Air Force Institute of Technology

AFIT Scholar

Theses and Dissertations

Student Graduate Works

3-2002

Microdot - A Four-Bit Microcontroller Designed for Distributed Low-End Computing in Satellites

Anthony R. Woodcock

Follow this and additional works at: https://scholar.afit.edu/etd

Part of the Space Vehicles Commons, and the VLSI and Circuits, Embedded and Hardware Systems Commons

Recommended Citation

Woodcock, Anthony R., "Microdot - A Four-Bit Microcontroller Designed for Distributed Low-End Computing in Satellites" (2002). *Theses and Dissertations*. 4465.
https://scholar.afit.edu/etd/4465

This Thesis is brought to you for free and open access by the Student Graduate Works at AFIT Scholar. It has been accepted for inclusion in Theses and Dissertations by an authorized administrator of AFIT Scholar. For more information, please contact richard.mansfield@afit.edu.

MICRODOT – A FOUR-BIT MICROCONTROLLER DESIGNED FOR DISTRIBUTED LOW-END COMPUTING IN SATELLITES

THESIS

Anthony R. Woodcock, Captain, USAF

AFIT/GE/ENG/02M-28

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 074-0188

The public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of the collection of information, including suggestions for reducing this burden to Department of Defense, Washington Headquarters Services, Directorate for Information Operations and Reports (0704-0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to an penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number.
PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From – To)*
15-03-2002	**Master's Thesis**	Jun 2001 – Mar 2002

4. TITLE AND SUBTITLE

MICRODOT – A FOUR-BIT MICROCONTROLLER DESIGNED FOR DISTRIBUTED LOW-END COMPUTING IN SATELLITES

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Woodcock, Anthony R., Captain, USAF

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAMES(S) AND ADDRESS(S)

Air Force Institute of Technology
Graduate School of Engineering and Management (AFIT/EN)
2950 P Street, Building 640
WPAFB OH 45433-7765

8. PERFORMING ORGANIZATION REPORT NUMBER

AFIT/GE/ENG/02M-28

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

AFRL/VSSE
Attn: James C. Lyke
3550 Aberdeen Ave. SE
Kirtland AFB, NM 87117-5776
DSN: 246-5812
e-mail: James.Lyke@kirtland.af.mil

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

13. SUPPLEMENTARY NOTES

Lt Col James A. Lott, PHD
DSN: 785-3636 ext. 4576, Email: James.Lott@afit.edu

14. ABSTRACT

Many satellites are an integrated collection of sensors and actuators that require dedicated real-time control. For single processor systems, additional sensors require an increase in computing power and speed to provide the multi-tasking capability needed to service each sensor. Faster processors cost more and consume more power, which taxes a satellite's power resources and may lead to shorter satellite lifetimes. An alternative design approach is a distributed network of small and low power microcontrollers designed for space that handle the computing requirements of each individual sensor and actuator. The design of Microdot, a four-bit microcontroller for distributed low-end computing, is presented. The design is based on previous research completed at the Space Electronics Branch, Air Force Research Laboratory (AFRL/VSSE) at Kirtland AFB, NM, and the Air Force Institute of Technology at Wright-Patterson AFB, OH. The Microdot has 29 instructions and a 1K x 4 instruction memory. The distributed computing architecture is based on the Philips Semiconductor I^2C Serial Bus Protocol. A prototype was implemented and tested using an Altera Field Programmable Gate Array (FPGA). The prototype was operable to 9.1 MHz. The design was targeted for fabrication in a radiation-hardened-by-design gate-array cell library for the TSMC 0.35 micrometer CMOS process.

15. SUBJECT TERMS

Microprocessor, VLSI, Radiation Hardening, Satellite, Low Power, Microelectronics

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Lt Col James A. Lott, PHD
					19b. TELEPHONE NUMBER *(Include area code)*
U	U	U	UU	143	(937) 255-3636, ext 4576; e-mail: James.Lott@afit.edu

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std. Z39-18

The views expressed in this thesis are those of the author and do not reflect the official policy or position of the United States Air Force, Department of Defense, or the U.S. Government.

AFIT/GE/ENG/02M-28

MICRODOT – A FOUR-BIT MICROCONTROLLER DESIGNED FOR DISTRIBUTED LOW-END COMPUTING IN SATELLITES

THESIS

Presented to the Faculty

Department of Electrical and Computer Engineering

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

In Partial Fulfillment of the Requirements for the

Degree of Master of Science in Electrical Engineering

Anthony R. Woodcock, BS

Captain, USAF

March 2002

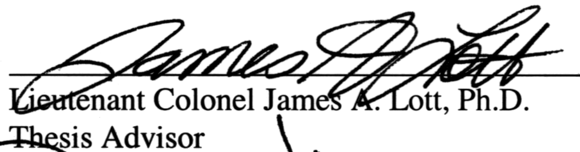
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

AFIT/GE/ENG/02M-28

MICRODOT – A FOUR-BIT MICROCONTROLLER DESIGNED FOR DISTRIBUTED LOW-END COMPUTING IN SATELLITES

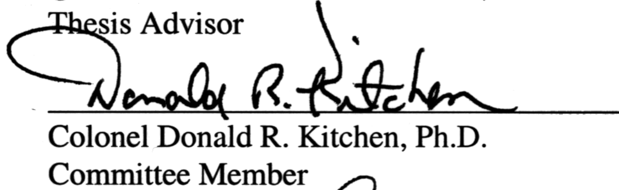
Anthony R. Woodcock, BS
Captain, USAF

Approved:



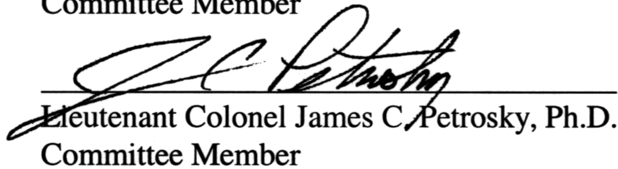
Lieutenant Colonel James A. Lott, Ph.D.
Thesis Advisor

5 MARCH 2002
Date



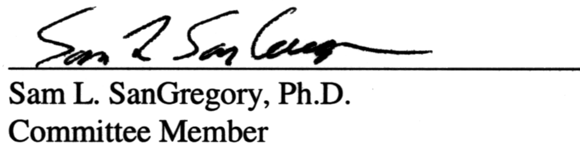
Colonel Donald R. Kitchen, Ph.D.
Committee Member

5 MARCH 2002
Date



Lieutenant Colonel James C. Petrosky, Ph.D.
Committee Member

05 MAR 02
Date



Sam L. SanGregory, Ph.D.
Committee Member

05 Mar 02
Date

Acknowledgements

I would like to thank my thesis advisor, Lieutenant Colonel Jim Lott, for his support in helping me complete this thesis. Thanks also to Colonel Don Kitchen, Lieutenant Colonel Jim Petrosky, and Dr. Sam SanGregory for serving on my committee.

Thank you to Jim Lyke and the Air Force Research Laboratory Space Electronics Branch (AFRL/VSSE) for supporting my research and funding my design work with .

I would also like to thank Dan King, Jeff Black, Dave Alexander and the rest of the engineers at Mission Research Corporation, Microelectronics Division in Albuquerque, NM for assisting me with the radiation-hardened VLSI design libraries.

Thanks to Greg Richardson for keeping the VLSI computer lab up and running. Also, thanks to Dr. Clint Kohl at Cedarville University for helping me get started with Altera FPGAs and Charlie Powers for helping me to get AFIT set up with the Altera University Program.

Most importantly, I would like to thank the late Lieutenant Colonel Charles Brothers for teaching me VLSI design and getting me started with this thesis.

Table of Contents

Acknowledgements iv

Table of Contents v

List of Figures viii

List of Tables xi

Abstract xii

1. Introduction 1-1

  1.1 Introduction 1-1
  1.2 Problem Statement 1-1
  1.3 Methodology 1-3
  1.4 Scope 1-3
  1.5 Summary of Results 1-4
  1.6 Overview 1-5

2. Literature Review 2-1

  2.1 Introduction 2-1
  2.2 VLSI Design Flow 2-1
  2.3 Stack Processors 2-5
  2.4 Serial Bus Protocols 2-7
    2.4.1 Candidate Serial Bus Protocols 2-8
    2.4.2 I^2C Serial Bus Protocol 2-9
  2.5 I^2C Data Transfer 2-13
  2.6 Radiation Hardening of Electronics 2-15
    2.6.1 Space Radiation Environment 2-16
    2.6.2 Total Ionizing Dose 2-17
    2.6.3 Single Event Effects 2-20
      2.6.3.1 Single Event Upset 2-20
      2.6.3.2 Single Event Latchup 2-23
    2.6.4 Radiation Hardening By Design 2-24
    2.6.5 Radiation Hardened By Design Gate Array Cell Library 2-26
  2.7 Summary 2-30

3. Design Overview 3-1

3.1 Introduction 3-1
3.2 Design Goals 3-1
3.3 Microdot Design 3-2
3.4 Instruction Set 3-3
3.5 Control Unit 3-6
3.6 Arithmetic and Logic Unit 3-9
3.7 Stack Unit 3-12
3.8 Input/Output Unit 3-13
3.9 Memory Unit 3-15
3.10 I²C Communication Unit 3-16
3.11 Status Multiplexer 3-19
3.12 Summary 3-20

4. Design Implementation 4-1

4.1 Introduction 4-1
4.2 State Machine Design 4-1
4.3 Gated Clocking 4-3
4.4 Control Unit 4-4
4.5 Arithmetic and Logic Unit 4-6
4.6 Stack Unit 4-9
4.6.1 Stack Unit Sub-blocks 4-11
4.6.2 Stack Buffer Registers 4-13
4.7 Input/Output Unit 4-15
4.8 Memory Unit 4-17
4.8.1 Memory Unit Sub-blocks 4-17
4.8.2 Memory Buffer Register 4-20
4.9 I²C Communication Unit 4-21
4.9.1 I²C Commands 4-26
4.9.1.1 Reading the Stack (READ STACK command) 4-28
4.9.1.2 Writing Output Data (READ DATA command) 4-30
4.9.1.3 Programming (PROGRAM command) 4-32
4.9.2 Interrupt 4-35
4.10 Microdot Top Level 4-36
4.11 Microdot Production Design 4-38
4.12 Summary 4-38

5. Testing and Analysis 5-1

5.1 Introduction 5-1
5.2 Altera Overview 5-1
5.3 Behavioral VHDL Testing 5-4
5.4 Structural VHDL Testing 5-5
5.5 Full FPGA Testing 5-5

5.6 I²C Driver 5-6
5.7 FPGA Results 5-7
5.7.1 FPGA Timing 5-7
5.7.2 FPGA Power Consumption 5-8
5.8 RHBD Gate Array Results 5-10
5.8.1 RHBD Gate Array Timing 5-11
5.8.2 RHBD Gate Array Memory Design 5-12
5.9 Summary 5-13

6. Conclusions and Future Work 6-1

6.1 Summary and Accomplishments 6-1
6.2 Conclusions and Lessons Learned 6-2
6.3 Future Research 6-4

Appendix A. Microdot Functional Blocks and Signals A-1

Appendix B. Microdot Design Summary – Radiation Hardened By Design Gate Array Library B-1

Appendix C. Microdot Chip Level Interconnection Diagrams C-1

Appendix D. List of Abbreviations D-1

References R-1

Vita VITA-1

List of Figures

Figure Page

1-1. Distributed Processing with the Microdot....1-2

2-1. VLSI Design Process: Y-Diagram....2-2

2-2. VLSI Linear Design Flow....2-3

2-3. I^2C Serial Bus Concept....2-10

2-4. I^2C Bit Transfer....2-11

2-5. I^2C START and STOP Conditions....2-11

2-6. I^2C Data Transfer....2-13

2-7. Van Allen Radiation Belts....2-16

2-8. Transistor Ionization Process....2-18

2-9. Normalized N-Channel MOSFET I-V Curve....2-19

2-10. Single Particle Strike....2-21

2-11. Six Transistor SRAM Cell....2-22

2-12. CMOS PNPN Structure....2-23

2-13. Bird's Beak Diagram....2-25

2-14. Radiation Hardened By Design Gate Array Cell....2-27

2-15. AOI22 (AND-OR-INVERT) gate in RHBD Gate Array....2-28

2-16. NAND Gate Size Comparison....2-29

3-1. Microdot Distributed Computing Concept....3-2

3-2. Microdot Block Diagram....3-3

3-3. Control Unit Interface....3-7

3-4. Instruction State Diagram....3-8

3-5. ALU Interface 3-9

3-6. Stack Unit Interface 3-12

3-7. Input/Output Unit Interface 3-13

3-8. Memory Unit Interface 3-15

3-9. I²C Communication Unit Interface 3-17

3-10. Status Multiplexer Interface 3-20

4-1. Gated Clocking versus Continuous Clocking 4-4

4-2. Control Unit Interconnection Diagram 4-5

4-3. Arithmetic and Logic Unit Interconnection Diagram 4-7

4-4. Stack Unit Interconnection Diagram 4-10

4-5. Stack Timing Errors During PUSH Instruction 4-14

4-6. SWAP Instruction Timing 4-15

4-7. Input/Output Unit Interconnection Diagram 4-16

4-8. Memory Unit Interconnection Diagram 4-18

4-9. Memory Buffer Register Timing Diagram 4-21

4-10. I²C Communication Unit Interconnection Diagram 4-23

4-11. COM Word State Machine State Transition Diagram 4-26

4-12. Read Command Cycle 4-28

4-13. I²C READ STACK Command Cycle 4-30

4-14. READ DATA Command Cycle 4-31

4-15. Programming Example to Wait for Input Data 4-32

4-16. I²C Program Cycle 4-34

4-17. Continuation of Program Cycle 4-34

4-18. Microdot Interface Diagram 4-36
4-19. Microdot Interconnection Diagram 4-37
4-20. Microdot Production Version Pinout 4-38
5-1. Altera UP-1 Design Laboratory Package 5-2
5-2. Altera FLEX 10K Logic Element 5-3
5-3. Microdot FPGA Test Setup 5-6
C-1. Microdot Test Chip Diagram (1 of 3) C-1
C-2. Microdot Test Chip Diagram (2 of 3) C-2
C-3. Microdot Test Chip Diagram (3 of 3) C-3

List of Tables

Table Page

2-1. Representative Sample of the Four-bit Microcontroller Market........................2-14

3-1. Microdot Instruction Set..3-5

3-2. Microdot Number Representation...3-10

3-3. ALU Overflow Conditions..3-11

4-1. State Machine Design Comparison..4-2

4-2. Status Bit Output by ALU Instruction..4-9

4-3. Microdot I^2C Commands 4-27

5-1. FPGA Timing Results 5-8

5-2. Microdot FPGA Supply Current, Power Dissipation, and Performance.............5-9

5-3. RHBD Microdot Design Summary...5-10

5-4. RHBD Gate Array Timing Results 5-11

A-1. Microdot Design Hierarchy...A-1

A-2. Microdot Signal Table...A-2

AFIT/GE/ENG/02M-28

Abstract

As satellites become more complex, the on-board processing capabilities must keep up. Many satellites are an integrated collection of sensors and actuators with many requiring dedicated real-time control to operate correctly. For single processor systems, adding more sensors requires an increase in computing power and speed to provide the multi-tasking capability needed to service each sensor. Faster processors are more costly and consume more power, which can tax a satellite's power resources and may lead to shorter satellite lifetimes. Commercial-Off-The-Shelf (COTS) electronic components are usually not acceptable for satellite design because they have not been hardened against the radiation environment of space. An alternative design approach is to use a distributed network of small and low power microcontrollers designed for space to handle the computing requirements of each individual sensor and actuator. The design of microdot, a four-bit microcontroller for distributed low-end computing, is presented. The design is based on previous research completed at the Space Electronics Branch, Air Force Research Laboratory (AFRL/VSSE) at Kirtland AFB, NM, and the Air Force Institute of Technology at Wright-Patterson AFB, OH. The Microdot has 29 instructions and a 1K x 4 instruction memory. The distributed computing architecture is based on the Philips Semiconductor I^2C Serial Bus Protocol. A prototype was implemented and tested using an Altera Field Programmable Gate Array (FPGA). The prototype was operable up to 9.1 MHz. The design was also targeted for fabrication using a radiation-hardened-by-design gate-array library from Mission Research Corporation. The gate-array library is designed for the TSMC 0.35 micrometer CMOS process.

MICRODOT – A FOUR-BIT MICROCONTROLLER DESIGNED FOR DISTRIBUTED LOW-END COMPUTING IN SATELLITES

1. Introduction

1.1 Introduction

In this research I design and implement a very large scale integrated (VLSI) circuit for the purpose of interfacing and controlling sensors and actuators in space applications. This research extends previous work done at the Air Force Institute of Technology [1] and the Air Force Research Laboratory Space Electronics Branch (AFRL/VSSE) [2,3]. I present the design process of "Microdot", a small and specialized 4-bit microcontroller, from the initial concepts to circuit layout and testing. I designed this application specific integrated circuit (ASIC) for fabrication with a radiation-tolerant gate array standard-cell library for the Taiwan Semiconductor Manufacturing Company (TSMC) 0.35 micrometer process. The design was implemented in an Altera [22] field programmable gate array (FPGA).

1.2 Problem Statement

Satellites must operate with limited power budgets and in a space radiation environment that is detrimental to the reliability of semiconductor micro-electronics. A hierarchical satellite control and data network architecture with VLSI Microdot microcontrollers at the lowest level can both reduce power consumption and improve reliability of a satellite's control and sensing electronics.

In a conventional design, a satellite uses a central microprocessor to interface to the system sensors and actuators. The microprocessor must continuously run while polling

sensors to collect data or adjust actuators. This leads to high power consumption. If a central processor can delegate control and data collection to a network of low-power Microdot microcontrollers, it only needs to operate and consume power when transferring

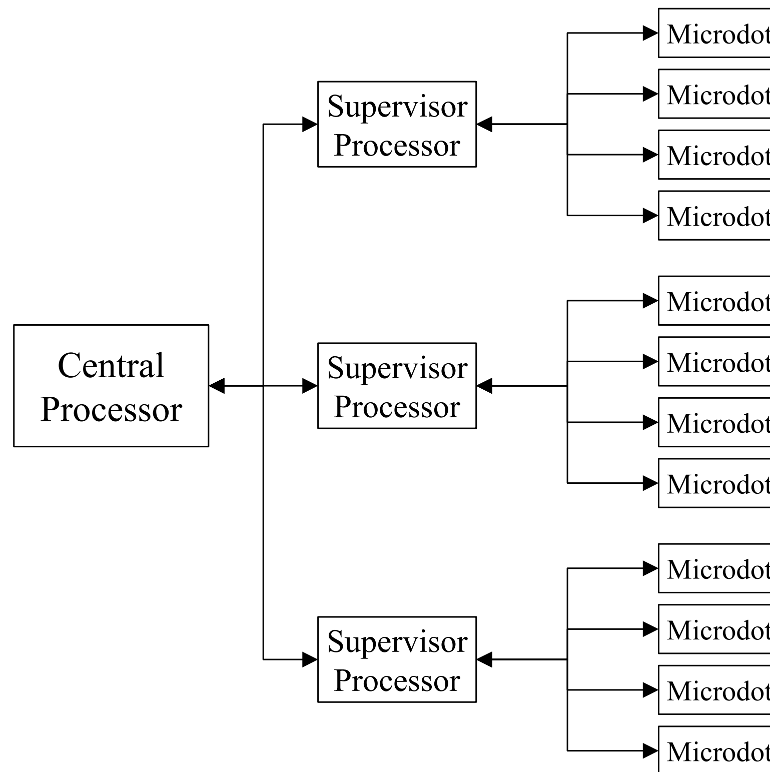


Figure 1-1. Distributed Processing with the Microdot. Processing tasks are delegated to the lowest possible level (Microdot). This reduces the processing workload for the central processor.

data or programming a Microdot. Thus, it can use a standby mode to conserve power or perform other functions when not needed. This is the concept illustrated in Figure 1-1.

