row address decoder is 0.48 mm^2 .
- The area of the column address decoder is 0.14 mm^2 .

With the number of memory output bits increased to 19, the area of the memory array is increased to

$$\text{- Area(memory array)} = 4.29 \text{ mm}^2 * (19/8) = 10.19 \text{ mm}^2.$$

Assume that the address supporting logic takes approximately the same area as the total area of the address decoders. The area of the supporting logic can be assumed to increase proportionally to $(\log_2 N)^2$.

$$- \text{Area}(\text{address supporting logic}) = 0.48 \text{ mm}^2 + 0.14 \text{ mm}^2 = 0.62 \text{ mm}^2.$$

The total area of a conventional non-encoded microcontroller with 1K of non-encoded microcodes and 6 output control bits, is

$$\begin{aligned} \text{Area}(\text{microcontroller}) &= \text{Area}(\text{memory array}) \\ &\quad + \text{Area}(\text{row addr decoder}) + \text{Area}(\text{col addr decoder}) \\ &\quad + \text{Area}(\text{addr supporting logic}) \\ &= 10.19 + 0.48 + 0.14 + 0.62 [\text{mm}^2] \\ &= 11.43 \text{ mm}^2 \end{aligned}$$

The area of this conventional non-encoded microcontroller is 11.43 mm², 3.32 mm² or 41% larger than the equivalent hardwired microcontroller.

7.1.3.2 CONVENTIONAL MICROCONTROLLER WITH 2K OF NON-ENCODED MICROINSTRUCTIONS AND 18 OUTPUT CONTROL BITS

The total area of this conventional non-encoded microcontroller is 36.33 mm², 7.52 mm² or 26% larger than the equivalent hardwired microcontroller. Detailed calculations are shown in section 1 of appendix D.

7.1.3.3 CONVENTIONAL MICROCONTROLLER WITH N NON-ENCODED MICROINSTRUCTIONS AND m OUTPUT CONTROL BITS

In general, the total area of a conventional non-encoded microcontroller with N non-encoded microinstructions and m output control bits is

$$\begin{aligned} \text{Area} &= 4.29 \cdot (N/1024) \cdot ((3 + \log_2 N + m)/8) \\ &\quad + 0.48 \cdot (X/64) \cdot (\log_2 X/6) + 0.14 \cdot (Y/16) \cdot (\log_2 Y/4) \\ &\quad + 0.62 \cdot ((\log_2 N)^2/100) \end{aligned}$$

The area of the memory array is increased in the order of $N \cdot (m + \log_2 N)$. The area of the row address decoder is increased in the order of $X \cdot \log_2 X$. The area of the column address decoder is increased in the order of $Y \cdot \log_2 Y$. The area of the address supporting logic is increased in the order of $(\log_2 N)^2$. Detailed area calculations are shown in section 2 of appendix D.

7.1.4 CONVENTIONAL MICROCONTROLLER WITH ENCODED MICROCODES

7.1.4.1 CONVENTIONAL MICROCONTROLLER WITH 1.1K OF ENCODED MICROINSTRUCTIONS AND 6 OUTPUT CONTROL BITS

Assume that the non-encoded conventional microcontroller with 1K of microinstructions and 6 output control bits is redesigned using encoded microinstructions to eliminate the branch address field. Each branch, or subroutine call consists of two microinstructions. The first microinstruction specifies the opcode and the output control bits, just like other sequential microinstructions. Immediately following this microinstruction is the branch address. The 'branch address' microinstruction generally consists of an opcode field, and a branch address field which is used as the output control field in other microinstructions. Assume that the additional microinstructions used for the branch addresses increase the number of microinstructions by 10%. The total number of microinstructions is 1.1K words.

Assume that the 8X02 microinstruction set is used and the number of bits required for the opcode field is 3. The output control field of the microinstruction must have at least six bits as required. To sufficiently address 1.1K of microinstructions the branch address must have 11 bits, therefore the output control field must be increased to 11 bits. The total number of bits in the encoded microinstruction is $M = 4 + 11 = 15$. The size of the required microprogram memory is 1.1K or 1127 15-bit words.

Assume that the number of columns, Y, is the same, 16 columns. The area of the column address decoder stays the same at 0.14 mm^2 .

The number of rows, X, is increased to 71 so that

$$N = X*Y = 71*16 \text{ (words)} = 1136 \text{ words.}$$

The area of the row address decoder is

$$\begin{aligned} \text{Area(row address decoder)} &= 0.48*(71/64)*(\log_2(128)/\log_2(64)) \\ &= 0.62 \text{ mm}^2. \end{aligned}$$

With the number of memory output bits increased to 15, and the number of words increased by 10%, the area of the memory array is increased to

$$\text{Area}(\text{memory array}) = 4.29 \text{ mm}^2 * (1136/1024) * (15/8) = 8.92 \text{ mm}^2.$$

The total area for the 1.1K X 15 bit ROM is

$$\begin{aligned} \text{Area}(\text{ROM}) &= \text{Area}(\text{mem array}) + \text{Area}(\text{row decoder}) + \text{Area}(\text{col decoder}) \\ &= 8.92 \text{ mm}^2 + 0.62 \text{ mm}^2 + 0.14 \text{ mm}^2 = 9.68 \text{ mm}^2 \end{aligned}$$

Assume that the complexity of decoding the encoded microinstructions causes the area of the address supporting circuit to increase by 20%. Also, the area of the address supporting logic increases proportionally to $(\log_2 N)^2$.

$$\begin{aligned} \text{Area}(\text{address supporting logic}) &= 0.62 \text{ mm}^2 * 1.2 * ((\log_2 2048)^2 / (\log_2 1024)^2) \\ &= 0.90 \text{ mm}^2. \end{aligned}$$

The total area of the 1.1K X 15 bit conventional encoded microcontroller is

$$\begin{aligned} \text{Area}(\text{microcontroller}) &= \text{Area}(\text{ROM}) + \text{Area}(\text{address supporting logic}) \\ &= 9.68 \text{ mm}^2 + 0.90 \text{ mm}^2 = 10.58 \text{ mm}^2. \\ &= 10.58 \text{ mm}^2 \end{aligned}$$

The area of this conventional encoded microcontroller is 10.58 mm², 2.47 mm² or 30% larger than the equivalent hardwired microcontroller.

7.1.4.2 CONVENTIONAL MICROCONTROLLER WITH 2.2K OF ENCODED MICROINSTRUCTIONS AND 18 OUTPUT CONTROL BITS

The total area of this conventional encoded microcontroller is 28.61 mm², 0.20 mm² or 0.70% smaller than the equivalent hardwired microcontroller. Detailed calculations are shown in section 3 of appendix D.