The Microdot also provides a high level of versatility to the system design as it is designed to be easily reprogrammed by a central or supervisor processor. As an example, a Microdot interfacing a temperature sensor could be programmed to detect temperatures that fluctuate more than five degrees from a target temperature and only report back to the central

processor when that happens. If a five degree margin proved too small, the central processor could reprogram the Microdot for the same task with a temperature margin of ten degrees.

Delegating control and data collection also may reduce the computing power needed for the central processor. Because it eliminates polling, it eliminates the clock cycles used for executing polling routines and also eliminates the power used to execute the extra clock cycles. Taking the Microdot from this concept to an actual integrated circuit requires a design methodology.

1.3 Methodology

The design methodology for ASICs is the process of taking a functional concept from its initial behavioral definition to its physical layout and finally testing and design verification. Circuit functionality is first described in a Hardware Definition Language (HDL) only by its behavior. This can occur before any decision is made on what particular semiconductor technology will be used. Next, logic synthesis converts the behavioral description to logic gates. Physical layout is the process of defining the placement of the logic gates and their interconnects. The layout can then be fabricated in a semiconductor process. The final step is verifying the design and checking for errors in the physical layout. The design process is further explained in Chapter Two.

1.4 Scope

For this thesis, my goal was to design, implement, and test the Microdot microcontroller. I used the VLSI design tools that included Synopsys Design Analyzer and VHDL Simulator to create the behavioral and structural models of the Microdot. I also used Altera field programmable gate array software and hardware to implement and test the

behavioral and structural models of the Microdot microcontroller. My research was limited to the design of the Microdot processor core. While I present the Microdot as part of a larger distributed processing system, it was beyond the scope of this project to develop or demonstrate an example of the distributed computing system.

The size of the memory is what dominates the overall size of the Microdot. This research effort was primarily concerned with the design of the Microdot core and did not focus on the memory design. Consequently, designing compact, radiation-hardened static random access memories (SRAM) requires an additional research and design effort.

1.5 Summary of Results

The final Microdot design successfully realizes the concept of a small distributed computing element that can provide simple data collection and control operations for a microdevice. The Microdot is a four-bit microcontroller with 29 instructions. It has a program memory size of 1024x4 and a data memory size of 128x8. I implemented the Microdot design in an Altera field programmable gate array (FPGA) and successfully tested it to a maximum operating frequency of 9.1 MHz. The power consumption was computed at 48.5 mW per MHz.

The Microdot was also designed for radiation-hardened by design (RHBD) gate array standard-cell library and simulated to a maximum operating frequency of 23.3 MHz. The Microdot core in the RHBD cell library requires 782 logic gates, which requires an area (without pads) of 1.84 mm^2 (1.44 x 1.28 mm).

1.6 Overview

This thesis is comprised of six chapters. Chapter One provides an overview of the design problem including the motivation, scope, and design methodology for the Microdot ASIC.

Chapter Two covers the integrated circuit design process, serial bus protocols for distributed computing, and basic information on four-bit microcontrollers. It also covers radiation effects on integrated circuits and the design process for making VLSI circuits radiation-tolerant.

In Chapter Three I present my Microdot design. I present function and reason for my design decisions. The main element of the chapter is the instruction set, which defines the processing capability of the Microdot. I also explain the circuit architecture and major functional elements of the design.

Chapter Four covers important design approaches and goes in depth into the design of each of the major functional elements. The functionality and interface of each of the six functional blocks is explained in detail by considering the operation of each sub-block. Also in this chapter, I detail the operation and design of the I^2C serial bus interface that gives the Microdot its distributed computing capability.

In Chapter Five I explore the simulation and testing process and results for each design level. The behavioral and structural description were defined and simulated using the Very High Speed Integrated Circuit Hardware Description Language (VHDL). The Microdot design was implemented in an Altera field programmable gate array (FPGA). The FPGA was used for design verification and testing because cost and time constraints prevented VLSI fabrication.

In Chapter Six, I conclude the thesis and present the overall results and conclusions for the Microdot design. I also cover the lessons learned and make recommendations for further research and development of the Microdot and distributed low-end computing and microdevice interfacing.

2. Literature Review

2.1 Introduction

This chapter presents background research for the design of radiation-tolerant application-specific integrated circuits (ASICs). It briefly covers the VLSI/ASIC design process and then looks at design specific issues including stack processing and serial bus protocols. It also includes the current landscape of 4-bit microcontrollers, radiation effects on CMOS circuits, and techniques for radiation hardening by design.

2.2 VLSI Design Flow

The number of individual transistors and their integration into complex logic functions make a VLSI design difficult to accomplish without good computer-aided design (CAD) tools and a very structured and incremental design process. Also, with constantly improving VLSI manufacturing capability, the design process must be fast, yet able to allow designs to retain compatibility as semiconductor technology evolves. Figures 2-1 and 2-2 diagram the VLSI design process. Figure 2-1 shows the Y-chart, which illustrates the three development domains and the design hierarchy levels. Figure 2-2 shows the traditional linear design approach. There are three domains of circuit design: Behavioral, Structural, and Layout. The traditional linear approach is to first design and test the behavior, then move to structural design, and finally circuit layout.

Design starts in the behavioral domain. The function of the chip is defined by an algorithm. At the chip level only the functionality is initially considered. The internal layout is not considered and no attention is paid to the timing. This first step gives the designer a

chance to compare the algorithm against the initial design requirements. If it does not function correctly, the algorithm is changed to meet the requirements.

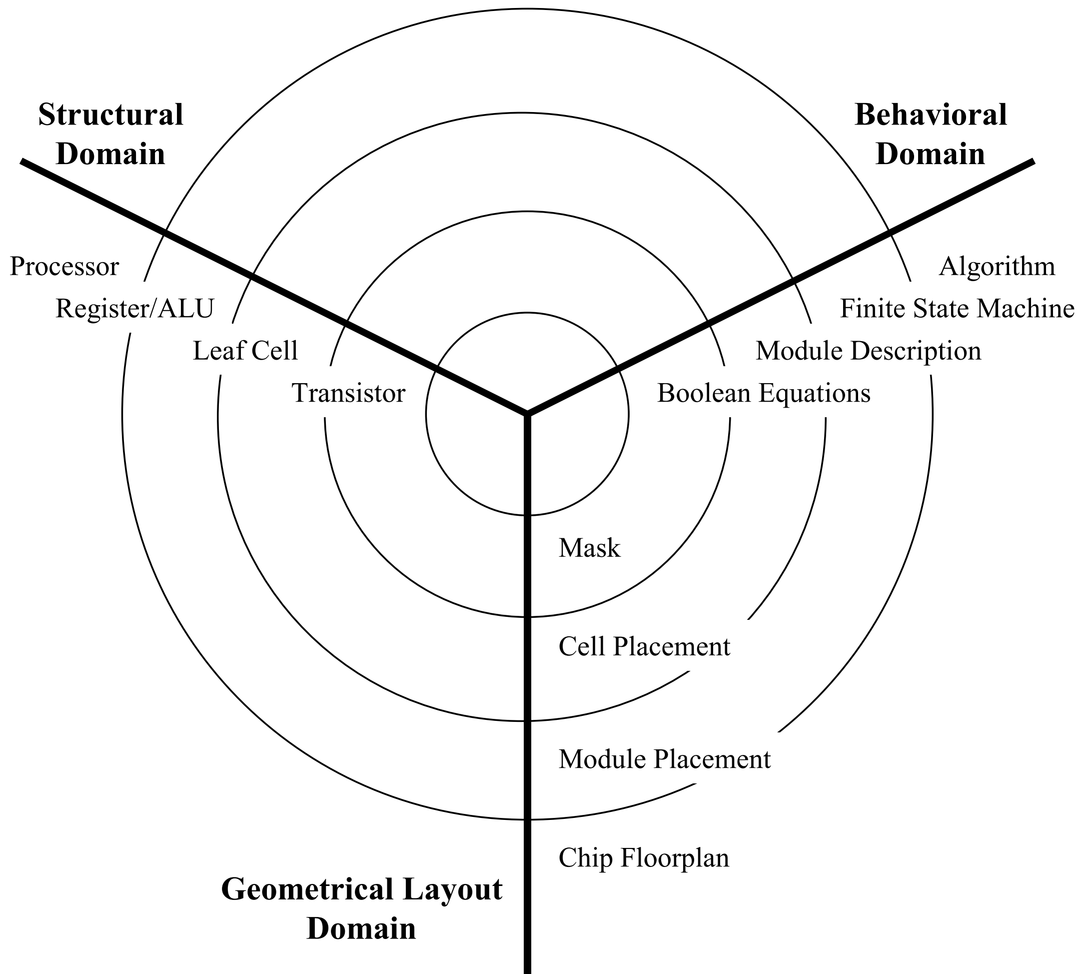


Figure 2-1. VLSI Design Process: Y-Diagram [4]

The structural development starts by defining the external interface of the chip itself. At successively lower levels of hierarchy, the functional modules and their interconnections are defined. Eventually, the structural design gets down to the gate or transistor level. At this level, timing can be analyzed starting with the gates themselves and then moving up the hierarchy to the functional blocks and finally the complete chip.

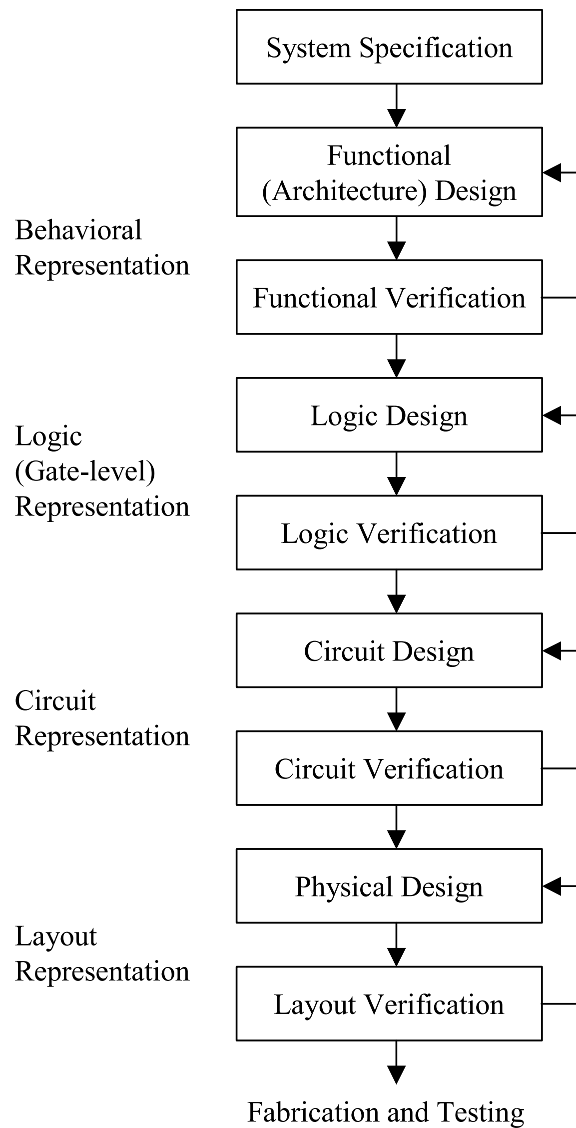


Figure 2-2. VLSI Linear Design Flow [4]

The layout domain links the elements defined in the structural design to the physical layout of the chip. Initially, the floorplan maps the functional elements onto the chip to both reduce chip area and minimize interconnect length. This is completed prior to any layout of the functional elements. Consequently, this requires good estimates for the size and dimensions of each functional block. After floorplanning, each functional block is laid out by

its sub-blocks. The sub-blocks are then laid out at the gate or transistor level. Layout is often done using standard-cell gate libraries and CAD place-and-route tools.

For the Microdot, I used VHDL (**V**ery high-speed integrated circuit **H**ardware **D**escription **L**anguage) to build both the behavioral and structural circuit representations. Hardware description languages such as VHDL are used to create circuit designs in both the behavioral and structural domains. They allow seamless integration between the two design domains. Functional elements can either be written as a set of equations that make up an algorithm or as a collection of structural elements that execute a particular function. VHDL also incorporates timing that allows complete simulation of both behavioral and structural designs. Converting behavioral representation to structural is often accomplished with logic synthesis tools. Synopsys Design Analyzer is one such tool and was the choice for logic synthesis of the Microdot. Design Analyzer converts behavioral VHDL to its structural design using user-specified gate libraries. The synopsys gate libraries are usually designed for specific standard-cell layout libraries. This way, the structural designs created with Design Analyzer port directly to automatic place-and-route tools. This allows the designer to make a quick jump from behavioral VHDL code to physical layout. For the Microdot, I used a radiation-tolerant gate array library designed by Mission Research Corporation (MRC), Microelectronics Division [18]. MRC Microelectronics specializes in the design and analysis of radiation-hardened semiconductor devices. They support the Space Electronics Branch of the Air Force Research Laboratory (AFRL/VSSE), the sponsor of this research, in the design and testing of space-qualified semiconductor devices. MRC has a place-and-route tool known as GARDS for auto-layout of gate array designs.

2.3 Stack Processors

The Microdot was designed as a modified stack-based processor. Unlike most processors that use general-purpose registers to hold data operands, a stack processor holds all operands in the stack memory. Instructions retrieve operands from the top of the stack and return results to the top of the stack. For example, an ADD instruction would pop two elements off the stack, perform the addition and then return the result to the top of the stack.

The stack memory is either a dedicated RAM or part of a combined RAM that includes both instructions and data. The top of the stack (TOS) is defined by a hardware register that contains the address of the last element placed on the stack. In an "increasing" stack, the stack pointer is incremented when new data is "pushed" onto the stack, and decremented when the data is "popped" off the stack. Stacks are also designed as decreasing where the stack pointer is decremented for a push and incremented for a pop. In addition to the stack pointer, there may also be registers that store the stack size and stack limit.

The advantage of a stack machine is that instructions and logic are quite simple because the operands are always coming from the top of the stack. The disadvantage of the stack machine is that it often requires many additional instructions to manage the stack. This can make programming a stack machine more difficult. Another disadvantage of the stack architecture is that it is not compatible with the highly parallel nature of modern high performance processors that require simultaneous access to several operands in a large register set [5].

Most general-purpose processors use a stack defined somewhere in the memory structure to store processor state data as subroutines are executed in the current program. For example, before the processor jumps to a subroutine, it may store the value of the program

counter and registers in the stack memory. When the subroutine has completed execution, returning to the correct spot in the main program is as easy as reloading the data from the stack. The stack allows the use of nested subroutines, which makes programming easier and more versatile. The number of subroutine calls is limited by the size of the stack divided by the size of the memory needed for a subroutine call. Microdot is considered the smallest element of computing in a distributed computing architecture. Therefore, each Microdot can be viewed as a single subroutine dedicated to a single device. This means we can eliminate subroutine capability for keeping the Microdot small and simple [2].

The modified stack architecture expands the flexibility of the stack while retaining a fairly simple design. The top of the stack is stored in a register separate from the stack memory. For all ALU operations, the top of stack register is always used as the first or only operand. Results are then stored back to the top of stack register. Instead of limiting the second operand to the second element on the stack, the modified stack uses an operand index from the instruction to select the second operand relative to the stack pointer. For the four-bit Microdot, the instruction operand index (also called stack increment) is added to the stack pointer to address the second operand. This allows the programmer to select any of the top 16 stack elements as the second operand for a two-operand instruction. The modified stack architecture presented in [2] uses the swap and pick instructions to manage data within the stack. The swap instruction swaps the top of stack register and any of the stack addresses. The pick instruction replaces the top of stack register with any of the stack addresses. The additional hardware elements required for the modified stack are a stack adder and a stack address multiplexer.

2.4 Serial Bus Protocols

The Microdot is not only a microcontroller, but also the lowest computing element in a distributed computing hierarchy. The Microdot is controlled by a supervisor processor, which downloads programs and transfers data between itself and the Microdot. The supervisor-Microdot data interface is key to making the Microdot useful in a distributed computing architecture. We want a simple interface that efficiently transfers data, yet reduces the wiring needed for interconnecting the devices. If each Microdot requires dedicated signal lines between itself and the supervisor processor, the supervisor processor will require an excessive amount of input/output lines. This limitation necessitates that the Microdots share a single data bus controlled by the supervisor processor. A design goal for Microdot has been to keep it small and simple with a low pin count. A serial data bus will both minimize the pin count and ultimately reduce the wiring required for the distributed computing architecture. Data throughput is not a concern and is sacrificed for the simplicity of the serial bus. A number of serial bus protocols have been created for inter-IC data transfer and several were investigated for use in the Microdot serial bus. Rather than create a serial bus unique to only the Microdot, I wanted to use an accepted serial bus standard that would ease the design burden for designers that utilized the Microdot.

The terms master and slave are used to describe the functional role a circuit has on the bus. A master controls the bus and initiates data transfer. A slave can monitor the bus at all times, but does not drive data on the bus unless commanded by the master. There can only be one active bus master although devices may be designed to act as master, slave, or both. A bus protocol allowing multiple masters is referred to as multi-master and has a defined

arbitration procedure for resolving bus control. For the Microdot design, the supervisor processor is the bus master and each Microdot is a slave.

2.4.1 Candidate Serial Bus Protocols. The Serial Peripheral Interface (SPI) and Microwire synchronous serial bus protocols are full duplex and use three signal lines for data transfer [6]. Full duplex means that data can be transferred in both directions simultaneously. The three signals on the SPI and Microwire buses are the clock, data-in, and data-out. The clock signal synchronizes data transfer on the data-in and data-out signals. For the Microwire bus, the low-to-high transition (rising edge) of the clock is the synchronizing edge for slave devices. Input data must be valid around this clock edge and output data becomes valid shortly after the clock transition. SPI uses a similar synchronization scheme, although data transition clock edge can be specified during bus setup. The disqualifying factor for these two bus protocols is that they require chip select lines for each device. A design decision was made not to use a bus protocol that required chip select signals. Instead, it was decided that a bus protocol that sends the slave address over the serial data line would be advantageous to reducing the wiring in the distributed architecture because it eliminates the need for chip select lines to each slave on the bus. The two most viable candidates using this type of serial bus protocol are the Dallas Semiconductor 1-Wire and the Phillips I^2C (Inter-IC Communication) buses.

The 1-Wire bus is the absolute minimum serial bus. As the name indicates, only one signal wire is required to transfer data. Like all the serial bus protocols presented here, there must also be a common ground signal that allows voltage referencing. The drawback to the 1-Wire bus is that it is asynchronous. Each device on the bus requires its own clock for bit timing. The design decision to use an external clock eliminated the possibility of using the

Dallas 1-Wire bus. Creating an on-board clock was beyond the scope of this VLSI project. For future versions of the Microdot, an on-board clock would allow the use of the 1-Wire serial bus protocol. Another potential advantage of 1-Wire bus is that it can also be used to power devices [7]. The use of small capacitors allows devices receive power and data over the same line.

2.4.2 I^2C Serial Bus Protocol. The only serial bus that met the required design goals was the I^2C (Inter IC Communication) serial bus protocol. The I^2C serial bus was originally designed by the Philips Semiconductor Corporation to provide an Inter-IC communication standard for the typical devices that were integrated into most designs like microcontrollers, RAMs, EEPROMS [8]. The utility of I^2C quickly made it a de facto world standard and it is now utilized by over 50 semiconductor companies [8]. Also, I^2C was used as the data bus for the Emerald research satellite; a small satellite developed at Stanford to study distributed computing in space systems [9]. Emerald was part of the University Nanosatellite Program funded by the Air Force Office of Scientific Research (AFOSR) and the Defense Advanced Research Project Agency (DARPA) [9]. Much of the reasoning for using I^2C on Emerald fit with my own reasoning about using I^2C for the Microdot. Specifically, the simplicity of I^2C and its ability to scale well to arbitrarily sized networks made it the best candidate [9]. Standard I^2C specifies data transfer speeds up to 100 kilobits per second (Kbps) although a newer high-speed mode specifies the data rate up to 3.4 Mbps. Figure 2-3 illustrates the I^2C bus concept.

The two signal lines that comprise the I^2C bus are the serial data (SDA) and serial clock (SCL). Both signal lines are connected to the positive supply voltage (Vdd) through a

pull-up resistor. Output connections to the bus signals must be open-drain. This gives the devices on the bus a wired-AND configuration and prevents any damage from multiple

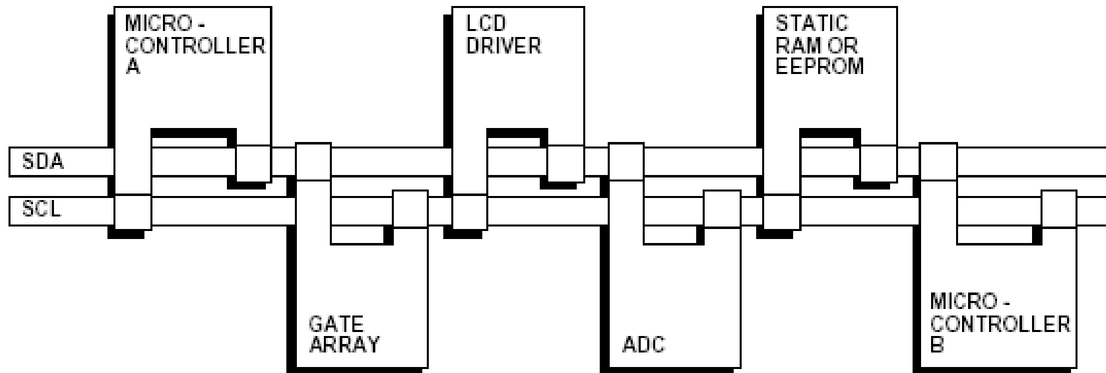


Figure 2-3. I^2C Serial Bus Concept. Devices interconnect with two common signal lines: Serial Data (SDA) and Serial Clock (SCL) [8]

devices driving the bus lines at the same time. SDA carries the bit data while SCL synchronizes the transfer. This synchronized bus offers the design advantage that no internal clocking is required. I^2C is also a multi-master bus. Although the original concept of the Microdot distributed computing system is to have a single supervisor processor as the bus master, a single I^2C bus could have multiple supervisor processors. This gives a designer potential redundancy, which is important in designs that must be highly reliable.

Data transfer on the I^2C bus is completed in eight-bit (byte) blocks with the most significant bit (MSB) first. The I^2C bus is designed so that each device connected to the bus has a unique address. This seven-bit address is part of the first byte transferred, which is called the address byte. The seven-bit address space allows 128 possible device addresses. During data transfer, the SDA signal is only allowed to change when SCL is low. Figure 2-4 illustrates the bus rules for standard data transfer.

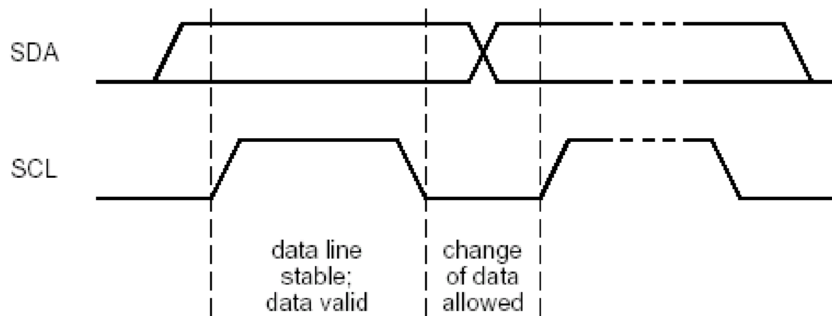


Figure 2-4. I^2C Bit Transfer. Bit values on SDA can only change when SCL = ‘0’ [8]

When the master is ready to execute a data transfer it asserts the start condition on the bus lines. The start and stop conditions are non-standard bus conditions that signal the start or stop of data transfer. Figure 2-5 shows the start and stop conditions. The start condition is indicated by a high-to-low transition of the SDA signal when SCL is high.

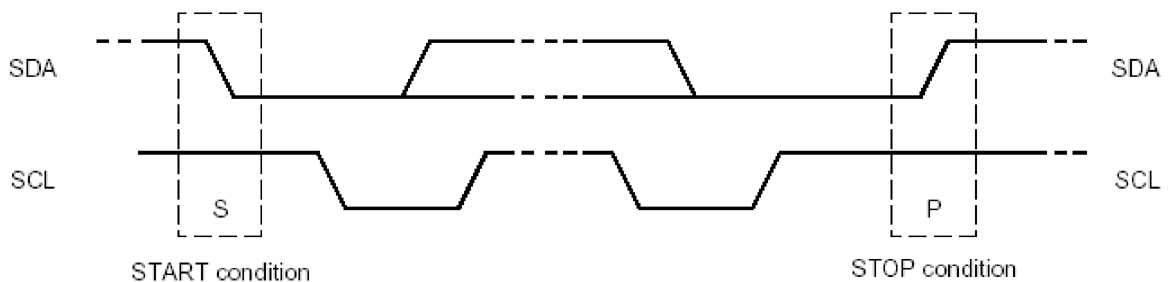


Figure 2-5. I^2C START and STOP Conditions [8]

After the start condition is asserted, all devices on the I^2C bus will begin shifting in the first byte. Data is valid when SCL is high. The first byte contains the seven-bit address plus the data direction bit. A data direction bit of ‘0’ indicates a master-to-slave transfer, while a ‘1’

indicates a slave-to-master transfer. If a slave detects its address in the address byte, it acknowledges its presence to the master by pulling the SDA line low during the acknowledge clock cycle. The acknowledge clock cycle comes after the eighth bit is sent when the master releases the SDA line (SDA = ‘1’) for one SCL cycle. The master continues the data transfer if the address is acknowledged or terminates it if no acknowledge is received. For a master-to-slave transfer, the master continues the serial transmission one byte at a time checking for acknowledge after each byte. After any byte during the transmission, the slave can stop the data transfer by not asserting the byte acknowledge. The master signals the end of a transmission by either asserting the stop condition or another start condition. The stop condition is the low-to-high transition of the SDA line when SCL is high.

For a slave-to-master transfer, the master releases the SDA line after the address byte and becomes the data receiver. The slave drives its serial data on the SDA line synchronized to SCL, which is still driven by the master. After each byte is received, the master now acknowledges receipt by pulling SDA low during the acknowledge clock cycle. Like the master-to-slave transmission, the master can terminate the transfer after any byte by not asserting the byte acknowledge and then asserting the stop condition. Figure 2-6 demonstrates a complete data transfer on the I^2C bus.

The I^2C bus specification is more complicated than what is presented here, but the basics presented in this section are sufficient for designing the Microdot serial communication interface.

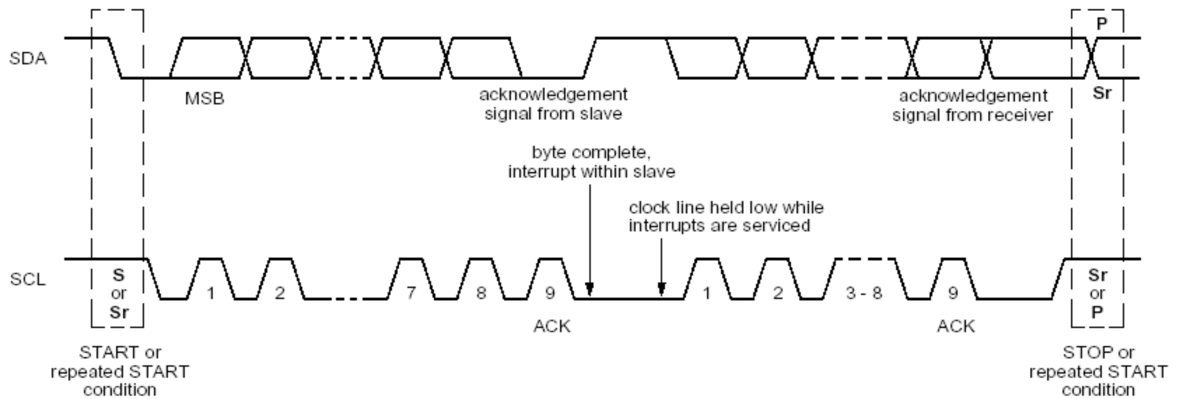


Figure 2-6. I^2C Data Transfer [8]

2.5 Four-bit Microcontrollers

The market for four-bit microcontrollers is quite diverse, with at least 20 companies producing them. Table 2-1 is a representative selection of the four-bit microcontrollers currently on the market. The performance is not charted, although performance, operating current, and clock frequency have a linear relationship (high performance = high current). Most four-bit microcontrollers are designed for small, battery-powered devices. Operating frequencies are slow to minimize power consumption, with 32 kHz being a popular choice. As can be seen, the operating current is quite low with the EM Microelectronic (EM6603) and Epson Electronics (SIC60N01 and E0C63158) devices operating in the single-microamp range. For comparison, a single AAA battery (1150 milliamp-hour) could power an EM6603 for 638,000 hours (72.9 years). Obviously, the battery would go bad long before the current was drained. Most four-bit microcontrollers are also designed with a standby or sleep mode that minimizes current consumption. For all devices available, the instruction memory is either a masked ROM, one time programmable (OTP) ROM, or EEPROM. Instruction

memories range from 1K x 12 up to 24K x 8. Data memories range from 52 nibbles up to 256 nibbles. Most of these microcontrollers also have peripheral functions like Analog-to-Digital converters, LCD drivers, timers, and event counters that offer designers a one-chip solution for small battery-powered devices.

Table 2-1. Representative Sample of the Four-bit Microcontroller Market

Manufacturer	Part Number	Instruction Memory	Data Memory	I/O	Operating Frequency	Supply Voltage	Operating Current (Typical)	Standby Current (Typical)	Chip Size (mm)
ELAN Microelectronics Corporation	EM73PE00	24K x 8	128 x 4	8 input 8 output 5 bidirectional	200 KHz – 4 MHz	2.5 - 5.0	0.4 mA @ 3.0 V	0.4 uA @ 3.0 V	1.52 x 3.55
ELAN Microelectronics Corporation	EM73201	2K x 8	52 x 4	4 input 4 output 8 bidirectional	32 KHz – 4 MHz	2.4 - 6.0	0.7 mA @ 5.0 V	60 uA @ 5.0 V	1.82 x 1.51
Epson Electronics	SIC60N01	1K x 12	80 x 4	4 input 2 output 4 bidirectional	32 KHz	1.8 - 3.6	2.5 uA @ 3.0 V	1.0 uA @ 3.0 V	2.64 x 2.18
Epson Electronics	E0C63158	8K x 13	512 x 4	9 input 12 output 20 bidirectional	32 KHz – 4 MHz	0.9 - 3.6	4 uA @ 3.0V 32 KHz	2 uA @ 1.5V 32 KHz	N/A
Hitachi Semiconductor	HD404054	4K x 8	128 x 4	8 input 27 bidirectional	4 MHz	1.8 - 6.0	0.6 mA @ 3.0V 800KHz	0.2 mA @ 3.0V 800 KHz	N/A
Mitsubishi Semiconductors	M34280M1-XXXFP	1K x 9	32 x 4	1 input 7 output 7 bidirectional	4 MHz	1.8 - 3.6	0.35 mA @ 3.0V 500 KHz	1 uA @ 3.0V 500 KHz	N/A
Mitsubishi Semiconductors	M34570M4-XXXFP	4K x 10	128 x4	6 input 10 output 12 bidirectional	4.2 MHz	2.0 - 5.5	0.4 mA @ 3.0 V 500 KHz	0.1 uA @ 3.0V 500 KHz	N/A
Toshiba Electronics	TMP47C422N/F	4K x 8	256 x 4	20 bidirectional	30 kHz – 8 MHz	2.2 - 5.5	0.5 mA @ 3.0V 400KHz	0.5 uA @ 5.5V	N/A
EM Microelectronic	EM6603	2K x 16	96 x 4	4 input 12 bidirectional	32 kHz	1.2 - 3.6	1.8 uA @ 1.5V	0.1 uA @ 1.5V	3.048 x 2.59
Oki Semiconductor	MSM64152A	1.5K x 8	128 x 4	4 input 4 output 4 bidirectional	32 kHz	1.15 - 3.5	6 uA @ 1.15V	0.9 uA @ 3.0 V	3.90 x 3.48

No four-bit microcontrollers were identified with reprogrammable instruction memory. Most outmatch the Microdot in processing capability, but none offers its unique

capabilities. The Microdot concept is to create a device for low-end distributed and re-configurable computing. Thus, while many of the four-bit microcontrollers available have adequate performance and could likely be integrated into a distributed system, they would not work for the reprogrammable satellite design. Also, none of the four-bit microcontrollers identified were radiation-hardened.

2.6 Radiation Hardening of Electronics

Radiation can degrade the performance of semiconductor devices and may cause them to fail. Electronics designers must take special steps to improve the reliability of electronics in a radiation environment. This process is referred to as "radiation hardening". To understand the process of radiation hardening electronics, it is important to understand radiation environments and the effects of radiation on semiconductor devices.

There are two radiation environments of concern. The first is the space radiation environment that consists primarily of high-energy electrons, protons, alpha particles, and cosmic rays [10]. The second is the nuclear weapon environment that consists of a high dose rate pulse of X-rays, gammas, neutrons, and other reaction constituents [10]. For this research, I was concerned with the space radiation environment and the effects of space radiation on Complementary Metal-Oxide Semiconductor (CMOS) devices. Military and space application designers prefer CMOS technology because of its high noise margins and low static power requirements [10]. Scaling and integration are another advantage CMOS technology has over other semiconductor technologies. The drawback is that CMOS is susceptible to two types of space radiation effects. These are total ionizing dose (TID) and single event effects (SEE). TID effects are the result of accumulated exposure to ionizing

radiation. SEE are the result of a single high-energy particle that strikes the device. These radiation effects are detailed in the sections that follow.

2.6.1 Space Radiation Environment. The two sources of radiation in the near-Earth space environment are trapped particles and transient radiation. The earth's magnetic field traps energetic protons, electrons, and heavy ions in the Van Allen radiation belts as shown in Figure 2-7.

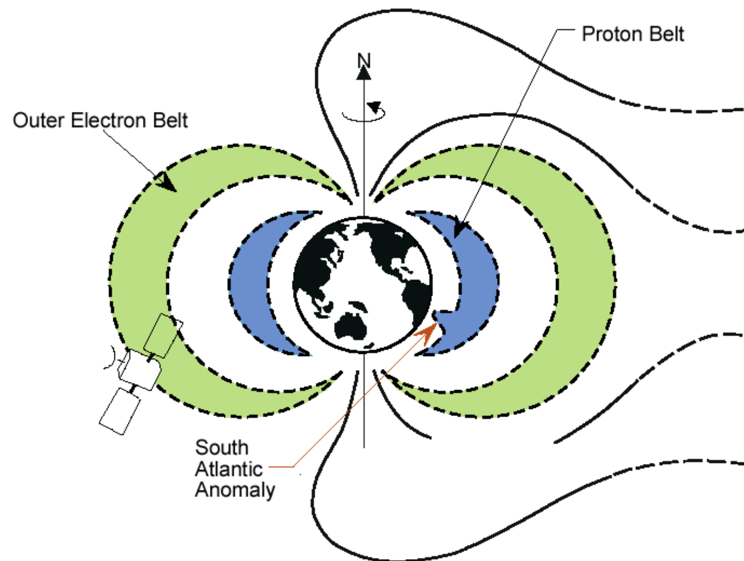


Figure 2-7. Van Allen Radiation Belts [11]

The inner belt consists primarily of high-energy protons and extends from 1.1-3.3 earth radii [12]. The South Atlantic Anomaly is a particular area of interest in the inner belt. Irregularities with earth's magnetic field extend the proton belt to altitudes as low as 250 km centered in an area east of Brazil [12]. Satellites in low earth orbit that pass through this region are subject to a high flux of energetic protons that increase radiation exposure and cause single event effects. The outer radiation belt consists mainly of energetic electrons and

extends from 3-9 earth radii. The outer belt converges with earth's atmosphere at high latitudes where the magnetic field lines converge at the magnetic poles. The intensity of the belts is affected by solar activity. Shielding is quite effective at stopping electrons because of their low energy and low mass. However, it is not as effective against high energy protons and heavy ions. As you can see, the satellite orbital altitude and inclination will affect the amount of radiation exposure.

2.6.2 Total Ionizing Dose. When ionizing radiation penetrates a semiconductor device, it generates electron-hole pairs along its traversed path. In the silicon dioxide (SiO_2) that makes up both the gate and field oxides, holes that do not quickly recombine can become trapped and cause a buildup of positive charge. This occurs because holes have a very low mobility in SiO_2, while electrons have higher mobility and are quickly swept away by the electric field in the oxide. The holes then slowly migrate in the direction of the electric field (positive to negative). This process is illustrated in Figure 2-8.