7.1.4.3 CONVENTIONAL MICROCONTROLLER WITH 1.1N ENCODED MICROINSTRUCTIONS AND m OUTPUT CONTROL BITS

In general, the total area of a conventional encoded microcontroller with 1.1N encoded microinstructions and m output control bits is

CASE 1   The number of output control bits, m, is larger than the number of bits required to address 1.1N locations

$$\begin{aligned} \text{Area} = & 4.72*(N/1024)*((3+m)/8) \\ & + 0.53*(X/64)*((\log_2 X + 1)/6) + 0.14*(Y/16)*(\log_2 Y/4) \\ & + 0.74*((\log_2 N + 1)^2/100) \end{aligned}$$

The area of the memory array is increased in the order of $N*m$. The area of the row address decoder is increased in the order of $X*\log_2 X$. The area of the column address decoder is increased in the order of $Y*\log_2 Y$. The area of the address supporting logic is increased in the order of $(\log_2 N)^2$.

CASE 2   The number of output control bits, m, is smaller than the number of bits required to address 1.1N locations

$$\begin{aligned} \text{Area} = & 4.72*(N/1024)*((4+\log_2 N)/8) \\ & + 0.53*(X/64)*((\log_2 X + 1)/6) + 0.14*(Y/16)*(\log_2 Y/4) \\ & + 0.74*((\log_2 N + 1)^2/100) \end{aligned}$$

The area of the memory array is increased in the order of $N*\log_2 N$. The area of the row address decoder is increased in the order of $X*\log_2 X$. The area of the column address decoder is increased in the order of $Y*\log_2 Y$. The area of the address supporting logic is increased in the order of $(\log_2 N)^2$.

Detailed area calculations are shown in section 4 of appendix D.

7.1.5 AREA COMPARISON SUMMARY

	1K NON-ENCODED MICROCODES, 6 OUTPUT BITS	2K NON-ENCODED MICROCODES, 18 OUTPUT BITS
HARDWIRED CONTROLLER	8.11 mm ²	28.81 mm ²
NON-ENCODED CONVENTIONAL CONTROLLER Compared to the hardwired controller	11.43 mm ² 3.32 mm ² , 41% LARGER	36.33 mm ² 7.52 mm ² , 26% LARGER
ENCODED CONVENTIONAL CONTROLLER Compared to the hardwired controller	10.58 mm ² 2.47 mm ² , 30% LARGER	28.61 mm ² -0.20mm ² , 0.7% SMALLER

Table 7-1 Summary of the areas taken by different microcontrollers given the same number of non-encoded microcodes and the same number of output control bits.

Table 7-1 summarizes the areas taken by different microcontrollers given the same number of non-encoded microcodes and the same number of output control bits.

To implement a microcontroller with 1K of non-encoded microcodes and 6 output control bits, the hardwired microcontroller takes up an area of 8.11 mm². The equivalent conventional non-encoded microcontroller takes 11.43 mm², 41% larger than the hardwired microcontroller. The conventional encoded microcontroller takes 10.58 mm², 30% larger than the hardwired microcontroller.

To implement a microcontroller with 2K of non-encoded microcodes and 18 output control bits, the hardwired microcontroller takes up an area of 28.81 mm². The equivalent conventional non-encoded microcontroller takes 36.33 mm², 26% larger than the hardwired microcontroller. The conventional encoded microcontroller takes 28.61 mm², 0.7% smaller than the hardwired microcontroller.

	N NON-ENCODED MICROCODES, m OUTPUT BITS
HARDWIRED CONTROLLER	$4.29 * ((X+1) * Y / 1040) * ((m+2) / 8) +$ $3.82 * ((X+1+Y) / 81) * (\log_2 N / 10)$
NON-ENCODED CONVENTIONAL CONTROLLER	$4.29 * (N / 1024) * ((3 + \log_2 N + m) / 8) +$ $0.48 * (X / 64) * (\log_2 X / 6) + 0.14 * (Y / 16) * (\log_2 Y / 4) +$ $0.62 * ((\log_2 N)^2 / 100)$
ENCODED CONVENTIONAL CONTROLLER Case 1: $m \geq \log_2 N$	$4.72 * (N / 1024) * ((3 + m) / 8) +$ $0.53 * (X / 64) * ((\log_2 X + 1) / 6) + 0.14 * (Y / 16) * (\log_2 Y / 4) +$ $0.74 * ((\log_2 N + 1)^2 / 100)$
ENCODED CONVENTIONAL CONTROLLER Case 2: $m < \log_2 N$	$4.72 * (N / 1024) * ((4 + \log_2 N) / 8) +$ $0.53 * (X / 64) * ((\log_2 X + 1) / 6) + 0.14 * (Y / 16) * (\log_2 Y / 4) +$ $0.74 * ((\log_2 N + 1)^2 / 100)$

Table 7-2 Areas taken by different microcontrollers given the same number of non-encoded microcodes, N, and the same number of output control bits, m.

Table 7-2 generalizes the area taken by each microcontroller given the same number of non-encoded microcodes, N, and the same number of output control bits, m.

	MEMORY ARRAY	ADDR DECODERS/ SHIFT REGISTERS	ADDRESS SUPPORTING LOGIC
HARDWIRED CONTROLLER	$N*m$	$(X+Y)*\log_2 X$ + $(X+Y)*\log_2 Y$	
NON-ENCODED CONTROLLER	$N*(m+\log_2 N)$	$X*\log_2 X$ + $Y*\log_2 Y$	$(\log_2 N)^2$
ENCODED CONTROLLER $m \geq \log_2 1024*n$	$N*m$	$X*\log_2 X$ + $Y*\log_2 Y$	$(\log_2 N)^2$
ENCODED CONTROLLER $m < \log_2 1024*n$	$N*\log_2 N$	$X*\log_2 X$ + $Y*\log_2 Y$	$(\log_2 N)^2$

Table 7-3 Increasing order of the area taken by different microcontrollers.

Table 7-3 shows the increasing order of the areas taken by different microcontrollers with respect to the number of non-encoded microcodes, N , the number of rows, X , the number of columns, Y , and the number of output control bits, m .

The area taken by the memory array of the hardwired microcontroller increases proportionally to $N*m$. The memory array of the conventional non-encoded microcontroller increases proportionally to $N*(m+\log_2 N)$. The memory array of conventional encoded microcontroller increases proportionally to $n*m$ or $N*\log_2 N$ depending on whether the output control field (m) is large enough to accommodate the branch address ($\log_2 N$). Generally, the hardwired microcontroller memory array is the smallest and most area efficient. The memory array of the conventional encoded microcontroller is slightly larger because the conventional encoded microinstruction is wider and there are more microinstructions. The conventional microinstruction is wider because the conventional OP CODE is generally larger and some extra

bits may be added to the output control field to accommodate the branch address. The memory array of the conventional non-encoded microcontroller is the largest. Each non-encoded microinstruction has $\log_2 N$ more bits for the branch address. The area of the memory array is where the hardwired microcontroller architecture saves the most.