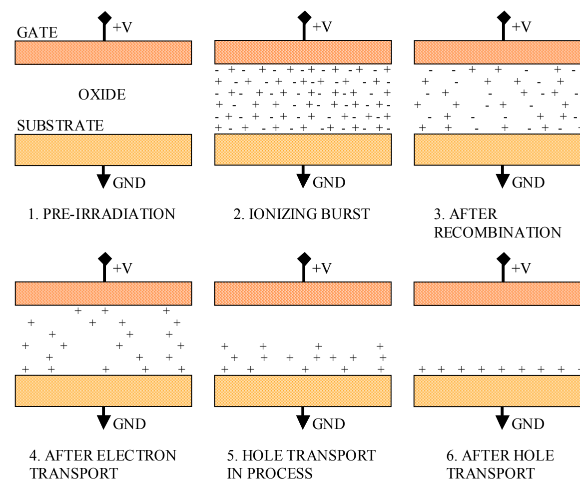


Figure 2-8. Transistor Ionization Process [10]

In the case of a positive-biased gate voltage, the holes move away from the gate toward the SiO_2-Si interface [10]. In addition to creating oxide-trapped holes, ionizing radiation also

generates interface traps at the SiO_2-Si boundary. For n-channel transistors, interface traps are negatively charged and cause positive threshold voltage shift. For p-channel transistors, interface traps are positively charged and contribute to negative threshold voltage shift. Additionally, interface traps can also cause degradation of channel carrier mobility and channel conductance. Ionizing radiation also ruptures the chemical bonds in the SiO_2 leading to more defects in the material, which act as carrier traps and store positive charge [10]. For an n-channel transistor (nfet), the buildup of positive charge in the gate oxide causes a negative shift in the threshold voltage (turn-on voltage).

Figure 2-9 is an I-V plot for a MOS transistor before and after irradiation. The oxide-trapped holes cause the negative shift while the interface traps cause the reduction in slope. As the total dose increases, the voltage shift from oxide-trapped holes may eventually cause the transistor to conduct even with no applied gate voltage. This is referred to as depletion mode and leads to device failure. Conduction of the nfet at zero gate voltage also causes an increase in power consumption in CMOS because with both nfets and pfets (p-channel transistors) conducting, there is a direct path from supply voltage to ground limited only by the resistance of the transistor channels. As total dose increases above 100 krad(Si), the effect of negatively-charged interface traps dominates over oxide-trapped charge. This condition is called “rebound” and causes the nfet threshold voltage shift back positive eventually leading to a higher threshold voltage than before irradiation. This causes the nfet to be harder to turn-on. Figure 2-9 also illustrates the effects of rebound. Increased threshold voltage and

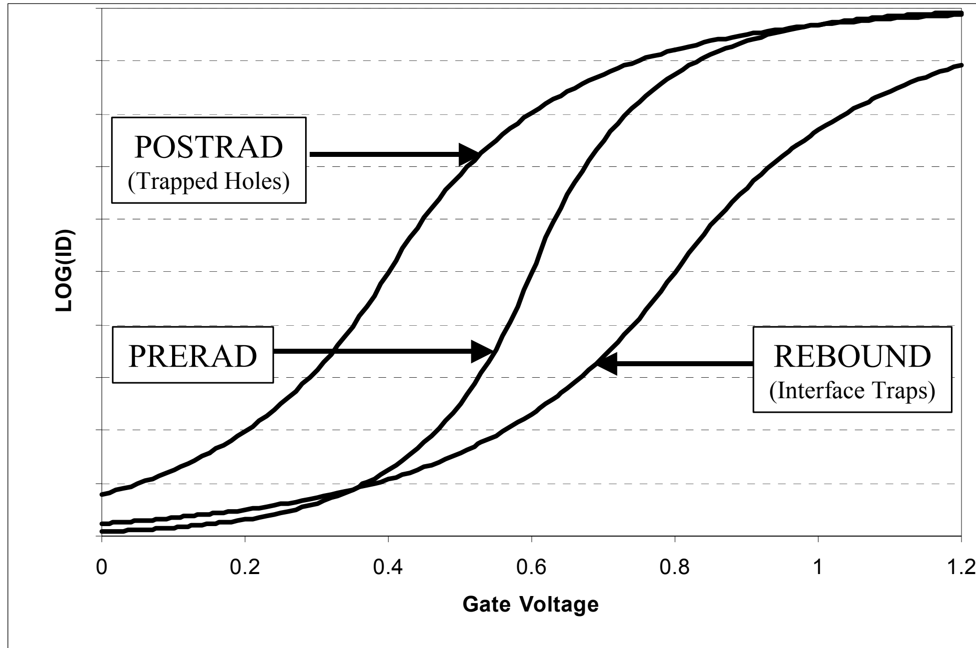


Figure 2-9. Normalized N-Channel MOSFET I-V Curve. Illustrates the effects of ionizing radiation on threshold voltage and transconductance. At lower total ionizing doses, trapped holes in the gate oxide (SiO^2) cause negative shift in the threshold voltage. As total ionizing dose increases above 100 krad(Si), negatively charged interface traps cause positive shift in the threshold voltage and a reduction in transconductance.

decreased transconductance will reduce the switching speed of CMOS circuits and may eventually lead to device failure.

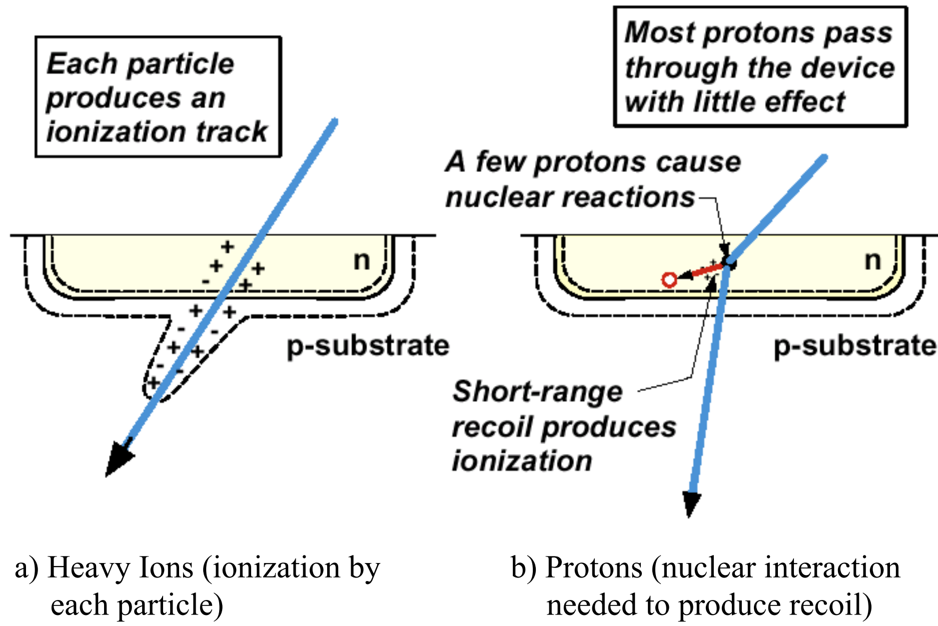
P-channel transistors are more resistant to total ionizing dose. First, since the gate voltage is negative, the holes collect near the gate and have less effect on the transistor channel than in nfets. Also, the threshold voltage shift from both oxide-trapped charge and positively-charged interface traps is negative, so the P-channel transistor becomes more difficult to turn on which leads to slower response time. CMOS transistors must be designed to increase drive and reduce channel resistance to counter the negative threshold voltage shifts in pfets and positive threshold voltage shifts in nfets.

Total dose radiation exposure is measured in rads. The term rad (radiation absorbed dose) quantifies the total radiation exposure of a material [13]. One rad (Si) is equal to 100 ergs of energy absorbed per gram of the material, which, in this case, is silicon. The total dose radiation threshold of a device is the minimum level of rad(Si) that will cause device failure [10]. Total ionizing dose exposure at low earth orbit can be quite low with exposures of less than 10 krads(Si) over a 20 year mission [14]. Geosynchronous satellites can receive up to 100 krads(Si) after 10 years on orbit. The most severe total dose orbits are one-half geosynchronous, which can reach 1Mrad(Si) after 8 years [14].

2.6.3 Single Event Effects. The space radiation environment exposes circuits to a varying flux of ionizing particles. At geosynchronous orbits, the cosmic ray heavy ion flux is approximately 100 particles/cm^2 per day [10]. Shielding does little to stop this flux of high-energy particles that can singly cause errors and/or damage when they strike the circuit. The effects of a single particle strike are known as Single Event Effects (SEE). If space electronics are to be reliable, they must be designed to reduce susceptibility to SEE. Single event upset (SEU) is a change in a stored bit value caused by a single particle strike. Single event latchup (SEL), a hard error, occurs when a particle strike induces latchup, which may lead to device failure. These effects are examined further in the next two sections.

2.6.3.1 Single Event Upset. Single event upset (SEU) describes the event when a stored bit in a circuit changes state due to the charge deposited on a sensitive node by a single high-energy particle. SEU is considered a soft error because the circuit is not damaged. Particles causing SEU can be photons, protons, alpha and beta particles, and heavy ions [10]. The sources of this radiation are the Van Allen radiation belts, solar events (sun) and galactic cosmic rays that originate outside the solar system. Figure 2-10 illustrates the effects of a

single particle strike. An ionizing particle that penetrates the semiconductor creates a narrow ionization track (radius < 1um) as it traverses the material [15].



a) Heavy Ions (ionization by each particle)

b) Protons (nuclear interaction needed to produce recoil)

Figure 2-10. Single Particle Strike [11]

The sensitive circuit node then collects these mobile charge carriers. SEU occurs when the charge collected on the sensitive node causes a charge imbalance and changes the value of the stored bit. The number of electron-hole pairs created is a function of the particle energy and the stopping power of the semiconductor in relation to the particle. The stopping power (energy deposition per unit depth – MeV/µm) is related to the linear energy transfer (LET), which is measured in units of MeV/(cm2/gm). For silicon, one electron-hole pair is created for every 3.6 eV of deposited energy [16]. SEU susceptibility is characterized by the LET threshold at which bit upsets begin to occur. Figure 2-11 illustrates the sensitive nodes in a typical six-transistor SRAM cell storing a ‘0’. A particle strike on the drain of pfet 1

produces charge that must be dissipated through nfet 3. If the charge is sufficient to raise the gate voltage of the feedback inverter (pfet 2 – nfet 4) past the nfet threshold voltage, it will switch the inverter which then switches the state of the memory bit.

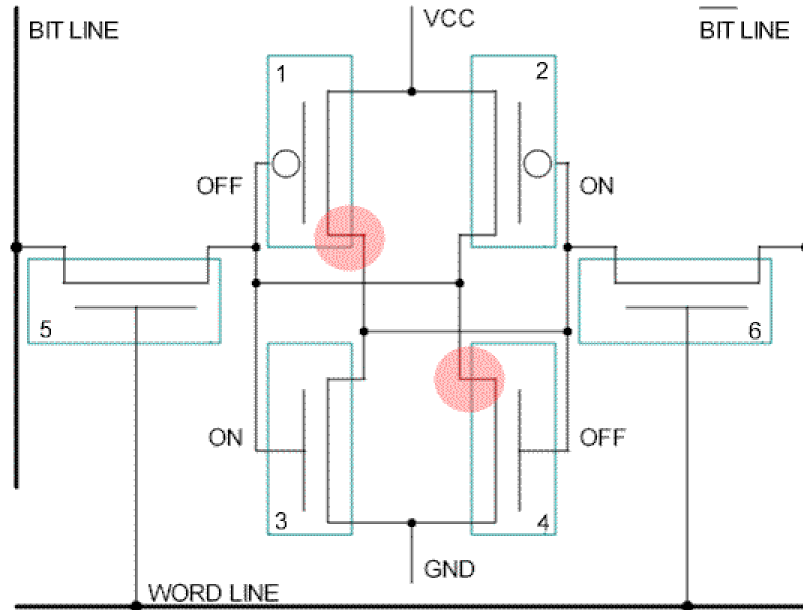


Figure 2-11. Six-transistor SRAM Cell. Shaded circles mark nodes sensitive to single particle strike [12]

Circuits can be hardened against SEU in a number of different ways. Process methods include using thin epitaxial layers on heavily doped substrates or silicon-on-insulator (SOI) for fabrication. These methods minimize SEU by reducing the volume of charge that can be collected at sensitive nodes. One popular design method is the use of polysilicon resistors in the feedback paths of memory circuits. Polysilicon resistors increase the RC time constant of the feedback paths creating a low-pass filter that rejects the transient pulses cause by single particle strikes. The drawback to this method is that it increases the memory response time,

thus reducing performance. System level methods of reducing SEU involve using redundant memory cells with error detection and correction. Triple modular redundancy (TMR) is one such method that uses three memory cells and a 2-of-3 voting circuit for each memory cell [17]. Additional circuitry detects bit errors and corrects them. SEU still occurs with TMR when a particle strike upsets two bits simultaneously. Therefore, it is important to put enough space between the redundant memory cells to reduce the possibility of multi-bit upset. The obvious drawback to redundancy is the area penalty, but it is the price that must be paid for reliability.

2.6.3.2 Single Event Latchup. The layout of conventional CMOS transistors forms a parasitic silicon controlled rectifier (SCR) from a four-layer PNPN structure that can be turned on by a heavy ion strike [18]. The parasitic SCR turn-on condition is called single event latchup (SEL) and is a self-reinforcing condition that can be destructive to the circuit. SEL is a considered a hard error because it damages the circuit. The PNPN structure that creates the parasitic SCR is shown in Figure 2-12.

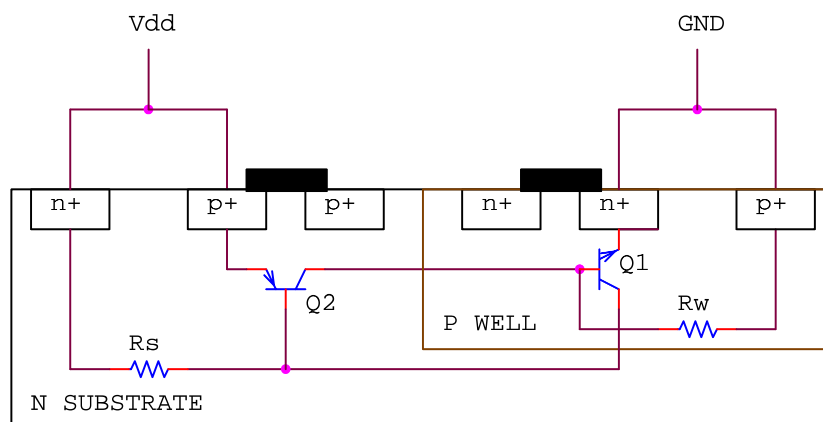


Figure 2-12. CMOS PNPN Structure [10]

An ion strike deposits charge in either the n-substrate or p-well. As the charge from the ion strike flows to the substrate or well contacts, there may be sufficient voltage drop across the intrinsic resistance (Rs or Rw) to forward bias the base-emitter junction of the transistor (Q2 or Q1) and turn it on. If the gain product of the two parasitic bipolar junction transistors is greater than one, latchup occurs when the current from one transistor causes a voltage drop sufficient for turning the other transistor on. The currents from both transistors then begin to reinforce themselves causing ever-larger current flow from power to ground, which eventually leads to device burnout if the current cannot be limited or stopped. Without current limits, the only way to stop the latchup condition is remove the power and allow the device to reset. The key to eliminating latchup is designing the transistors so the gain product is less than one. Design practices used to reduce latchup susceptibility are [18]:

1. Increasing the spacing between the well edge and n+ and p+ source/drain regions.
2. Using n+/n well and p+/p well guardbands to reduce parasitic transistor gain.
3. Increasing the number of well/substrate contacts and placing them closer to the latch path.

The drawback to these design techniques is that they incur a significant area penalty, which is illustrated in Figure 2-16.

2.6.4 Radiation Hardening By Design. Shielding reduces radiation exposure by stopping or slowing particles, but does not eliminate it. A realistic design cannot simply shield its way to radiation hardness. Consequently, circuits must be fabricated and/or designed to reduce their susceptibility to radiation effects.

Radiation Hardening by Design (RHBD) is the process of using layout design techniques to reduce radiation effects on transistor circuits. The quality of commercial

CMOS processes and the use of RHBD techniques facilitates the design of radiation-tolerant VLSI circuits that are less expensive than circuits created with traditional rad-hard processes. Also, the demand for electronics capacity in commercial markets has greatly decreased the manufacturer interest in developing rad-hard components [12]. This both keeps the cost high for rad-hard components and leaves their performance at least one generation behind the leading commercial processes. RHBD offers a low-cost approach to designing circuits for radiation environments. The RHBD design techniques are discussed below.

One total ionizing dose effect on conventional n-channel transistor design is referred to as parasitic edge leakage. This effect is illustrated in Figure 2-13. The cross section view of a typical n-channel transistor shows positive charge collected in the bird's beak regions.

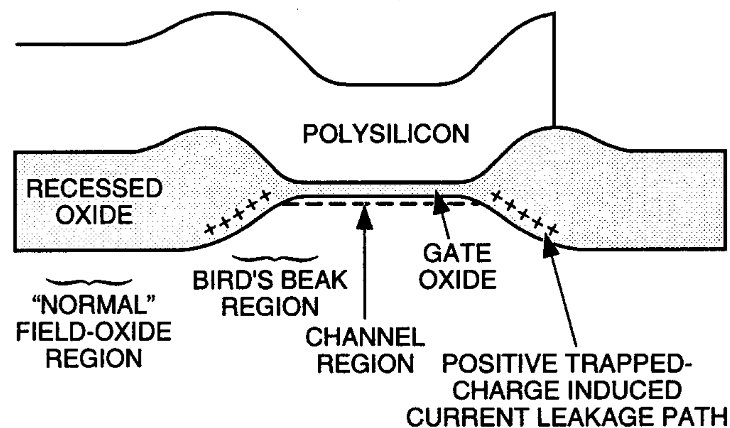


Figure 2-13. Bird's Beak Diagram. Positive charge buildup in the bird's beak region causes parasitic edge leakage [18]

The bird's beak is the transition from gate oxide to field oxide and the edges of the transistor. The trapped positive charge in the bird's beak creates conduction channels at the edge of the transistor that leak current even when the transistor is off. As total dose increases, these parasitic transistors increase their conduction, which increases power supply current and may

lead to functional failure of the device [18]. The annular or re-entrant design of n-channel transistors, shown in Figure 2-14, completely eliminates the oxide transition that creates the bird's beak region between the source and drain. The drawback of this design is that re-entrant designs incur an area penalty over conventional transistor designs.