The area of the shift registers of the hardwired microcontroller increases in the order of $((X+Y)*\log_2 X + (X+Y)*\log_2 Y)$. The area of the address decoders of both types of the conventional microcontroller increases in the order of $(X*\log_2 X + Y*\log_2 Y)$. The area of the address supporting logic of a conventional microcontroller increases in the order of $(\log_2 N)^2$. The area taken by the hardwired shift registers is larger and increases slightly faster than the area taken by the conventional address decoders. However, the hardwired microcontroller does not require the address supporting logic. For a small number of microinstructions, $0 < N \leq 10K$, the area of the address supporting logic may not be large enough so the area taken by the hardwired shift registers is larger than the total area taken by the conventional address decoders and the address supporting logic. When the number of microinstructions is large enough, say $N > 10K$, the area of the conventional address supporting logic, which increases in the order of $(\log_2 N)^2$, may be large enough to offset the difference between the hardwired shift registers and the conventional address decoders. The area taken by the hardwired shift registers may be smaller than the total area taken by the conventional address decoders and the address supporting logic.

In summary, the hardwired microcontroller takes about the same area as the conventional encoded microcontroller, and much less than the conventional non-encoded microcontroller.

7.2 PERFORMANCE

7.2.1 NON-PIPELINED MICROCONTROLLERS

Figure 7-4 Block diagram of a non-pipelined conventional microcontroller.

NON-PIPELINED HARDWIRED MICROCONTROLLER

Figure 7-5

Block diagram of a non-pipelined hardwired microcontroller.

NON-PIPELINED MICROINSTRUCTION EXECUTION SEQUENCE

HARDWIRED MICROCONTROLLER

CONVENTIONAL MICROCONTROLLER

Clock edge

- Rows/columns output from the shift registers
- ¹\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$ <-----
- Memory array read
- Data operation
- OP-CODE and conditions decoded
- Hardwired connections arbitrated
- Shift registers input

Clock edge

Clock edge

- New address output from the address register
- Address decoded
- Memory array read
- Data operation/ PC <- PC + 1
- OP-CODE and conditions decoded
- Addresses multiplexed
- Address register loaded

Clock edge

¹ Time saving for the hardwired microcontroller.

Table 7-4 Microinstruction execution sequence of a non-pipelined hardwired microcontroller and a non-pipelined conventional microcontroller.

Figure 7-4 and figure 7-5 show the block diagrams of a non-pipelined conventional microcontroller and a non-pipelined hardwired microcontroller. The microinstruction execution sequence of these microcontrollers are shown in table 7-4.

Each microinstruction of the non-pipelined hardwired microcontroller is executed as follows

- First, the rows and columns are output from the shift register (following clock phase θ_2 going active).

- Second, the memory array outputs the corresponding microinstruction (OE going active).
- Third, the function of the controlled system driven by the microinstruction (data operation) is carried out.
- Fourth, after the data operation has been carried out, the branch conditions and the OP-CODE are evaluated.
- Fifth, all hardwired connections are arbitrated based on the decoded status signals, BRANCH, CALL and RETURN.
- Sixth, the shift registers latch in the new rows and columns (Θ_1 going active).

The microinstruction execution of the non-pipelined conventional microcontroller is similar to the hardwired microcontroller except that it requires an extra step to decode the address. Each microinstruction of the non-pipelined conventional non-encoded microcontroller is executed as follows

- First, the address is output from the address register.
- Second, the address is decoded.
- Third, the microinstruction is output from the memory array.
- Fourth, the function of the controlled system driven by the microinstruction (data operation) is carried out.
- Fifth, after the data operation has been carried out, the branch conditions and the OP-CODE are evaluated.
- Sixth, the addresses are multiplexed based on the OP-CODE and the branch conditions.

- Seventh, the address register latches in the new address.

The conventional microinstruction execution sequence shown in table 7-4 is only for non-encoded microcontrollers. Encoded microcontrollers usually require extra time, sometimes an extra clock cycle to decode the microinstruction (conditional branches), therefore are much slower than the non-encoded microcontrollers and the hardwired microcontrollers.

In general, a non-pipelined hardwired microcontroller is much faster than an equivalent non-pipelined conventional microcontroller. The time difference is the address decoding time required by the conventional microcontroller. Among non-pipelined conventional microcontrollers, a non-encoded microcontroller is much faster than the equivalent encoded microcontroller.

7.2.2 PIPELINED MICROCONTROLLER

Figure 7-6 Block diagram of a pipelined conventional microcontroller.

PIPELINED HARDWIRED MICROCONTROLLER

Figure 7-7

Block diagram of a pipelined hardwired microcontroller.

PIPELINED MICROINSTRUCTION EXECUTION SEQUENCE

HARDWIRED MICROCONTROLLER

CONVENTIONAL MICROCONTROLLER

Clock edge

Instruction 1

- Rows and columns
output from the
shift registers
- ¹ \$\$\$\$\$\$\$\$\$\$\$\$
- Memory array read
- Data register
loaded

Instruction 2

<-----

Clock edge

Instruction 1

- New address
output from the
address register
- Address decoded
- Memory array read
- Data register
loaded

Instruction 2

Clock edge

- Data register
output
- Data operation
- OP-CODE and
conditions decoded
- Hardwired
connections
arbitrated
- Shift registers
input

- Rows and columns
output from the
shift register
- \$\$\$\$\$\$\$\$\$\$\$\$
- Memory array read
- Data register
loaded

Clock edge

- Data register
output
- Data operation
- OP-CODE and
conditions decoded
- Address
multiplexed
- Address register
loaded

- New address
output from the
address register
- Address decoded
- Memory array read
- Data register
loaded

Clock edge

Clock edge

¹ Time saving for the hardwired microcontroller.

Table 7-5 Microinstruction execution sequence of a pipelined hardwired microcontroller and a pipelined conventional microcontroller.

Figure 7-6 and figure 7-7 show the block diagrams of a pipelined conventional microcontroller and a pipelined hardwired microcontroller. A register is added to the output of the memory array to allow prefetching microinstructions. The execution sequence of each pipelined microinstruction is divided into two phases: fetching and execution. The fetching phase of each microinstruction is overlapped with the execution phase of the previous microinstruction i.e. the next microinstruction is fetched while the current microinstruction is being executed. The microinstruction execution sequence of these pipelined microcontrollers are shown in table 7-5.

Microinstructions of the pipelined hardwired microcontroller are executed as follows

- On the first clock edge, the first microinstruction is fetched. The rows and columns are output from the shift registers. Then, the corresponding microinstruction is output and latched into the data register.
- On the second clock edge, two events happen simultaneously: the first microinstruction is executed and the second microinstruction is fetched. The data register outputs the latched microinstruction. The data operation is carried out followed by the branch condition decoding. The hardwired connections are arbitrated and the new rows and columns are latched into the shift registers. At the same time, the shift registers output the rows and columns. The new microinstruction is output and latched into the data register.
- On the third clock edge, the second microinstruction is executed and the third microinstruction is fetched.

Similarly, the microinstruction execution of the pipelined non-encoded conventional microcontroller is as follows

- On the first lock edge, the first microinstruction is fetched. The address is output from the address register and decoded. The corresponding

microinstruction is output and latched into the data register.

- On the second clock edge, the first microinstruction is executed and the second microinstruction is fetched simultaneously. The data register outputs the latched microinstruction. The data operation is carried out followed by the branch condition decoding. The addresses are multiplexed and the address of the third microinstruction is latched into the address register. At the same time, the address is output from the address register and decoded. The second microinstruction is output and latched into the data register.

- On the third clock edge, the second microinstruction is executed and the third microinstruction is fetched.

If the fetching time of the pipelined conventional microcontroller is greater than the execution time, the extra time required to decode the address will make the pipelined conventional microcontroller slower than the pipelined hardwired microcontroller. The clock period will be the time required to fetch a microinstruction. If not, both microcontrollers will run at the same speed and the clock period will be the time required to execute the microinstruction.

Similarly, if the fetching time of the pipelined conventional encoded microcontroller is greater than the execution time, the extra time required to decode the address will make the pipelined conventional encoded microcontroller much slower than the conventional non-encoded microcontroller and the hardwired microcontroller.

One major set back for all pipelined microcontrollers is that when a conditional branch microinstruction is executed, the next instruction address is not known until the branch conditions are evaluated. If the branch is true, the pipeline must be flushed and an extra clock cycle is required to fetch the branch address.

7.2.3 PERFORMANCE COMPARISON SUMMARY

In general, a non-pipelined hardwired microcontroller is almost as fast as a pipelined non-encoded conventional microcontroller. Also, the non-pipelined hardwired microcontroller does not need to flush the pipeline on conditional branches.

A non-pipelined hardwired microcontroller is much faster than an equivalent non-pipelined conventional microcontroller. The time difference is the address decoding time required by the conventional microcontroller.

A pipelined hardwired microcontroller is also faster than an equivalent pipelined conventional microcontroller if the fetching time of the pipelined conventional microcontroller is greater than the execution time. If not, the two microcontrollers will run at the same speed.

Among the conventional microcontrollers, an encoded conventional microcontroller requires a much smaller space than a non-encoded microcontroller, however it is much slower than the non-encoded microcontroller due to the extra time required to decode and fetch the address.

CHAPTER 8

8. CONCLUSIONS AND FUTURE WORK

8.1 CONCLUSIONS

8.1.1 THESIS SUMMARY

The thesis has reviewed the basic architecture and characteristics of the conventional microcontroller architecture and the conventional ROM. After the background was presented, the hardwired microcontroller architecture was introduced. The principles of operations and the basic sequencing blocks, branch, subroutine call and return, and their usage were explained in details. An example implementation of a hardwired microcontroller was then presented to illustrate the principles of the hardwired microcontroller architecture. From the example, several design issues of the hardwired microcontroller architecture were discussed. A detailed analysis comparing the hardwired microcontroller and the equivalent conventional microcontroller was also presented. The first part of this chapter summarizes the advantages, disadvantages and limitations of the hardwired microcontroller architecture. The remainder of the thesis discusses future work that can be done to examine other issues such as distributed controllers, testability, and a hardwired microcontroller IC synthesis program.

8.1.2 ADVANTAGES OF HARDWIRED MICROCONTROLLERS

SPEED

The most obvious advantage of the hardwired microcontroller architecture and other hardwired architectures is the speed. In general, a non-pipelined hardwired microcontroller is almost as fast as an equivalent pipelined conventional microcontroller. A non-pipelined hardwired microcontroller also does not need to flush the pipeline upon a

conditional branch as required by pipelined conventional microcontrollers. A non-pipelined hardwired microcontroller can generate a condition code, test and branch in the same clock cycle.

A non-pipelined hardwired microcontroller is much faster than an equivalent non-pipelined conventional microcontroller. The major time saved is in the address decoding that is eliminated in the hardwired microcontroller.

A pipelined hardwired microcontroller is about the same or faster than an equivalent pipelined conventional microcontroller.

AREA

The hardwired microcontroller architecture also produces area efficient designs. Generally, a hardwired microcontroller takes about the same area as the equivalent conventional encoded microcontroller, and much less than the conventional non-encoded microcontroller. The major area saved is in the memory array. The hardwired microinstruction is narrower than the conventional microinstruction.

Overall, the hardwired microcontroller architecture gives the best combination for area and speed. A hardwired microcontroller generally takes the least area but runs fastest. An encoded conventional microcontroller may take less space than the equivalent hardwired microcontroller, however, it is much slower than the hardwired microcontroller.

TESTABILITY

Various design for testability techniques make use of shift registers to test imbedded memories. Potentially, the shift registers used as the row and column decoders of the hardwired microcontroller may also be modified and used to test the microprogram memory.

DISTRIBUTED CONTROLLERS

Hardwired microcontrollers may also be advantageous for distributed controllers. Hardwired microcontrollers are fast and simple to synchronize. A hardwired microcontroller can be started and stopped by simply shifting a logic 1 in or out of the shift registers.

8.1.3 DISADVANTAGES OF HARDWIRED MICROCONTROLLERS

INFLEXIBILITY

One obvious disadvantage of the hardwired microcontroller architecture and most customized hardwired architectures is the inflexibility. The hardwired microcontroller can not be constructed until the microprogram is designed and well proven. Changes in the microprogram can result to changes in several masks which can be very expensive. The microprogram memory can not be erased and reprogrammed easily as in conventional microcontrollers, which use EPROM.

This disadvantage may not present any setback if the prototype microcontroller is designed and tested using the conventional non-encoded architecture. Once the microprogram has been well proven, the conventional non-encoded prototype can be transformed to a hardwired microcontroller for maximum speed and minimum area.

The flexibility for a hardwired microcontroller can also be improved with the generic 'gate array like' hardwired microcontroller architecture. However, there is an area trade off. The generic hardwired microcontroller can take more area than the fully customized hardwired microcontroller.