Another total ionizing dose problem with CMOS processes is referred to as field oxide leakage. During ionizing radiation exposure, holes accumulate in the field oxide that separates the transistors. This can lead to two possible leakage paths. The first leakage path occurs between n-wells connected to Vdd and an n+ source connected to ground. The second is between n+ source and drain regions of adjacent transistors. Eliminating field oxide leakage is accomplished with a channel stop. A channel stop is simply a more heavily doped P region implanted between potential leakage paths. The increased doping makes the channel stop much more difficult to invert and create a channel. Consequently, the leakage path is broken by the channel stop. Channel stops surround the n+ source/drains of a transistor in what is referred to as a guard ring. The guard ring minimizes channel leakage between adjacent transistor n+ source/drain regions.

In addition to providing a channel stop for field oxide leakage, the grounded p+ guardband suppresses latchup by holding the base of the parasitic npn bipolar junction transistor (BJT) at ground. Since both the base and emitter are held at ground, the base cannot get above the threshold voltage to turn the transistor on and begin the latchup cycle. Likewise, the n+ guardband at Vdd holds the base of the parasitic PNP BJT at Vdd, which prevents it from ever conducting.

2.6.5 Radiation-Hardened By Design Gate Array Cell Library. Mission Research Corporation, Microelectronics Division has used radiation-hardness by design (RBHD)

techniques to develop a gate array cell library for the Taiwan Semiconductor (TSMC) 0.35 micrometer CMOS process.

The gate array design is used for its uniformity and design simplicity. It facilitates the design of logic gates with a predictable level of radiation tolerance for all the library cells. Also, library cells can be quickly designed as new cells are defined only by the interconnects and metallization between the gate array cells. Figure 2-14 shows the primitive gate array cell incorporating the RHBD design strategies.

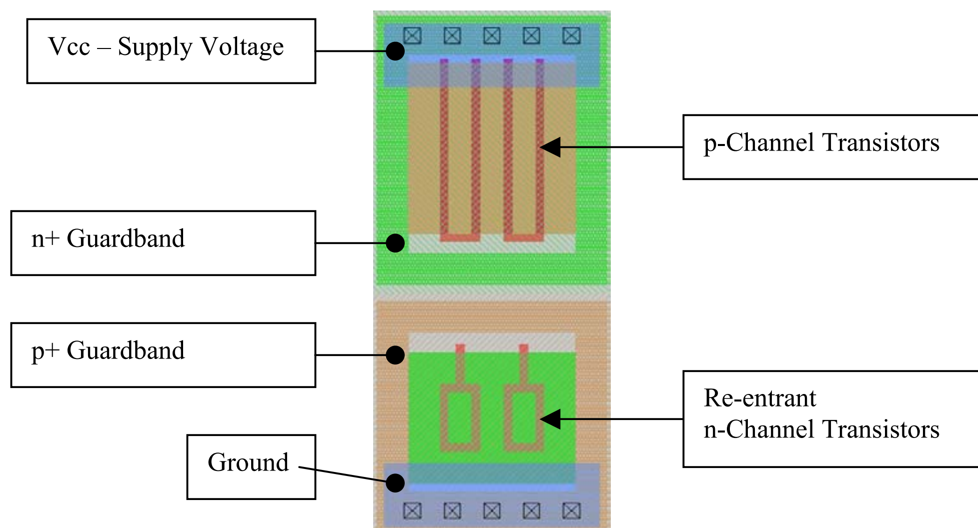


Figure 2-14. Radiation Hardened By Design Gate Array Cell [18]

The re-entrant n-channel transistors eliminate the parasitic leakage channels that form in the bird's beak region. The p-channel transistors use a two-fingered layout and are sized larger to maintain drive strength as ionizing radiation negatively shifts the threshold voltage and reduces drive capability. The p+ guardband in the p well minimizes field oxide leakage and reduces latchup susceptibility. The n+ guardband in the n well and frequent well contacts further reduces latchup susceptibility as discussed above.

Most of the gates use multiple gate array cells as shown in the AOI222 gate of Figure 2-15. Defining a gate is as simple as positioning the metal and interconnects. Most gate

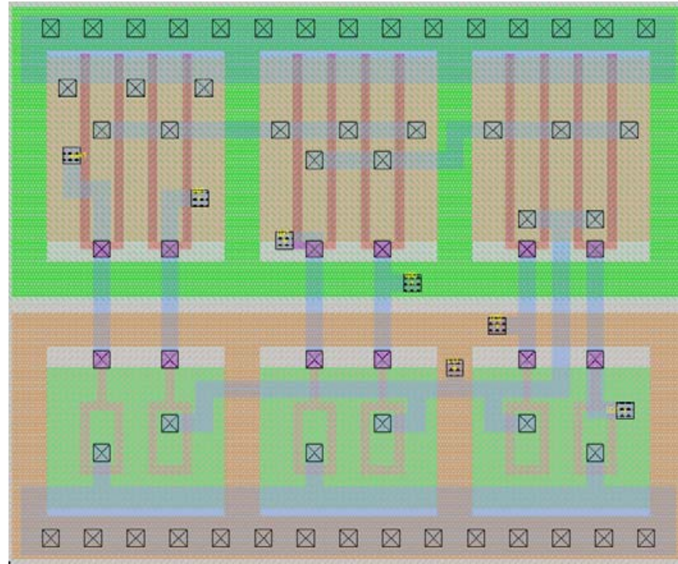


Figure 2-15. AOI222 (AND-OR-INVERT) gate in RHBD Gate Array

arrays are pre-fabricated without metallization. Then, the only fabrication step left is the metallization, which often can be done very quickly. The RHBD gate array is actually just a standard-cell library designed with gate-array cells. The benefits of this gate array design are the quick and simple design of library cells and the uniform level of radiation hardness throughout the design. The drawback is the area penalty incurred from using the same transistor structure for each gate in the library.

An example of the area penalty of radiation hardening is Figure 2-16 which shows a comparison between a RHBD NAND gate using gate-array cells and a non-rad-hard minimum-sized NAND gate in the TSMC 0.35 micron process. The gate array NAND gate is 5.15 times larger than the minimum-sized NAND gate. However, depending on the design,

the overall area penalty would likely be 2-3 times greater than a standard cell, non-rad-hard design. This is a considerable penalty, but it is the design tradeoff that must be made to achieve radiation-tolerant VLSI circuits that will operate reliably in the space environment.

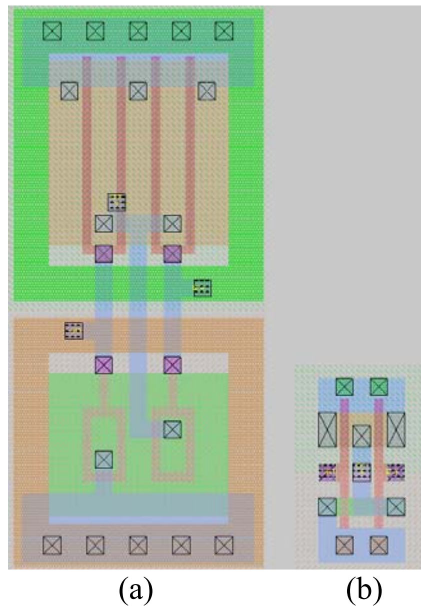


Figure 2-16. NAND Gate Size Comparison. RHBD gate array (a) versus minimum-size transistors (b)

In addition to standard logic gates, the RHBD gate array library also includes latches and flip-flops designed to be SEU hard. The latches are based on the DICE design that uses double redundancy and eliminates direct feedback. This prevents a strike on a single node from upsetting the memory bit [19].

Radiation testing of the gate array design [18] in an older 0.8 micron process showed an ionizing dose hardness of greater than 300 krad(Si) and a SEU LET threshold greater than 50 MeV-cm2/mg. Latchup was not observed in the test facility limit of 102 MeV-cm2/mg. The thinner gate oxides in modern advanced CMOS processes will likely increase total dose

hardness of the RHBD gate array. However, the tradeoff to smaller feature size will reduce SEU hardness, requiring additional design techniques to reduce SEU susceptibility. A similar RHBD test circuit fabricated in the TSMC 0.25 micron process showed total ionizing dose hardness up to 500 krad(Si) [20].

2.7 Summary

The goal of my thesis is to create a small but useful microcontroller for distributed microdevice interfacing in satellites. In this chapter I have covered the VLSI design process, stack processors, four-bit microntrollers, and serial bus protocols that can be used for distributed computing. Finally, I presented an overview of the effects of radiation on CMOS circuits and the design techniques used to harden them for operation in the space environment.

3. Design Overview

3.1 Introduction

In this chapter, I present an overview of the Microdot design. The purpose is to introduce the Microdot architecture and explain how it works. First, I cover the design goals and summarize Microdot architecture. Next, I examine the instruction set and explain the purpose and operation for each functional block.

3.2 Design Goals

The Microdot is conceived as a small, low-power microcontroller for distributed interfacing and control of microsensors and actuators on satellites. Figure 3-1 shows the Microdot distributed computing concept. Arrays of Microdots assigned to individual microdevices are connected to the supervisor processor with an I^2C serial bus. This is designed to distribute computing among the array of microcontrollers instead of using a single microcontroller for all the processing. The first design goal for the Microdot is that it must be capable of operating in the space radiation environment for an extended amount of time. This requires the use of logic gates designed specifically to minimize the radiation effects on CMOS circuits as presented in Chapter 2. Additionally, the Microdot must be designed to consume minimum power, as satellites must operate on limited power budgets either from on-board batteries or solar panels or both. The final major design goal is small area. I envision that Microdots could actually be fabricated on the same substrate adjacent to the microdevices with which they interface. Plus, a satellite with possibly hundreds of microsensors might require nearly an equal number of Microdots, one dedicated to each device. Consequently,

minimizing the size of the Microdot results in a decrease in the potential size and weight of the satellite, which is also important in minimizing launch costs.

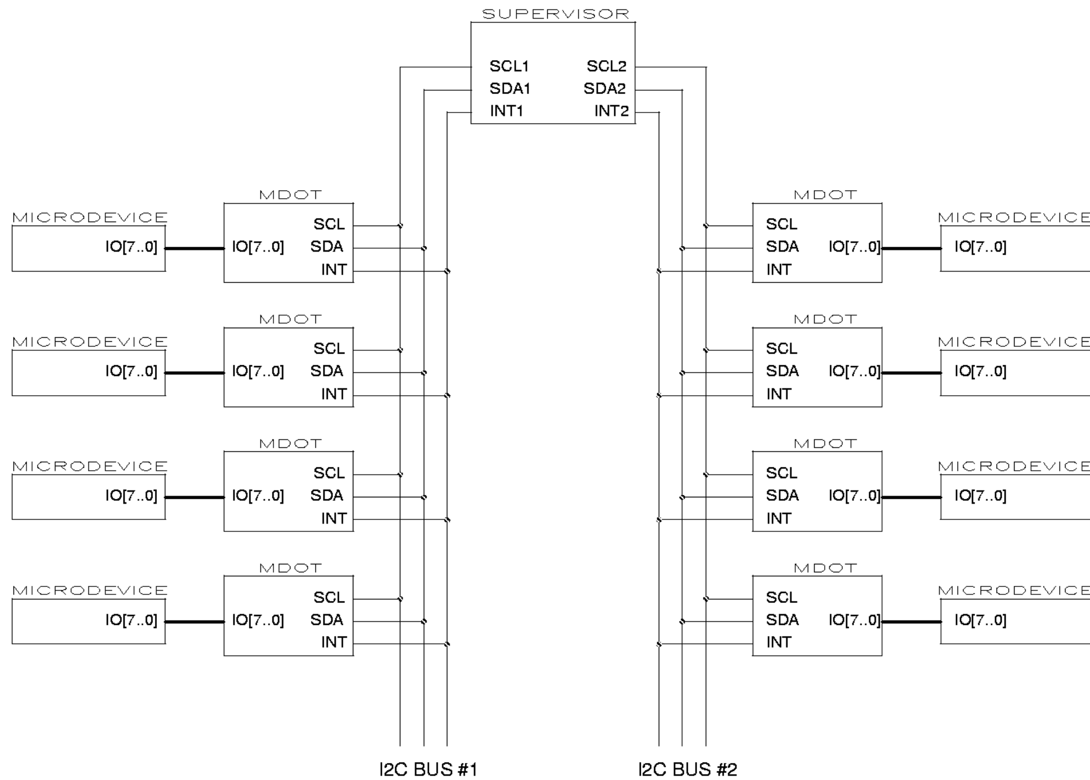


Figure 3-1. Microdot Distributed Computing Concept. Microdots are connected to the supervisor processor with the I^2C bus. Each I^2C bus can connect up to 128 Microdots. Processing tasks for each microdevice are delegated to its assigned Microdot.

3.3 Microdot Design

The Microdot is a four-bit microcontroller with twenty-nine instructions and eight input/output lines. Instructions vary in length from one to four nibbles (four-bit words). It has a program memory size of 1024 nibbles and a data memory size of 128 nibbles. The Microdot is programmed serially using an I^2C serial bus interface. The I^2C interface is also used to transfer data between the Microdot and its supervisor processor. The Microdot design

hierarchy consists of six functional blocks and thirty-five sub-blocks. The Microdot block diagram in Figure 3-2 shows the overall architecture. Thick lines represent data paths and thin lines represent control signals. The Control Unit controls program execution. The Stack Unit is the data memory. The Memory Unit is the program or instruction memory. The Input/Output Unit is the interface to the eight bi-directional external data ports. The I^2C COM Unit is the interface to the I^2C bus, which is used for distributed control and data transfer. Before the functional blocks were designed the instruction set was defined.

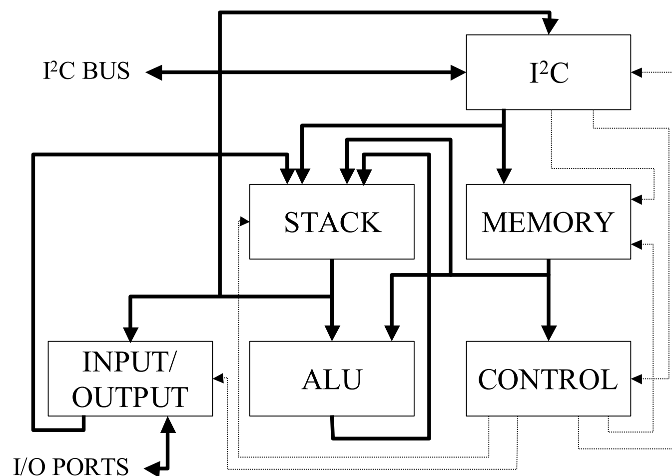


Figure 3-2. Microdot Block Diagram

3.4 Instruction Set

One of the first steps in designing any microprocessor is the definition of the instruction set. The number of instructions and their complexity determines the amount of logic required to implement them. There is a tradeoff between having a small number of simple instructions and a large number of complex instructions. Reduced Instruction Set Computers (RISC) use a small set of simple instructions. The logic needed to implement the instruction set is smaller, but it takes more instructions to execute a typical program. Thus,

more memory is required to store programs. A Complex Instruction Set Computer (CISC) uses a large set of complex instructions that requires fewer instructions and less memory to implement and store programs [5]. The Microdot is a RISC machine. The instruction set is small and quite simple, but still capable of meeting the overall design goals. The previous version of the Microdot had twenty-three instructions. The addition of the I^2C interface required three instructions for transferring data on the I^2C bus. Table 3-1 is an overview of the instruction set for the Microdot.

The instructions vary in length from one to four nibbles. The first nibble of each instruction is the operation code, or OPCODE. The OPCODE defines either the individual instruction or the instruction category. The instruction categories are load (3), ALU one-operand (4), general (5), and ALU two-operand (8). All other OPCODES define individual instructions. OPCODE C (1100) is not a legal instruction although the control logic executes it as a two-operand ALU instruction that loads the INDEX 2 register even though it is not used. The stack instructions POP and DUP are the only one-nibble instructions. The second instruction nibble is referred to as the ALU CODE. For ALU instructions, the ALU CODE defines the specific ALU operation (ADD, SUB, etc.). For load and general instructions, the ALU CODE specifies the instruction. It is basically an extension of the OPCODE. For other multi-nibble instructions, the ALU CODE is part of the instruction operand. For instance, the ALU CODE in the PUSH instruction holds the nibble to be pushed onto the stack. The third instruction nibble is INDEX 1 and is part of the instruction operand. In an ALU two-operand instruction, INDEX 1 is the offset that is added to the stack pointer to determine the

Table 3-1. Microdot Instruction Set

INSTRUCTION	*OPCODE*	*ALU CODE*	*INDEX 1*	*INDEX 2*	*LENGTH*	*TYPE*	*DESCRIPTION*
POP	0				1	STACK	Pop an element off the Stack
DUP	1				1	STACK	Push a duplicate the Top of Stack to the Stack
PUSH	2	CONST			2	STACK	Push an element onto the Stack
LDU	3	1			2	I/O	Load Upper I/O Nibble to Top of Stack
LDL	3	2			2	I/O	Load Lower I/O Nibble to Top of Stack
LDCM	3	4			2	I2C COM	Load Lower Nibble of the I2C COM Register
STSR	3	8			2	I2C COM	Set the Status Register with state of I2C COM Unit (IDR/ODR/INT)
SHL	4	8			2	ALU-1	Shift Top of Stack Left with Carry
SHR	4	9			2	ALU-1	Shift Top of Stack Right with Carry
NOT	4	A			2	ALU-1	Invert the Top of Stack
SUSP	5	1			2	INTERNAL	Suspend the Processor
INT	5	2			2	I/O	Set the Interrupt Line asserted
CLSR	5	4			2	INTERNAL	Clear the Status Register
RST	5	8			2	INTERNAL	Reset the Processor
STIO	6	STK INC			2	I/O	Store Top of Stack and Stack Element to I/O Port
STCM	7	STK INC			2	I2C COM	Store Top of Stack and Stack Element to I2C COM Register
ADD	8	0	STK INC		3	ALU-2	Add Stack Element to Top of Stack without carry
ADDC	8	1	STK INC		3	ALU-2	Add Stack Element to Top of Stack with carry
SUB	8	2	STK INC		3	ALU-2	Subtract Stack Element from Top of Stack without carry
SUBC	8	3	STK INC		3	ALU-2	Subtract Stack Element from Top of Stack with carry
AND	8	4	STK INC		3	ALU-2	And Top of Stack with Stack Element
OR	8	5	STK INC		3	ALU-2	Or Top of Stack with Stack Element
XOR	8	6	STK INC		3	ALU-2	Exclusive-or Top of Stack with Stack Element
SWAP	9	STK ADR [6..4]	STK ADR [3..0]		3	STACK	Swap Top of Stack with Stack(STK ADR)
PICK	A	STK ADR [6..4]	STK ADR [3..0]		3	STACK	Load Top of Stack with Stack(STK ADR)
SEIO	B	IO MASK [7..4]	IO MASK [3..0]		3	I/O	Set I/O Mask Register (0 = input/1 = output)
	C	ALU CODE	STK INC	LOADED	4		Unused OPCODE
BRCH	D	MASK	PC INC [7..4]	PC INC [3..0]	4	PROGRAM	Branch to PC + PC INC if MASK * SR != 0
JMP	E	JMP ADR [9..8]	JMP ADR [7..4]	JMP ADR [3..0]	4	PROGRAM	Jump to specified address
WAIT	F	STK INC	WAIT MASK [7..4]	WAIT MASK [3..0]	4	INTERNAL	Wait for masked I/O inputs to change from Top of Stack and Stack Element

* Stack Element =
Stack Pointer + STK INC

second operand in the ALU operation (Top-of-Stack Register is always the first operand). The last part of the instruction register is INDEX 2. The only four-nibble instructions are WAIT, BRCH, and JMP. The WAIT instruction allows the Microdot to conditionally suspend itself while waiting for a selected input to change. The WAIT instruction is discussed further in the section 3.2.5. The BRCH instruction is a conditional branch determined from the status register and the mask loaded into the ALU CODE. If the bitwise-and of the status register and mask is nonzero, then the Microdot will jump forward by the amount specified in INDEX 1 and INDEX 2 (INDEX 1 * 16 + INDEX 2 < 256). The status register stores the condition of the last ALU operation except when the CLSR or STSR instruction is executed. The four status conditions are Carry, Negative, Overflow, and Zero. The CLSR instruction clears the contents of the instruction register. The STSR instruction causes the status register to load with status information from the I^2C COM Unit. The JMP instruction loads the program counter with the address given in ALU CODE, INDEX 1, and INDEX 2 (ALU CODE [1..0] * 256 + INDEX 1 * 16 + INDEX 2 < 1024).

3.5 Control Unit

The Control Unit (CONUNIT) manages instruction loading and execution for the Microdot. It generates all the control signals for each functional unit. The Control Unit interface is shown in Figure 3-3. For all interface diagrams, the inputs are on the left and the outputs are on the right. The signal definitions are presented in the signal table in Appendix A. The main inputs to the Control Unit are the OPCODE and ALU CODE registers from the Memory Unit. Other important inputs are control and state signals from the I^2C COM Unit. The I^2C COM Unit operates independently from the rest of the functional blocks and needs to

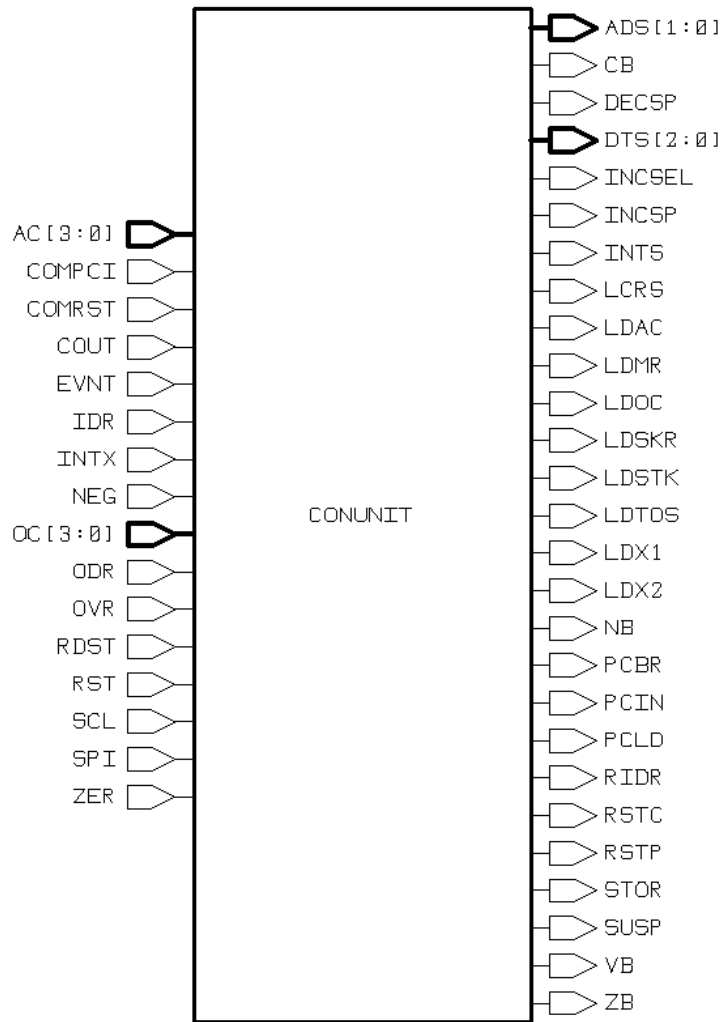


Figure 3-3. Control Unit Interface

control the Control Unit during I^2C data transfer operations. The five sub-blocks that comprise the Control Unit are the Control State Machine (CONTSTM), the Internal Control Logic (CONCNT), the Memory Control Logic (MEMCNT), the Stack Control Logic (STKCNT), and the I/O Control Logic (IOCNT). CONTSTM holds the instruction state and the status register. There are six possible instruction states. These are Fetch-OPCODE (FOC), Fetch-ALU CODE (FAC), Fetch-INDEX 1 (FX1), Fetch-INDEX 2 (FX2), Swap (SWP), and Execute (EXE). The instruction is loaded during the fetch states. Fetch-

OPCODE is the first state in an instruction cycle. Following the Fetch-OPCODE state, one and two nibble instructions enter the Fetch-ALU CODE state and then jump directly to the Execute state. The one-operand instructions, POP and DUP, have a Fetch-ALU CODE state although internal logic prevents the ALU CODE register from being loaded. This eliminated the need to have an instruction decode state immediately after the Fetch-OPCODE state. The instruction state transitions are illustrated in Figure 3-4. As can be seen, one-nibble and two-nibble instructions transition from FAC to EXE. Three-nibble instructions skip from FX1 to EXE. The SWAP instruction is the only one that transitions to the SWP state after FX1. This state is needed to facilitate reading from and then writing to the same memory location in the Stack RAM.

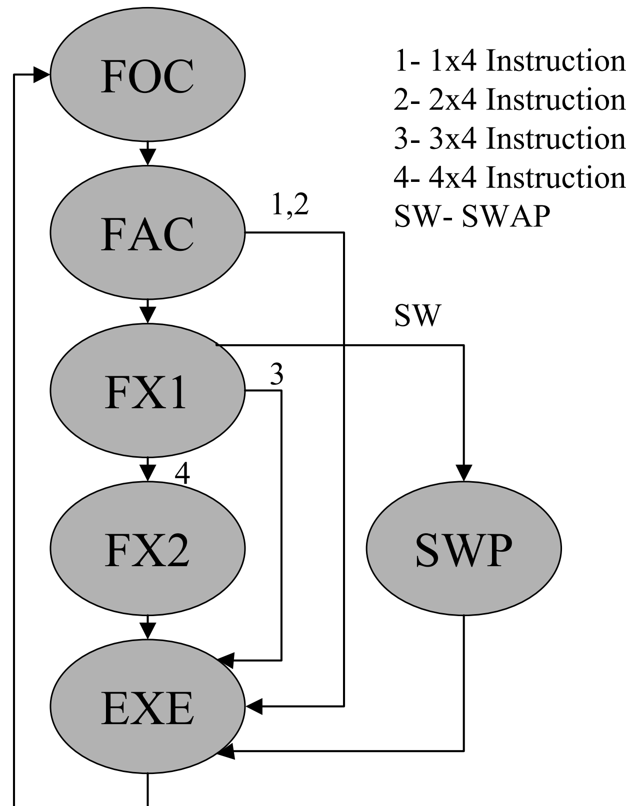


Figure 3-4. Instruction State Diagram

The control logic signals are generated according to the instruction code (OPCODE and ALU CODE), the instruction state, the status register, and the I^{2}C COM Unit state. The control logic blocks divide the control logic by the functional units. This was done to simplify the design and break the control logic into smaller and more manageable units. CONCNT generates the control signals used in the Control Unit itself. MEMCNT generates control signals for the Memory Unit. STKCNT generates control signals for the Stack Unit. The IOCNT generates control signals for the I/O blocks of the Microdot, which include the I/O Unit and the I^{2}C COM Unit.

3.6 Arithmetic and Logic Unit

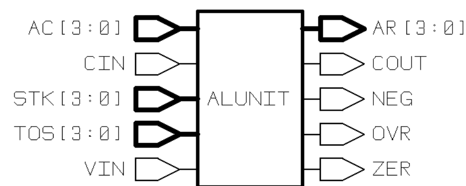


Figure 3-5. ALU Interface

The Arithmetic and Logic Unit (ALU) executes the logic function for one of the ten ALU instructions. The ALU interface is shown in Figure 3-5. The Microdot uses two's complement to represent signed numbers. Table 3-2 shows the unsigned and two's complement for four-bit numbers.

The ALU is comprised of the eight sub-blocks described below. The ALU Controller (ALUCONT) decodes the ALU CODE to activate the correct logic sub-block for the selected ALU instruction. The ALUNOT sub-block inverts the TOS Register. The ALUAND sub-block is the bitwise-and of the operands, the ALUOR sub-block is the bitwise-or, and the

Table 3-2. Microdot Number Representation

BINARY	HEX	UNSIGNED	SIGNED
0000	0	0	0
0001	1	1	1
0010	2	2	2
0011	3	3	3
0100	4	4	4
0101	5	5	5
0110	6	6	6
0111	7	7	7
1000	8	8	-8
1001	9	9	-7
1010	A	10	-6
1011	B	11	-5
1100	C	12	-4
1101	D	13	-3
1110	E	14	-2
1111	F	15	-1

ALUXOR sub-block is the bitwise-exclusive-or. The ALUSHF sub-block either shifts the Top-of-Stack (TOS) Register left or right depending on the ALUCONT control signals. The ALUADD sub-block performs four-bit addition or subtraction as directed by the ALUCONT. Both operands come directly from the Stack Unit. The first operand is always the Top-of-Stack (TOS) Register. For two operand instructions, the second operand is the Stack RAM output (STK). Stack RAM output is the RAM element addressed by the Stack Pointer (STKPTR) plus the increment loaded in the INDEX 1 Register (STKPTR + INDEX 1). The Status Logic (ALUSTR) sub-block determines the condition of the ALU operation. As noted before, the four status conditions are Carry, Negative, Overflow, and Zero. During an ALU instruction, the status output is stored by the Control Unit for use with future BRCH instructions. For instance, a program may require a branch if an ALU operation resulted in a zero output. The status register is used during the BRCH instruction to check if the last ALU result was zero and branch forward if it was. Add, subtract, and shift operations are the only

ALU instructions capable of generating a carry. The carry bit is used to perform multi-nibble addition, subtraction, and shift operations. An ADD or ADDC instruction generates a carry bit if the ALU result (AR) is greater than 15. A SUB or SUBC instruction generates a carry bit (considered a borrow) if the ALU result is less than 0. SHR and SHL instructions generate a carry if the shift-out bit is '1'. The negative bit is generated during all ALU instructions if the ALU result is a negative number in four-bit, two's-complement representation. If the most significant bit (MSB) of the ALU result is '1', the result is considered negative. The overflow bit is generated during add and subtract instructions when an overflow occurs and the ALU result may be considered invalid. Overflow occurs when the addition or subtraction results in the opposite of what is expected. For instance, when two positive numbers are added we expect a positive result, but if the result is negative an overflow has occurred. The other possible overflow conditions are listed in Table 3-3 below.

Table 3-3. ALU Overflow Conditions

	ADD/ADDC		SUB/SUBC	
TOS	+	-	+	-
STK	+	-	-	+
ALU RESULT	-	+	-	+

The zero bit is generated during all ALU instructions when the ALU result is zero. The zero bit is useful for executing a branch-on-equal. Two numbers can be compared by executing a SUB instruction followed by a BRCH instruction that masks the zero bit as the branch condition.

3.7 Stack Unit

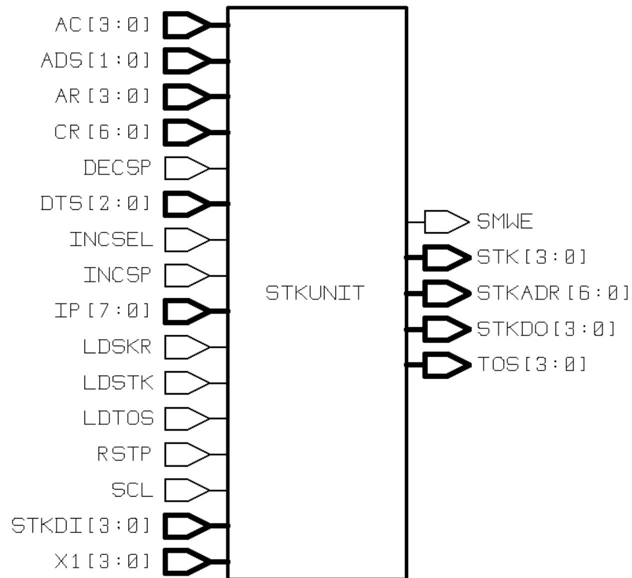


Figure 3-6. Stack Unit Interface

The Stack Unit (STKUNIT) functional block shown in Figure 3-6 is the Microdot data memory. The Stack Unit feeds operands to the ALU and stores the results of the ALU operations. It also stores data loaded from the program memory, I/O ports, and I^2C COM Unit. The Stack Unit is designed to interface the 128-nibble static random access memory (SRAM), referred to as the Stack RAM. An advantage of using SRAMs is that they can typically be designed much smaller and more efficiently than a register array of the same size. The Stack Unit is actually a modified stack design. This allows easier access to the stack elements, which makes it much simpler to use as the data memory by allowing simultaneous access to two operands. The modified stack also reduces the number of instructions needed to manage the stack. The Stack Unit is comprised of six sub-blocks. The top of the stack is actually stored in a register referred to as the Top-of-Stack Register (TOSREG or TOS). For

operand instructions (ALU or STORE), the TOSREG is always the first, or only, operand. Data is loaded to the TOSREG by the Top-of-Stack Multiplexer (TOSMUX), which selects the data input based on the instruction. The Stack Pointer (STKPTR) stores the address of the second element of the stack, which is actually the top of the Stack RAM. During a reset, the STKPTR is initialized to 00_{HEX}. The STKPTR decrements when elements are pushed onto the stack and increments when elements are popped from the stack. A PUSH instruction places the new element into the TOSREG while the TOSREG data is placed onto the Stack RAM and the STKPTR is decremented. A POP instruction loads the Stack RAM element addressed by the STKPTR to the TOSREG and then increments the STKPTR to point to the next element. The Stack Pointer Logic (STKPTRLOG) computes the increment or decrement for the STKPTR. The Stack Adder (STKADD) adds the four-bit stack increment to STKPTR to give two-operand instructions access to the elements from STKPTR to STKPTR + 15. Depending on the instruction, the four-bit stack increment comes from either the ALU CODE or INDEX 1 register. The Stack Address Multiplexer (STKADRMUX) selects the address input for the Stack RAM.

3.8 Input/Output Unit

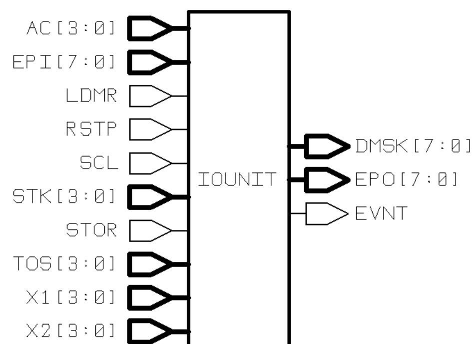


Figure 3-7. Input/Output Unit Interface

The Input/Output Unit (IOUNIT) is the functional unit that manages the interface to the eight input/output (I/O) ports. The previous Microdot design had only four I/O ports. While a design goal for the Microdot is to keep the pin count down, I also wanted a level of versatility that would make the Microdot useful for more than just the smallest designs. Plus, after examining the design, I concluded that the logic required to implement four additional I/O ports was only a small penalty so the tradeoff was made for more I/O capacity.

The Input/Output Unit interface is shown in Figure 3-7. It is comprised of only three sub-blocks. These are the Mask Register (MSKREG), the Output Register (OUTREG), and the Event Detector (EVNTDET). The MSKREG is an eight-bit register that determines whether the bi-directional I/O pads are configured as inputs or outputs. A ‘0’ stored in a MSKREG bit configures its corresponding I/O pad as an input. Conversely, a ‘1’ causes the I/O pad to drive the value from the corresponding bit in the Output Register. The MSKREG is loaded during execution of the SEIO (Set I/O Mask) instruction. The eight-bit OUTREG holds the output for the I/O ports. The value of the OUTREG is set during execution of the STIO (Store to I/O port) instruction. The Event Detector (EVNTDET) is used during the WAIT instruction to detect a data event on the input lines. The WAIT instruction puts the Microdot in a suspend state while the I/O ports are compared to the data input from TOS and STK according to the mask loaded in the INDEX 1 and INDEX 2 registers. If a mask bit is set (‘1’), then the I/O port is compared to its corresponding data input bit. If the two bits differ, the EVNTDET signals an event to the Control Unit and the Microdot resumes execution. Otherwise, the Microdot remains in the standby state waiting for data to change on the selected I/O ports.

3.9 Memory Unit

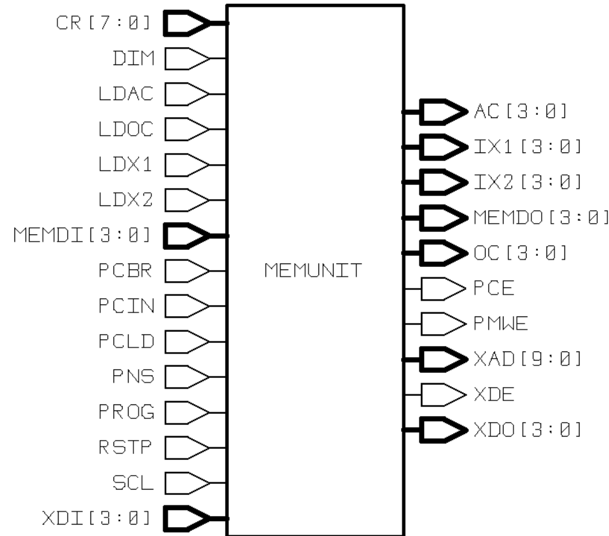


Figure 3-8. Memory Unit Interface

The Memory Unit (MEMUNIT) shown in Figure 3-8 is responsible for managing the Microdot's 1024x4 bit Program SRAM. Its two main functions are 1) delivering instructions to the Microdot and 2) loading new programs. The 1024x4 SRAM is a standard memory architecture. It has 10 address inputs, 4 data inputs, 4 data outputs, and a write enable input. Input data is written to the addressed location when the write enable input is high ('1'). When write enable is low, the addressed data is placed on the data output lines. The Memory Unit is made up of four sub-blocks, which are the Program Counter State Machine (PCSM), Program Counter Logic (PCLOG), Instruction Register (INSREG) and External Memory Multiplexer (XMEMMUX).

The PCSM is a 10-bit register that holds the current program memory address. Its 10-bit output feeds the address input of the Program SRAM as well as the external address (EXADR) output pads. The PCLOG sub-block is the logic for updating the PCSM. During a

BRCH instruction where the branch condition is set, the PCLOG adds the eight-bit branch value from the INDEX 1 and INDEX 2 registers to the current PCSM value to compute the branch address. During JMP instructions, the PCLOG loads the new address from the ALU CODE, INDEX 1, and INDEX 2 registers. Otherwise, during normal operation the PCLOG increments the PCSM value by one.