POOR FOR COMPLEX BRANCH STRUCTURES

Another major disadvantage that the hardwired microcontroller architecture suffers is that, the complexity of the microprogram can increase the area and slightly reduce the speed of the microcontroller. With the

microprogram sequencing implemented by hardwired connections and logic, the hardwired microcontroller gains speed and reduces the overall area. However, each conditional branch or subroutine call or return microinstruction requires a sequencing logic block and some wire connections. With a microprogram sloppily written, the number of branches and subroutine calls can be unnecessarily high which can significantly increase the area and slightly increase the propagation delay in sequencing the microprogram.

8.2 FUTURE WORK

8.2.1 TESTABILITY FOR HARDWIRED MICROCONTROLLERS

The memory array is the largest component of a microcontroller and therefore it is the major factor that determines the test circuit for the microcontroller. Different realizations of the memory array require different built-in test circuits. For a linear memory array, the hardwired microcontroller can have a very simple and small built-in self test (BIST) circuit as shown in chapter 6. For a virtual ground dynamic memory array, the BIST circuit is more complex and beyond the scope of this thesis. Potentially, the shift registers used in a hardwired microcontroller as row and column decoders can be modified and used in the BIST circuit to apply test patterns or to monitor the test results. To have a better evaluation on the testability of the hardwired microcontroller architecture, a BIST circuit should be designed for a hardwired microcontroller with a virtual ground dynamic memory array. Another BIST circuit should also be designed for an equivalent conventional microcontroller. Then the two circuits could be compared to determine which architecture is more testable.

8.2.2 DISTRIBUTED HARDWIRED MICROCONTROLLERS

Distributed microcontroller is another issue that is also complex and beyond the scope of this thesis. A simple example of a master-slave distributed hardwired microcontroller is given in chapter 6. It seems that

synchronizing hardwired microcontroller can be very simple. A logic 1 shifting into the shift registers starts the corresponding hardwired microcontroller, and the logic 1 shifting out of the shift registers stops the hardwired microcontroller. However there is not enough data to conclude whether a distributed hardwired microcontroller is better than the equivalent conventional distributed microcontroller or not. Further study could help determining if it is worth while to design a hardwired distributed microcontroller instead of a conventional distributed microcontroller.

8.2.3 'GATE ARRAY LIKE' GENERIC HARDWIRED MICROCONTROLLER

In chapter 6, a 'gate array like' generic hardwired microcontroller configuration was presented. The purpose of this configuration is to reduce the number of masks required to customize a hardwired microcontroller and consequently reduces the overhead cost to customize a hardwired microcontroller. The user needs to program only metal 1 and metal 2 masks to customize a hardwired microcontroller. Revising the design can also be easier and less costly. The configuration should be tried out with real life applications to determine accurately the best combinations of sequencing logic blocks, and AND and OR gates to be placed in the generic hardwired microcontroller. Also from these examples, the proportion of space wasted by this generic configuration can be formulated for future design considerations.

8.2.4 HARDWIRED MICROCONTROLLER IC SYNTHESIS

To make the task of designing a hardwired microcontroller easier, a hardwired microcontroller synthesis program that reads the microprogram and generates the logic model for the hardwired microcontroller should be designed. The synthesis program would first reads the microprogram to check for syntax errors and to generate the address lookup tables. Then it would place the shift registers, the sequencing logic blocks, AND and OR gates according to the lookup tables. And finally, all shift registers, sequencing logic blocks, and AND and OR gates would be connected according

to the lookup tables and the hardwired microcontroller rules. The sequence of steps to construct a hardwired microcontroller is illustrated in chapter 6.

8.2.5 LINEAR HARDWIRED MICROCONTROLLER

A model for linear hardwired microcontrollers should also be designed and evaluated. A linear hardwired microcontroller is generally smaller than the presented hardwired microcontroller (X-Y controller). The difference between a linear hardwired microcontroller and the 'X-Y' hardwired microcontroller is that the linear controller has only one shift register. Branching and especially subroutine calls for a linear hardwired microcontroller are much simpler. Clocking and timing are also much simpler.

Linear hardwired microcontrollers are usually suitable for applications where the microprogram is very wide and short. The microprogram memory can be implemented as a linear memory array which can be driven by a single shift register.

CHAPTER 9

9. REFERENCES

- [AleN84] Nikitas A. Alexandridis, *"Microprocessor System Design Concepts,"* Computer Science Press, 1984.
- [ChaY78] Yiu-Fai Chan, "A 4K CMOS Erasable PROM," *IEEE J. Solid-State Circuits*, Vol. SC-13, No. 5, pp. 677-680, October 1978.
- [CupR85] R Cuppens et al., "An EEPROM for microprocessors and custom logic," *IEEE J. Solid-State Circuits*, Vol. SC-20, pp. 603-608, 1985.
- [DaeW81] W. Daen and J. Mucha, "A Hardware Approach to Self-Testing of Large Programmable Logic Arrays," *IEEE Trans. Computers* vol. C-30, pp. 829-833, Nov. 1981.
- [FleH75] H. Fleisher and L. I. Maissel, "An Introduction to Array Logic," *IBM J. Res. Develop*, vol. 19, pp.98-109, March 1975.
- [FleW80] William I. Fletcher, *"An Engineering Approach to Digital Design,"* Prentice-Hall, New Jersey, 1980.

- [FujH81] H. Fujiwara and K. Kinoshita, "A Design of Programmable Logic Arrays with Universal Tests," *IEEE Trans. Computers*, vol. C-30, pp. 823-828, Nov. 1981.
- [GlaL88] Lance A. Glasser & Daniel W. Dobberpuhl, "*The Design and Analysis of VLSI Circuits*," Addison-Wesley, 1988.
- [HasS83] S. Z. Hassan and E. J. McCluskey, "Testing PLA's Using Multiple Parallel Signature Analyzers," *Dig. 13th International Symposium on Fault-Tolerant Computing*, pp. 422-425, June 1983.
- [HonS80] S. J. Hong and D. L. Ostapko, "FITPLA: A Programmable Logic Array for Functional Independent Testing," *Dig. 10th International Symposium on Fault-Tolerant Computing*, pp. 131-136, Oct. 1980.
- [IwaY90] Yoshihisa Iwata et al., "A High-Density NAND EEPROM with Block-Page Programming for Microcomputer Applications," *IEEE J. Solid-State Circuits*, Vol. 25, No. 2, pp. 417-424, April 1990.
- [KazR86] R. Kazerounian and B. Eitan, "A single poly EPROM for custom CMOS logic applications," *Proc. IEEE 1986 Custom Integrated Circuits Conf.*, pp. 59-62, 1986.