The Instruction Register (INSREG) sub-block is a combination of the four registers that hold the instruction nibbles: OPCODE, ALU CODE, INDEX 1, and INDEX 2. Each four-bit register has its own load input, which comes from the Control Unit. The four-bit data input comes from the XMEMMUX.

The External Memory Multiplexer (XMEMMUX) has two functions. The first is to provide an external memory interface so off-chip memory can be used for testing. This allows the Microdot to be tested if the internal SRAM fails. When the DIM (disable internal memory) input is asserted ('1'), the XMEMMUX selects the external memory data port (XDI) as the program memory. During normal operation, XMEMMUX selects the internal program memory (MEMDI). The second function of XMEMMUX is to select the correct nibble from the eight-bit I^2C COM Unit during the programming operation. Program data is loaded to the I^2C COM Unit in bytes, but must be written to the Program SRAM as nibbles. During programming the XMEMMUX selects the nibble based on the Program Nibble Select (PNS) input (0 = upper nibble, 1 = lower nibble).

3.10 I^2C Communication Unit

The I^2C Communication Unit (COMUNIT) is the link to the supervisor processor that controls the Microdot. I^2C COM Unit operations include loading new programs, transferring data over the I^2C bus, and receiving commands from the supervisor processor to control

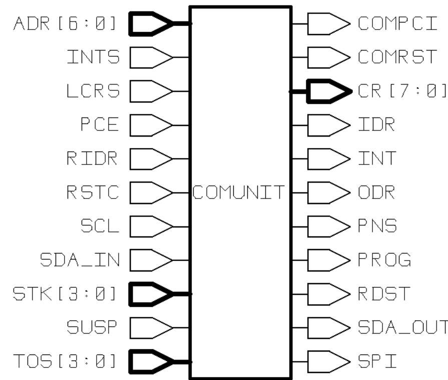


Figure 3-9. I^2C Communication Unit Interface

execution. The Microdot is designed for distributed computing and control interfacing of microdevices in satellite systems. The idea is a distributed architecture where a supervisor processor delegates computing and control tasks to a collection of Microdots that independently operate on different devices. Therefore, it is important for the Microdot to have a simple and efficient interface distributed computing. The interface requires two operations. The first operation is the programming of the Microdot. The second operation, data transfer, is equally as important. What good is a processor if you cannot get data from it? Control processes can possibly run independently, but data collection needs a path from the Microdot to the supervisor processor. The I^2C COM Unit uses the I^2C serial bus protocol introduced in Chapter 2. As I studied the I^2C bus and the Microdot design, I came up with a set of commands for data transfer, execution control, and programming that the supervisor processor could send to the Microdot as part of a command byte. The I^2C commands and command process are explained Chapter 4. The I^2C COM Unit functional block shown in Figure 3-9 consists of six sub-blocks that are briefly explained below.

The start detector (STDET) detects the start condition on the I^2C bus. Recall that the start condition is defined by a '1' to '0' transition of the SDA line while SCL is at '1'. After a start condition is detected, the I^2C COM Unit will load the address byte.

The COM Word State Machine (COMWS) holds the mode of the I^2C COM Unit. The modes are IDLE, READ ADDRESS, READ COMMAND, WRITE DATA, and PROGRAM. COMWS modes correspond to what the I^2C COM Unit is currently doing. The IDLE mode indicates that no I^2C data transfer is taking place. READ ADDRESS indicates that the I^2C COM Unit is reading the address byte that begins every I^2C data transmission to specify the device being addressed and the direction of the subsequent data transfer. READ COMMAND mode denotes that the I^2C COM Unit is reading a command byte from the supervisor processor. WRITE DATA mode indicates that the I^2C COM Unit is transferring a data byte to the supervisor processor. PROGRAM mode means that the supervisor processor is downloading a new program to the Microdot.

The COM Bit State Machine (COMBS) counts bit states and synchronizes the acknowledge state. Recall that the I^2C data byte is actually 8 data bits followed by the acknowledge state. When a data transfer begins, the COMBS is reset and then starts counting clock cycles by nine. Every ninth clock cycle is an acknowledge cycle and the COMBS signals it to the other sub-blocks of the I^2C COM Unit when the acknowledge cycle is reached. The acknowledge signal is used to synchronize other I^2C COM Unit operations.

The COM buffer (COMBUF) is the shift register that interfaces the serial data (SDA) line. When data is transferred into the Microdot, the SDA data is shifted into bit 0. When data is shifted out of the Microdot, the COMBUF is first loaded from the COM Register. The

data is then shifted out of bit 7. The COMBUF also contains the logic for I^2C address detection.

The COM Register (COMREG) is the link between the microcontroller and I^2C interface. When a byte is written to the Microdot during a write-nibble command or programming, the COMREG loads the output of the COMBUF. During the STCM (Store to COM Unit) instruction when the Microdot stores data to the I^2C COMUNIT for output on the I^2C bus, the COMREG loads the TOS and STK outputs.

The COM State Machine (COMSTM) holds the I^2C COM Unit state information and generates control signals for programming and reading the stack. The state information includes bits for Suspend, Interrupt, Output Data Ready (ODR), and Input Data Ready (IDR). The Suspend and Interrupt states are actually Microdot state information. The Suspend bit is included as part of the COMSTM because the execution is controlled over the I^2C bus by the supervisor processor. The interrupt line is integrated with the I^2C bus to signal the supervisor processor that the Microdot has data ready to send. The interrupt bit is set during the INT (Set Interrupt) instruction and is reset when the output data has been sent over the I^2C bus.

The COM Logic (COMLOG) sub-block has internal control logic for loading the COMBUF and COMREG and external control logic for I^2C reset and byte acknowledge.

3.11 Status Multiplexer

The status multiplexer shown in Figure 3-10 is the test structure that allows visibility to several of the Microdot's internal data paths. It is an eight input by four-bit multiplexer that selects from the following inputs:

0: ALU Result
1: Top-of-Stack Register
2: Stack RAM Output
3: Status Register (Carry, Negative, Overflow, Zero)
4: OPCODE Register
5: ALU CODE Register
6: COM Register [7..4]
7: COM Register [3..0]

The three-bit select input comes from off-chip. This configuration allows testing of the OPCODE and ALU CODE registers, the TOS and STK data, the ALU result, and COMREG. This allows an internal view of exactly what is happening within the Microdot that cannot be gained by watching the external outputs. This test structure would not be included in a production version of the Microdot.

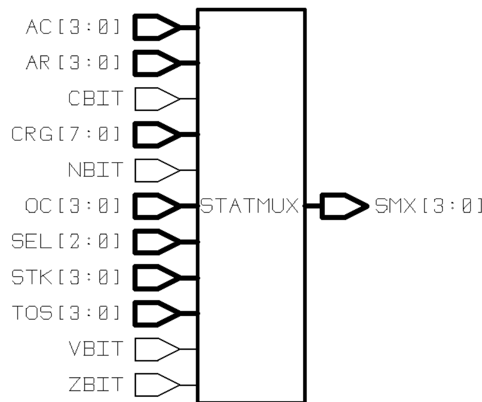


Figure 3-10. Status Multiplexer Interface

3.12 Summary

In this chapter, I have provided an overview of the Microdot design. I first presented the overall architecture and the instruction set. Next, I reviewed the six functional blocks that comprise the Microdot microcontroller.

4. Design Implementation

4.1 Introduction

In Chapter 4, I explore design methods that are useful for the creation of the Microdot. I look again at the functional units and sub-blocks and cover how they are interconnected. I also present an in-depth explanation of the I^2C Communication Unit design and operation.

The Microdot is essentially a scaled-down microprocessor. It has a basic instruction set with standard arithmetic and logic functions. It is designed to be small. There are no high performance design methods like pipelining or memory caching and it is not designed to run subroutines. Its role is really to run a single program to interface a single device that needs simple control. The most complicated part of the design is the interface for the I^2C serial bus. Of course, this is the part of the design that makes the Microdot more than just a mini-microcontroller. It is the basis for the Microdot distributed computing concept.

4.2 State Machine Design

The state machines in the Microdot were designed using a “one-hot” approach. Traditional state machines use n flip-flops to represent $< 2^n$ states. The one-hot approach uses a single flip-flop per state. Only one of the flip-flops holds a ‘1’, which represents the active state. For small state machines with simple state transition tables, the one-hot design approach makes them very simple to implement. It simplifies the next-state logic and no decode logic is needed to determine the active state. Eliminating decode logic can make this approach also work well for large state machines that have a small number of state transition cycles.

One of the drawbacks to the one-hot approach is the possibility of errors. If two flip-flops are simultaneously set to '1' by a bit upset, the state machine exists in two states at once. This is an illegal condition and will cause errors. To make the one-hot state machine more robust, extra logic is needed to detect the error condition and reset the state machine. This complicates the design, which reduces any advantage over traditional state machine design. Also, for the radiation-hardened gate array library, the SEU-hardened flip-flops are very large (24X larger than 2 input NAND). The area penalty for extra flip-flops quickly outweighs the combinational logic penalty of traditional designs. For comparison, the Control State Machine was designed with both the one-hot and traditional design methods. Table 4-1 summarizes the design comparison. You can see that the one-hot implementation has fewer gates and smaller combinational logic area, but the size of the SEU hardened flip-flops makes the one-hot design 1.5 times larger than the traditional design.

Table 4-1. State Machine Design Comparison

Approach	Flip-flops	Combinational Gates	Combinational Cell Area	Flip-Flop Cell Area	Total Cell Area
One-hot	6	10	16	144	160
Traditional	3	25	37	72	109

The state machine design is one area of redesign I would accomplish before pressing forward with a full fabrication of the Microdot in the RHBD gate array library. Redesigning the state machines will make the Microdot smaller and more robust to SEU.

4.3 Gated Clocking

One of the methods for reducing power consumption in the Microdot is the use of gated clocking. Gated clocking reduces power consumption by reducing the number of inputs that switch each clock cycle. Figure 4-1 shows a gated clock register versus a standard continuously clocked register design. The gated clock register clocks the flip-flops only when the load input is asserted. This register uses an asynchronous clear. The continuously clocked register clocks each flip-flop on every clock cycle. This requires logic to feed back the output to maintain the value when the load or reset input is not asserted. This register uses a synchronous clear.

The multi-cycle design of the Microdot allows the use of gated clocking. Multi-cycle refers to the fact that each instruction is executed in multiple clock cycles. For the Microdot, an instruction requires 3-5 clock cycles depending on the type. Most of these cycles are required for loading an instruction opcode or operand into one of the four-bit registers that make up the instruction register. Thus, there is no need to continuously clock these registers. The logic synthesis tools do not have the ability to synthesize behavioral logic with multiple or gated clocks. Therefore, the structural logic for the sequential elements of the Microdot had to be hand-designed. In anticipation of this inadequacy, elements of sequential and combinational logic for each register were separated into sub-blocks. This allowed for easy synthesis of the combinational blocks and manual design of sequential blocks.

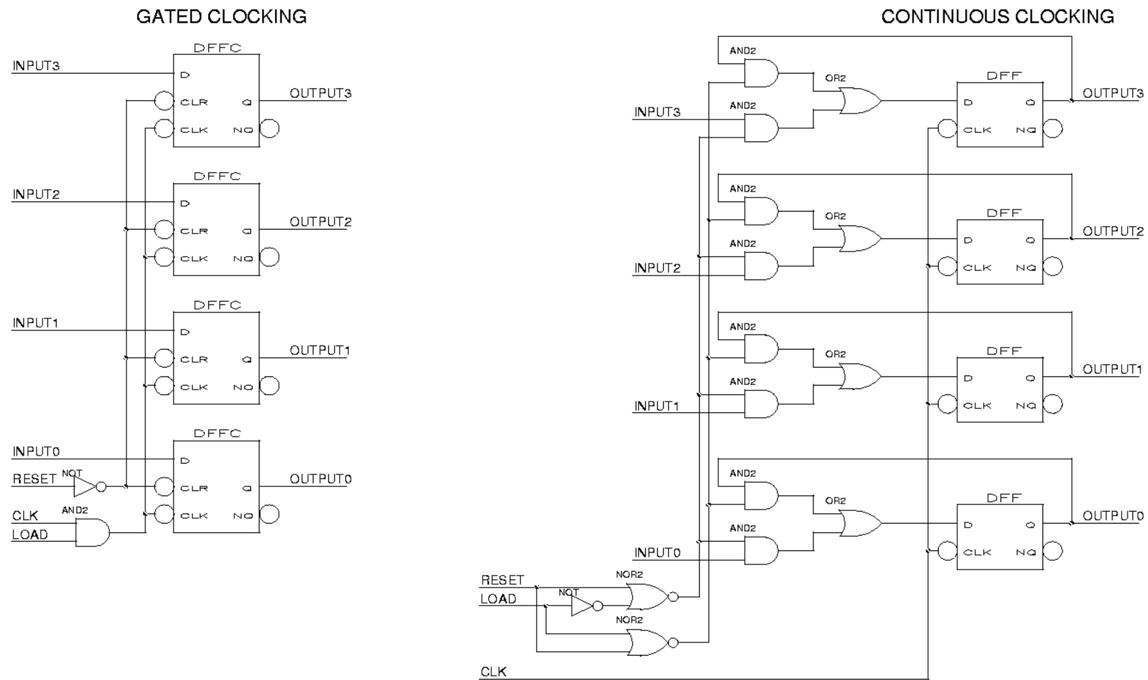
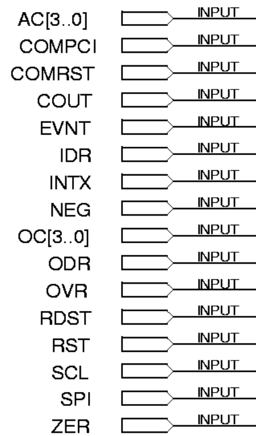
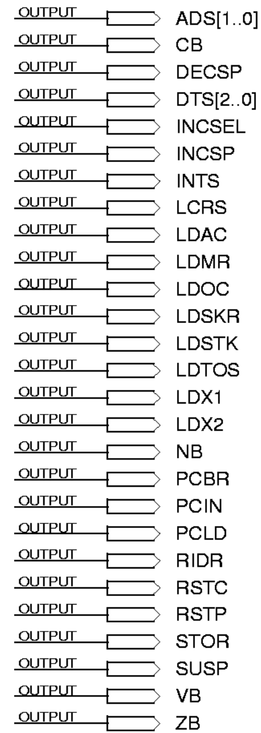


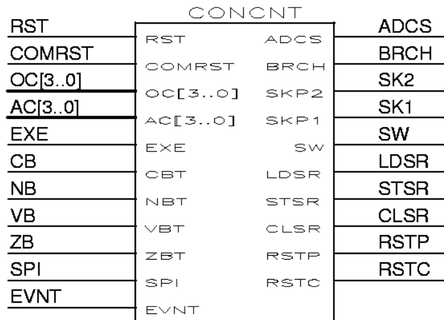
Figure 4-1. Gated Clocking versus Continuous Clocking. Gated Clocking reduces power consumption by gating the clock, which reduces clock fanout and switching capacitance.

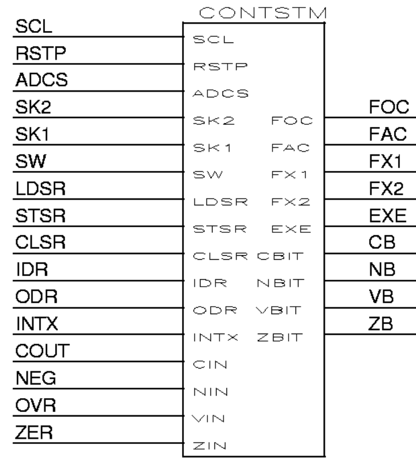
4.4 Control Unit

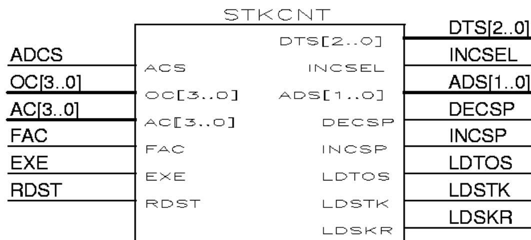
Figure 4-2 shows the connectivity of the Control Unit (CONUNIT). The wiring is not shown, but can be followed by tracing the signal names to their source and destinations. The OPCODE (OC) and ALU CODE (AC) inputs from the Memory Unit define the instruction that is currently being executed. The instruction input and control state stored in the Control State Machine (CONTSTM) define the output of the control signals. The CONTSTM also contains the status register, which stores the status output from the last ALU instruction or the I^2C COM Unit status stored during the Set Status Register (STSR) instruction. The ALU status inputs are COUT, NEG, OVR, and ZER. The I^2C COMUNIT status inputs are IDR, INTX, and ODR. The internal control and reset logic is determined in

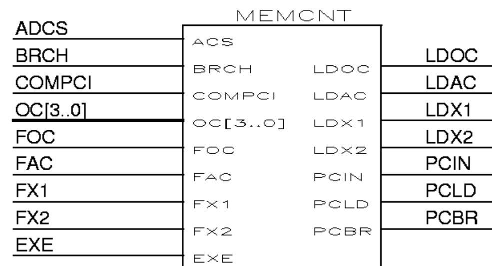












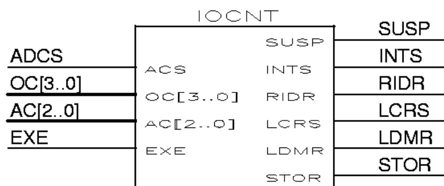


Figure 4-2. Control Unit Interconnect Diagram

the CONCNT block. CONCNT controls the CONTSTM instruction cycle based on the OPCODE and external inputs. It is also responsible for status register loading and reset and testing the status register to determine if the branch condition is met.

The external control logic is broken down by the functional unit control signals. Stack Control logic (STKCNT) uses instruction input and control state to determine the control signals for the Stack Unit. Memory Control Logic (MEMCNT) resolves the Memory Unit control signals. The I/O Control Logic (IOCNT) resolves the control signals for both the I/O Unit and the I^2C COM Unit.

The I^2C COM Unit primarily operates independently from the Microdot. It is constantly monitoring the I^2C bus looking for its unique address to be sent in an address byte. The I^2C COM Unit needs control over the Control Unit so it can execute commands sent from the supervisor processor. The SPI input signals the Control Unit that the I^2C COM Unit has suspended the Microdot. The COMPCI input is used during the programming process to signal the Control Unit to increment the program counter. The COMRST signal is the reset input from the I^2C COM Unit. The RDST input signals that the I^2C COM Unit is reading a stack element. I^2C COM Unit operation is examined in section 4.9.