- [KhaJ83] J. Khakbaz, "A Testable PLA Design with Low Overhead and High Fault Coverage," *Digest 13th International Symposium on Fault-Tolerant Computing*, pp. 426-429, June 1983.
- [KurM90] Masao Kuriyama et al., "A 16-ns 1-Mb CMOS EPROM," *IEEE J. Solid-State Circuits*, Vol. 25, No. 5, pp. 1141-1146, October 1990.
- [MavJ83] J. Mavor, M. A. Jack and P. B. Denyer, "Introduction to MOS LSI Design," Addison-Wesley, 1983.
- [MomM89] Masaki Momodomi et al., "An Experimental 4-Mbit CMOS EEPROM with a NAND-Structured Cell," *IEEE J. Solid-State Circuits*, Vol. 24, No. 5, pp. 1238-1243, 1989.
- [RamK83] K. S. Ramanatha and N. N. Biswas, "A Design for Testability of Undetectable Crosspoint Faults in PLAs," *IEEE Trans. Computers*, vol. C-32, pp. 551-557, June 1983.
- [SalK83] K. K. Saluja, K. Kinoshita and H. Fujiwara, "An Easily Testable Design of Programmable Logic Arrays for Multiple Faults," *IEEE Trans. Computers*, vol. C-32, pp. 1038-1046, Nov. 1983.

- [TanA76] Andrew S. Tanenbaum, *"Structured Computer Organization,"* Prentice-Hall, New Jersey, 1976.
- [TreR85] R. Treuer, H. Fujiwara and V. K. Agarwal, "Implementing a Built-in Self-Test PLA Design," *Digest 15th International Symposium on Fault-Tolerant Computing* June 1985.
- [WesN88] Neil West & Kamran Eshraghian, *"Principles of CMOS VLSI Design, A Systems Perspective,"* Addison-Wesley, 1988.
- [Wilt86] T. Williams (Editor), *"VLSI Testing,"* Elsevier Science Publishers B.V. (North-Holland), 1986.

Appendix A

LAYOUT EXAMPLE OF A 1K X 8 Bit ROM

Figure A1 Block diagram of a 1K X 8 bit conventional ROM.

Figure A2 Schematic diagram of a 20 row X 16 column virtual ground dynamic memory array.

Figure A3 Layout of the precharge transistors for a group of 9 bit lines.

Figure A4 Layout of a row of 8 memory cells.

Figure A5 Layout of an 8 X 8 memory cell array with only metal 1, poly and active regions are shown.

Figure A6 Layout of an 8 X 8 memory cell array with only metal 1, metal 2 and poly are shown.

Figure A7 Bit lines sensing and multiplexing, schematic diagram.

Figure A8 Bit lines sensing and multiplexing, floor planning.

Figure A9 Bit lines sensing and multiplexing, layout with metal 1 lines shown vertically and metal 2 lines shown horizontally.

Figure A10 Bit lines sensing and multiplexing, layout with only metal 1 and contacts are shown.

Figure A11 Bit lines sensing and multiplexing, layout with only metal 2 and via are shown.

Figure A12 Bit lines sensing and multiplexing, layout of transistors 1 to 12 with metal 2 shown not shaded.

Figure A13 Bit lines sensing and multiplexing, layout of transistors 1 to 12 with only metal 1, metal 2 and via are shown.

Figure A14 Bit lines sensing and multiplexing, layout of transistors and gates 49 to 74 with metal 2 shown not shaded.

Figure A15 Bit lines sensing and multiplexing, layout of transistors and gates 49 to 74 with only metal 1, metal 2 and via are shown.

4-16DEC.DWG

Figure A16 Column address decoder, 4-to-16, schematic diagram.

Figure A17 Column address decoder, 4-to-16, floor planning.

Figure A18 4-input-decoder cell, layout.

Figure A19 Column address decoder, 4-to-16, layout.

Figure A20 Row address decoder, 6-to-64, schematic diagram.

6 INPUT DECODER CELL

6-40000 D16

Figure A21 6-input-decoder cell, floor planning.

Figure A22 6-input-decoder cell, layout.

+-----+

Figure A23 Row address decoder, 6-to-64, floor planning.

Figure A24 Row address decoder, 6-to-64, layout.

6-TO-64 ADDRESS DECODER

Figure A24 Row address decoder, 6-to-64, layout.

1K x 8 BIT CONVENTIONAL ROM

146

Figure A25 Conventional 1K x 8 bit ROM, layout with only metal 1 and metal 2 are shown.

Appendix B

LAYOUT EXAMPLE OF A 1K X 8 BIT HARDWIRED MICROCONTROLLER

Figure B1 Block diagram of a 1K X 8 bit hardwired microcontroller.

Figure B2 Floor planning for row/column shift-register stages 1 and 2.

Figure B3 Layout for row/column decoder shift-register stages 1 and 2.

Figure B4 Layout for row/column decoder shift-register stages 1 and 2 with only metal 1, metal 2, via and contacts are shown.

Figure B5 Floor planning for two RESET-TO-LOW shift-register stages.

Figure B6 Layout for two RESET-TO-LOW shift-register stages.

Figure B7 Layout for two RESET-TO-LOW shift-register stages with only metal 1. metal 2, via and contacts are shown.

Figure B9 Hardwired microcontroller clock generator, layout.

Figure B10 Hardwired microcontroller clock generator, layout with only metal 1, metal 2 and via are shown.

Figure B11 Hardwired microcontroller branch block, schematic, truth table and floor planning.

Figure B12 Hardwired microcontroller branch block, layout.

Figure B13 Hardwired microcontroller branch block, layout with only metal 1, metal 2 and via are shown.

Figure B14 Hardwired microcontroller call block, floor planning.

Figure B15 Hardwired microcontroller call block, layout.

Figure B16 Hardwired microcontroller call block, layout with only metal 1, metal 2 and via are shown.

Figure B17 Hardwired microcontroller return block, schematic and floor planning.

Figure B18 Hardwired microcontroller return block, layout.

Figure B19 Hardwired microcontroller return block, layout with only metal 1.
metal 2 and via are shown.

Figure B20 Hardwired microcontroller column shift register, schematic.

Figure B21 Hardwired microcontroller shift register column decoder, layout with only metal 1 and metal 2 are shown.

16 BIT COLUMN DECODER

10-DEC (9/4/13)

SHIFT - REGISTERS

AND, OR - GATES

3

of/de

3

PM-1 3½"x4" PHOTOGRAPHIC MICROCOPY TARGET
NBS 1010a ANSI/ISO #2 EQUIVALENT

PRECISION^{SM} RESOLUTION TARGETS

PIONEERS IN METHYLENE BLUE TESTING SINCE 1974

MicroD

10000 COUNTY ROAD 2, BURNINGWELL, NEW BRISTOL, USA
TEL 012 436 7867 FAX 012 436 7867 TLM 510000048

3 of/de 3

PM-1 3 1/2" x 4" PHOTOGRAPHIC MICROCOPY TARGET
NBS 1010a ANSI/ISO #2 EQUIVALENT

PRECISION® RESOLUTION TARGETS

lth

168
A6
A22
A37
A24
A87
A8
A64

Figure B22 Hardwired microcontroller shift register row decoder, schematic.

Figure B23 Hardwired microcontroller shift register row decoder. Layout with only metal 1 and metal 2 are shown.

65 BIT ROW DECODER

171

Appendix C

**LAYOUT OF THE 3 μ m CMC INVERTER STANDARD CELL, AND
THE SCALED DOWN INVERTER, NAND, NOR STANDARD CELLS**

Figure C1 Layout of the 3 um CMC Inverter Standard Cell.

Figure C2 Layout of the scaled down non-driving inverter standard cell.

Figure C3 Layout of the scaled down driving inverter standard cell.

Figure C4 Layout of the scaled down NAND gate standard cell.

Figure C5 Layout of the scaled down NOR gate standard cell.

Appendix D

AREA CALCULATIONS FOR CONVENTIONAL MICROCONTROLLERS

APPENDIX D

Figure D-1 Area partition of a conventional microcontroller. There are four main partitions: memory array, row address decoder, column address decoder, and address supporting logic.

D.1 CONVENTIONAL MICROCONTROLLER WITH 2K OF NON-ENCODED MICROINSTRUCTIONS AND 18 OUTPUT CONTROL BITS

To address 2K of microcodes, the branch address field of the microinstruction is required to have $\log_2(2*1024)$ bits = 11 bits. Assume that 3 bits are used for the opcode and 18 bits are used for the output control field, the number of bits in a microinstruction is

$$M = (3 + 11 + 18) \text{ bits} = 32 \text{ bits.}$$

Assume that the number of columns remains the same. The area of the column address decoder is 0.14 mm^2 .

The number of rows is increased to 128. The area of the row address decoder is

$$\begin{aligned} \text{Area}(\text{row address decoder}) &= 0.48 * (128/64) * (\log_2(128)/\log_2(64)) \\ &= 1.12 \text{ mm}^2. \end{aligned}$$

With the number of words increased to 2K and the number of memory output bits increased to 32, the area of the memory array is increased to

$$\begin{aligned} \text{Area (memory array)} &= 4.29 \text{ mm}^2 * (2048/1024) * (32/8) \\ &= 34.32 \text{ mm}^2. \end{aligned}$$

The total area for the 2K X 32 bit ROM is

$$\begin{aligned} \text{Area(ROM)} &= \text{Area(mem array)} + \text{Area (row decoder)} + \text{Area(col decoder)} \\ &= 34.32 \text{ mm}^2 + 1.12 \text{ mm}^2 + 0.14 \text{ mm}^2 \\ &= 35.58 \text{ mm}^2 \end{aligned}$$

The area of the address supporting logic increases proportionally to $(\log_2 N)^2$.

$$\begin{aligned} \text{Area(address supporting logic)} &= 0.62 \text{ mm}^2 * ((\log_2 2048)^2 / (\log_2 1024)^2) \\ &= 0.75 \text{ mm}^2. \end{aligned}$$

The total area of the 2K X 32 bit conventional non-encoded microcontroller

is

$$\begin{aligned}\text{Area}(\text{microcontroller}) &= \text{Area}(\text{ROM}) + \text{Area}(\text{address supporting logic}) \\ &= 35.38 \text{ mm}^2 + 0.75 \text{ mm}^2\end{aligned}$$

$$\text{Area (2K X 32 bit conventional non-encoded microcontroller)} = 36.33 \text{ mm}^2$$

Let delta be the difference between the area of the 2K X 32 bit conventional non-encoded microcontroller and the equivalent (2K X 20 bit) hardwired microcontroller.

$$\begin{aligned}\text{Delta} &= \text{Area}(2\text{K X 32 bit conventional non-encoded microcontroller}) \\ &\quad - \text{Area}(2\text{K X 20 bit hardwired microcontroller}) \\ &= 36.33 \text{ mm}^2 - 28.81 \text{ mm}^2 = 7.52 \text{ mm}^2\end{aligned}$$

The conventional non-encoded microcontroller is 7.52 mm² or 26% larger than the equivalent hardwired microcontroller.

D.2 CONVENTIONAL MICROCONTROLLER WITH N NON-ENCODED MICROINSTRUCTIONS AND m OUTPUT CONTROL BITS

Let X be the number of rows and Y be the number of columns such that

$$X * Y = N$$

The number of memory array output bits is $M = (3 + \log_2 N + m)$ with 3 bits for the opcode, $\log_2 N$ bits for the branch address, and m bits for the output control signals.

The total area of the hardwired microcontroller is

$$\begin{aligned} \text{Area} &= \text{Area}(\text{mem. array}) \\ &+ \text{Area}(\text{row decoder}) + \text{Area}(\text{column decoder}) \\ &+ \text{Area}(\text{address supporting}) \\ &= 4.29 * (N/1024) * ((3 + \log_2 N + m)/8) \\ &+ 0.48 * (X/64) * (\log_2 X / \log_2 64) + 0.14 * (Y/16) * (\log_2 Y / \log_2 16) \\ &+ 0.62 * ((\log_2 N)^2 / (\log_2 1024)^2) \\ &= 4.29 * (N/1024) * ((3 + \log_2 N + m)/8) \\ &+ 0.48 * (X/64) * (\log_2 X / 6) + 0.14 * (Y/16) * (\log_2 Y / 4) \\ &+ 0.62 * ((\log_2 N)^2 / 100) \end{aligned}$$

The area of the memory array is increased in the order of $N * (m + \log_2 N)$. The area of the row address decoder is increased in the order of $X * \log_2 X$. The area of the column address decoder is increased in the order of $Y * \log_2 Y$. The area of the address supporting logic is increased in the order of $(\log_2 N)^2$.

D.3 CONVENTIONAL MICROCONTROLLER WITH 2.2K OF ENCODED MICROINSTRUCTIONS AND 18 OUTPUT CONTROL BITS

Assume that the non-encoded conventional microcontroller with 2K of microinstructions and 18 output control bits is redesigned using encoded microinstructions to eliminate the branch address field.

Assume that the additional microinstructions used for the branch addresses increase the number of microinstructions by 10%. The total number of encoded microinstruction is 2.2K or 2253 words.

Assume that the 8X02 microinstruction set is used and the opcode field requires 3 bits. To sufficiently address 2.2K of microinstructions, it is required have 12 bits for the branch address. The output control field consists of 18 bits which is enough for representing the branch address therefore the total number of bits in the encoded microinstruction is $M = 4 + 18 = 22$. The size of the required microprogram memory is 2.2K 22-bit words.

Assume that the number of columns remains the same at 16 and the area of the column address decoder stays the same at 0.14 mm^2 .

To address 2253 words, the number of rows must be increased to 141 rows. The area of row address decoder expands vertically proportionally to the number of rows and horizontally proportionally to $\log_2$ of the number of rows. The area of the row address decoder is

$$\begin{aligned} \text{Area}(\text{row address decoder}) &= 0.48 * (141/64) * (\log_2(256)/\log_2(64)) \\ &= 1.41 \text{ mm}^2. \end{aligned}$$

With the number of memory output bits increased to 22, and the number of rows increased to 141, the area of the memory array is increased to

$$\text{Area}(\text{memory array}) = 4.29 \text{ mm}^2 * (141/64) * (22/8) = 25.99 \text{ mm}^2.$$

The total area for the 2.2K X 22 bit ROM is

$$\text{Area}(\text{ROM}) = \text{Area}(\text{mem array}) + \text{Area}(\text{row decoder}) + \text{Area}(\text{col decoder})$$

$$= 25.99 \text{ mm}^2 + 1.41 \text{ mm}^2 + 0.14 \text{ mm}^2 = 27.54 \text{ mm}^2$$

With the number of address lines increased to 12, the area of the address supporting logic increases to

$$\text{Area}(\text{address supporting logic}) = 0.74 \text{ mm}^2 * (12^2/10^2) = 1.07 \text{ mm}^2.$$

The total area of the 2.2K X 22 bit conventional encoded microcontroller is

$$\begin{aligned} \text{Area}(\text{microcontroller}) &= \text{Area}(\text{ROM}) + \text{Area}(\text{address supporting logic}) \\ &= 27.54 \text{ mm}^2 + 1.07 \text{ mm}^2 = 28.61 \text{ mm}^2. \end{aligned}$$

$$\text{Area (2.2K, 22 bit output, conventional encoded microcontroller)} = 28.61 \text{ mm}^2$$

Let delta be the difference between the area of the 2.2K X 22 bit conventional encoded microcontroller and the area of the equivalent 2K X 20 bit hardwired microcontroller.

$$\begin{aligned} \text{Delta} &= \text{Area}(2.2\text{K X } 22 \text{ bit conventional encoded microcontroller}) \\ &\quad - \text{Area}(\text{equivalent } 2\text{K X } 20 \text{ bit hardwired microcontroller}) \\ \text{Delta} &= 28.61 \text{ mm}^2 - 28.81 \text{ mm}^2 = -0.20 \text{ mm}^2 \end{aligned}$$

The 2.2K X 22 bit conventional encoded microcontroller is 0.20 mm² or 0.70% smaller than the equivalent 2K X 20 bit hardwired microcontroller.

D.4 CONVENTIONAL MICROCONTROLLER WITH 1.1N ENCODED MICROINSTRUCTIONS AND m OUTPUT CONTROL BITS

Let X be the number of rows and Y be the number of columns such that

$$X * Y = N$$

N is the number of equivalent non-encoded microinstructions.

Assume that by encoding the microinstructions, the number of microinstructions is increased by 10%. The total number of encoded microinstructions is 1.1N.

Assume that 3 bits are required for the OP CODE field.

CASE 1   **The number of output control bits, m, is larger than the number of bits required to address 1.1N locations**

Assume that the number of output control bits is larger than the number of bits required to address 1.1N locations. The number of memory array output bits is $M = 3 + m$.

The total area of the encoded conventional microcontroller is

$$\begin{aligned} \text{Area} &= \text{Area}(\text{mem. array}) \\ &+ \text{Area}(\text{row decoder}) + \text{Area}(\text{column decoder}) \\ &+ \text{Area}(\text{address supporting}) \\ &= 4.29 * (1.1N/1024) * ((3+m)/8) \\ &+ 0.48 * (1.1X/64) * ((\log_2 X + 1)/\log_2 64) + 0.14 * (Y/16) * (\log_2 Y / \log_2 16) \\ &+ 0.74 * ((\log_2 N + 1)^2 / (\log_2 1024)^2) \\ &= 4.72 * (N/1024) * ((3+m)/8) \\ &+ 0.53 * (X/64) * ((\log_2 X + 1)/6) + 0.14 * (Y/16) * (\log_2 Y / 4) \\ &+ 0.74 * ((\log_2 N + 1)^2 / 100) \end{aligned}$$

The area of the memory array is increased in the order of $N * m$. The area of

the row address decoder is increased in the order of $X \cdot \log_2 X$. The area of the column address decoder is increased in the order of $Y \cdot \log_2 Y$. The area of the address supporting logic is increased in the order of $(\log_2 N)^2$.

CASE 2   The number of output control bits, m , is smaller than the number of bits required to address $1.1N$ locations

Assume that the number of output control bits is less than the number of bits required to address $1.1N$ locations. The number of memory array output bits is $M = 3 + \log_2 2N$.

The total area of the encoded conventional microcontroller is

$$\begin{aligned}
 \text{Area} &= \text{Area}(\text{mem. array}) \\
 &\quad + \text{Area}(\text{row decoder}) + \text{Area}(\text{column decoder}) \\
 &\quad + \text{Area}(\text{address supporting}) \\
 \\
 \text{Area} &= 4.29 \cdot (1.1N/1024) \cdot ((3 + \log_2 2N)/8) \\
 &\quad + 0.48 \cdot (1.1X/64) \cdot ((\log_2 X + 1)/\log_2 64) + 0.14 \cdot (Y/16) \cdot (\log_2 Y / \log_2 16) \\
 &\quad + 0.74 \cdot ((\log_2 N + 1)^2 / (\log_2 1024)^2) \\
 \\
 &= 4.72 \cdot (N/1024) \cdot ((4 + \log_2 N)/8) \\
 &\quad + 0.53 \cdot (X/64) \cdot ((\log_2 X + 1)/6) + 0.14 \cdot (Y/16) \cdot (\log_2 Y / 4) \\
 &\quad + 0.74 \cdot ((\log_2 N + 1)^2 / 100)
 \end{aligned}$$

The area of the memory array is increased in the order of $N \cdot \log_2 N$. The area of the row address decoder is increased in the order of $X \cdot \log_2 X$. The area of the column address decoder is increased in the order of $Y \cdot \log_2 Y$. The area of the address supporting logic is increased in the order of $(\log_2 N)^2$.

END

2|3·1 2·92

FIN