4.5 Arithmetic and Logic Unit

The Arithmetic and Logic Unit (ALU) is shown in Figure 4-3. The ALU data inputs come from the Stack Unit. They are the Top-of-Stack Register (TOS) and Stack RAM output (STK). The CIN and VIN inputs come from the Control Unit. CIN is the carry-in input for the ADDC, SUBC, SHL, and SHR instructions. VIN is the overflow bit input. It is included so the overflow bit can be maintained during non-arithmetic instructions.

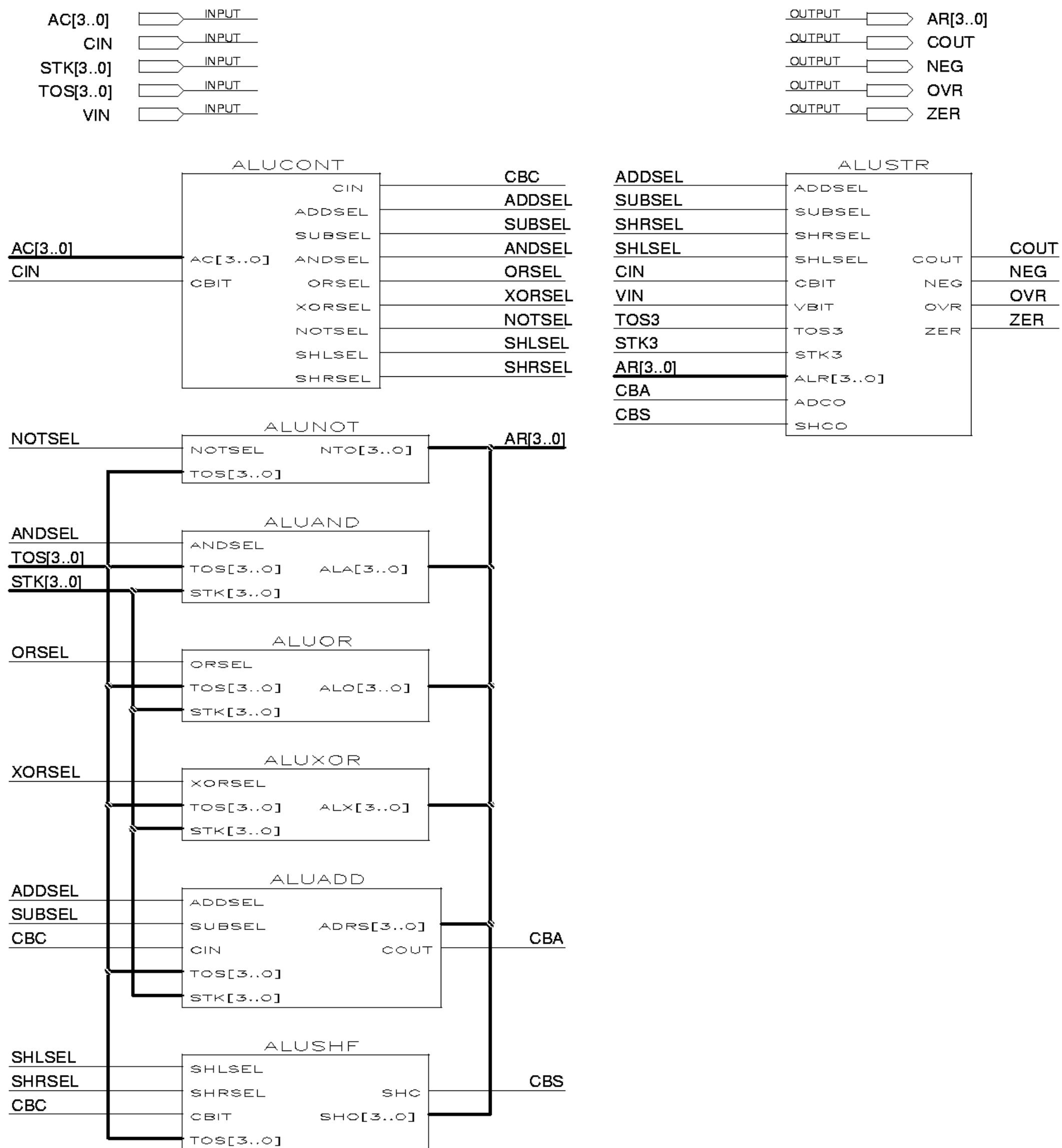


Figure 4-3. Arithmetic and Logic Unit Interconnection Diagram

The ALU Controller (ALUCONT) selects the logic function based on the ALU CODE. The outputs of the six logic blocks are all connected to the ALU Result (AR) output. The select input on each logic block enables a tri-state buffer to drive the result to the ALU Result output. The ALU Result bus is a form of distributed multiplexing that eliminates the need for a result multiplexer. The drawback to multiplexing with tri-state buses is that

leakage currents can cause a reduction in bus output voltage swing that may cause failure. However, for this implementation, the number of outputs driving the bus is small so leakage currents caused by ionizing radiation should remain small and voltage margins will not be adversely affected. The ALUCONT also controls the carry bit to the Adder and Shifter sub-blocks. It withholds the carry-bit during ADD and SUB instructions. ADDC, SUBC, SHL, and SHR instructions all receive the carry bit from the Control Unit status register.

The Adder (ALUADD) performs all add and subtract operations. The ADDSEL input enables the Adder tri-state buffer. The normal operation is addition. The SUBSEL input activates the subtraction logic. The adder uses ripple-carry adder/subtractor logic. Size is the issue with the Microdot and ripple-carry design is the minimum-area approach. Plus, ripple-carry delay through four bits is not the critical path in the circuit.

The Shifter (ALUSHF) is the sub-block for SHL and SHR instructions. The SHLSEL and SHRSEL inputs control which operation is performed. These are shift-with-carry (five-bit rotate) instructions so shifting without a carry requires a CLSR instruction prior to the Shift instruction.

The Status Logic (ALUSTR) determines the status of the ALU operation. There are four possible status conditions. Carry-out (COUT) signals that 1) the add operation generated a carry, 2) the subtract operation generated a borrow, or 3) the shift operation shifted out a '1'. Negative (NEG) signals that the result represents a negative number in four-bit two's complement. Overflow (OVR) indicates that arithmetic overflow occurred. It is only generated during add and subtract operations. The Zero (ZER) output signals that the four-bit ALU Result was zero. Table 4-2 shows the status bits output by each ALU instruction.

Table 4-2. Status Bit Output by ALU Instruction

Instruction	COUT	NEG	OVR	ZER
SHL	X	X		X
SHR	X	X		X
NOT		X		X
ADD	X	X	X	X
ADDC	X	X	X	X
SUB	X	X	X	X
SUBC	X	X	X	X
AND		X		X
OR		X		X
XOR		X		X

4.6 Stack Unit

The Stack Unit and Stack RAM are the Microdot data memory. The six sub-blocks that comprise the Stack Unit interface the 128x4 Stack RAM. The Stack Unit creates a modified stack as discussed in Section 2.3. The purpose of the modified stack is to gain the advantage of the stack architecture and improve the flexibility in selecting data elements. Normal stack processors only have access to elements at the top of the stack. The modified stack allows simultaneous access to the top of the stack and the one of the top 16 elements. Also, the SWAP and PICK instructions are used to gain access to the entire stack. Essentially, the modified stack reduces the number of instructions needed to move data around the stack. The top of the stack is actually stored in a register separate from the Stack RAM. The stack pointer points to the top of the Stack RAM, which is actually the second element on the stack. Additionally, the Stack Unit contains an adder and multiplexer for addressing the top 16 Stack RAM elements during two-operand instructions and any of the Stack RAM elements during SWAP and PICK instructions. The Stack Unit interconnection diagram is shown in Figure 4-4. The sub-blocks are described below.

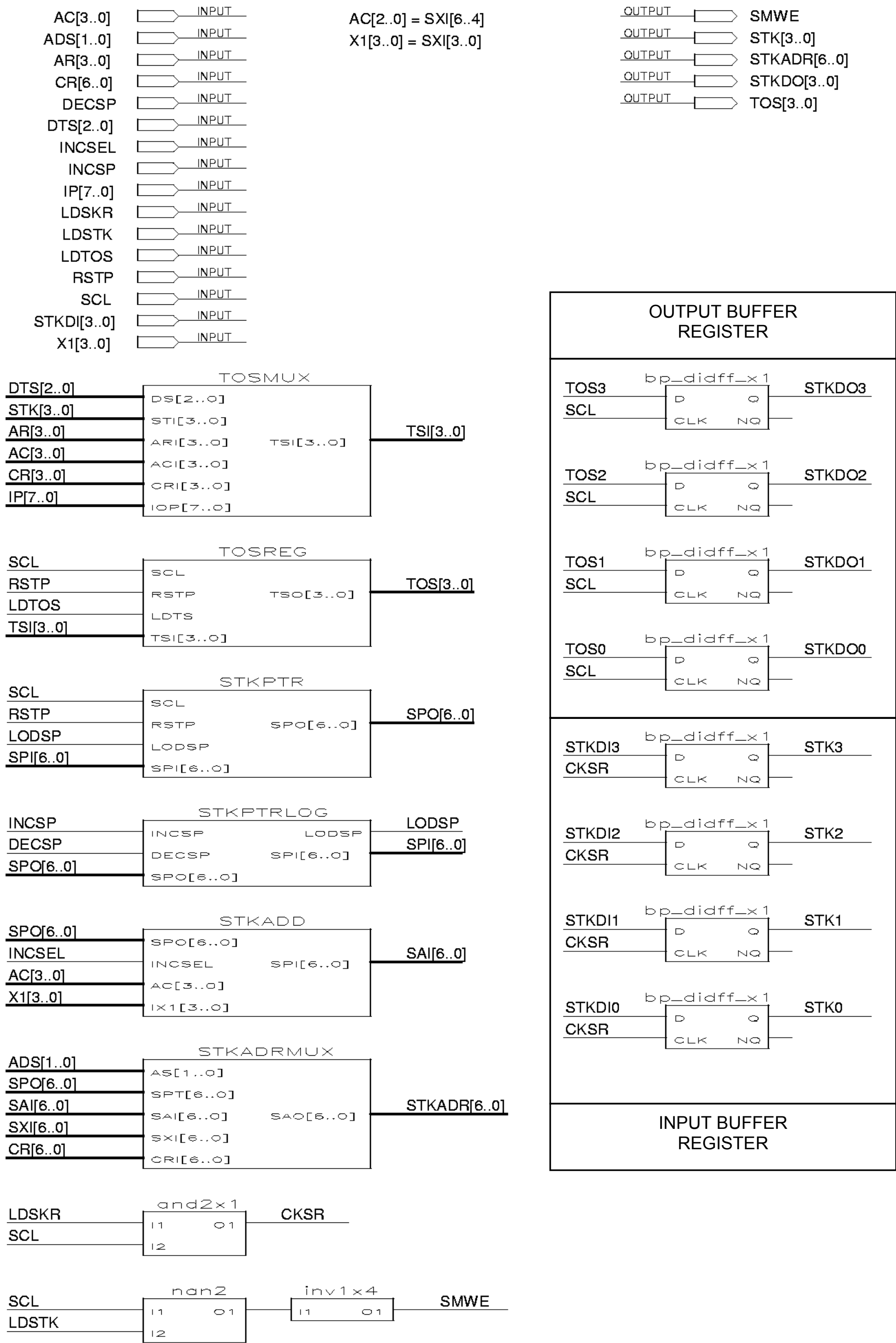


Figure 4-4. Stack Unit Interconnection Diagram

4.6.1 Stack Unit Sub-blocks. The Top-of-Stack Register (TOSREG) is the four-bit register that stores the first element on the stack. This register is the first or only operand in all ALU instructions. All loaded data and ALU result data is first stored to the TOSREG.

The Top-of-Stack Multiplexer (TOSMUX) is a six input by four-bit multiplexer that selects the input to the TOSREG from six data sources. Select input comes from the Control Unit. The data sources are:

1. Stack RAM
2. ALU Result
3. ALU CODE Register
4. COM Register [3..0]
5. I/O Port Upper Nibble [7..4]
6. I/O Port Lower Nibble [3..0]

The Stack Address Multiplexer (STKADRMUX) is a four input by seven-bit multiplexer that selects the address for the Stack RAM. Select input comes from the Control Unit. Data sources are:

1. Stack Pointer
2. Stack Adder
3. INDEX 1 [2..0] and INDEX 2 [3..0]
4. COM Register [6..0]

The STKADRMUX works as follows. POP, DUP, and PUSH instructions use the Stack Pointer to address the Stack RAM. Two-operand instructions use the Stack Adder to address the Stack RAM. SWAP and PICK instructions use the INDEX1 and INDEX2 registers to address any of the 128 Stack RAM elements. The COM Register input is used during the I^2C READ STACK command to transfer a Stack RAM element to the COM Register.

The Stack Pointer (STKPTR) is a seven-bit register that stores the address of the last element pushed onto the Stack RAM. The Stack Pointer Logic (STKPTRLOG) is the logic sub-block that increments and decrements the STKPTR. When elements are pushed onto the

Stack RAM with either the PUSH or DUP instructions, the STKPTRLOG decrements the STKPTR. When elements are popped off the Stack RAM with the POP instruction, the STKPTRLOG increments the STKPTR.

The Stack Adder (STKADD) adds a four-bit increment to the STKPTR to select one of the top 16 elements of the Stack RAM. For the STIO, STCM, and WAIT instructions, the ALU CODE register is selected as the increment. The ALU two-operand instructions use the INDEX 1 register as the stack increment.

The Stack RAM is the 128 x 4 static random access memory (RAM) that stores all data elements except the top of the stack. The interface signals to the Stack RAM are the STKADR, STKDO, and SMWE outputs and the STKDI input. STKADR is the seven-bit address that comes from the STKADRMUX. STKDO is the four-bit data input to the Stack RAM that comes from the TOSREG via the stack output buffer. SMWE is the write-enable signal for the Stack RAM. STKDI is the data output of the Stack RAM that is loaded into the stack input buffer.

It is important from a programming aspect to examine exactly how the Stack Unit operates. The PUSH and DUP instructions are the only instructions that transfer the TOSREG to the Stack RAM. Load instructions (LDU, LDL, LDCM) replace the TOSREG with the selected data source, but do not also push the TOSREG to the Stack RAM. Therefore, if the TOSREG data needs to be saved for later processing, it should be pushed to the Stack RAM with the DUP instruction. The POP instruction transfers the top element of the Stack RAM to the TOSREG. The ALU result is stored to the TOSREG, which is also the first operand, so if the first operand needs to be saved, the TOSREG must be duplicated to the Stack RAM before the ALU operation.

The STKPTR resets to address 00_{HEX}. A PUSH or DUP instruction decrements the STKPTR before the element is written to the Stack RAM. If no POP instruction occurs before the first PUSH instruction, the first PUSH instruction writes the ALU CODE to the TOSREG and writes the TOSREG (0 after reset) to Stack RAM address $7F_{HEX}$. The Stack RAM is circular, which means that when the STKPTR reaches the Stack RAM boundaries (0_{HEX} or $7F_{HEX}$), the increment or decrement simply rolls the address over ($7F_{HEX}$ -> 0_{HEX} for increment; 0_{HEX} -> $7F_{HEX}$ for decrement). A decrementing stack was selected to simplify the logic for the modified stack.

4.6.2 Stack Buffer Registers. The Stack Unit has both input and output buffer registers. These are the positive edge triggered flip-flops shown in Figure 4-4. The input buffer register is used to synchronize loading from the asynchronous Stack RAM. Additionally, it is needed for correct execution of the SWAP instruction. Recall that the SWAP instruction exchanges the TOSREG with any of the 128 Stack RAM elements. Since it is not possible to simultaneously read from and write to the Stack RAM, an extra register is required to temporarily hold the Stack RAM element so the TOSREG and Stack RAM can be written to simultaneously.

The stack output buffer register synchronizes the write from the TOSREG to the Stack RAM. The write operation occurs at the falling edge of the clock. I found through testing that the TOSREG was writing through to the Stack RAM. This was caused because the TOSREG loads before the Stack RAM write enable is de-asserted causing the TOSREG value to be lost. This error condition is illustrated in Figure 4-5. In this figure notice that the clock skew between the TOSREG and Stack RAM can cause the Stack RAM to load the new TOSREG value to the Stack RAM during the PUSH instruction. The PUSH instruction is

supposed to load a new value to the TOSREG while the current TOSREG is written to the Stack RAM. The timing error causes the current TOSREG value to be lost. The output buffer loads the TOSREG on the rising edge of the clock. This holds the TOSREG value over the falling edge of the clock while with TOSREG and Stack RAM load the new values. This prevents write-through of the new TOSREG value back to the Stack RAM.

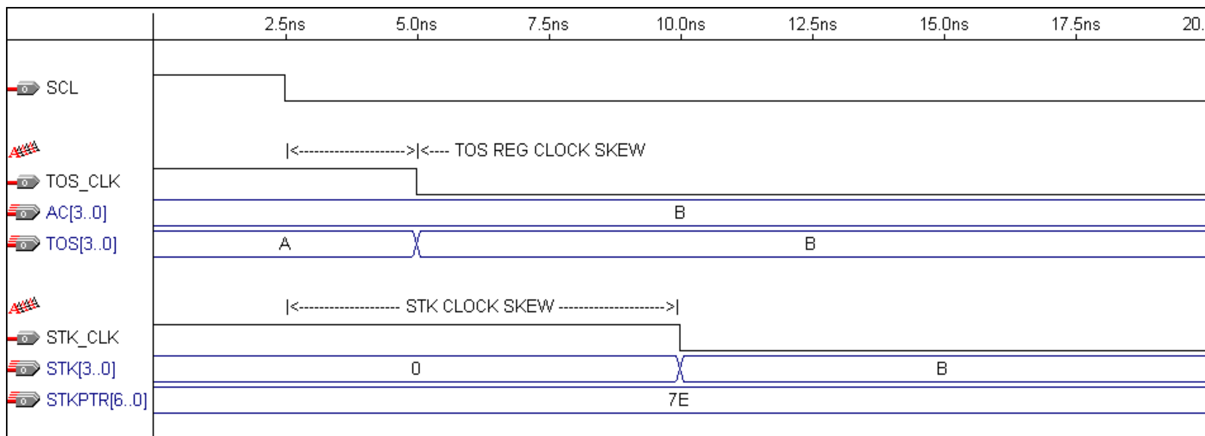


Figure 4-5. Stack Timing Errors During PUSH Instruction. Clock skew between TOS and STK causes new TOS value (B) to write-through to STK. Old TOS value (A) is lost.

Figure 4-6 shows the timing of the SWAP instruction and the role of the stack buffer registers. These registers are temporary storage for the TOSREG and Stack RAM values that synchronize the switch of the elements from one to the other. The Stack RAM value to be swapped is held in the stack input buffer register during the Execute instruction state. This way, the Stack RAM value can be written to the TOSREG as the TOSREG value is written to the Stack RAM. The stack output buffer loads on the rising-edge of the clock and holds the TOSREG value for Stack RAM input. This way, there cannot be write-through from the Stack RAM to the TOSREG and back to the Stack RAM at the negative clock edge during the Execute instruction state.

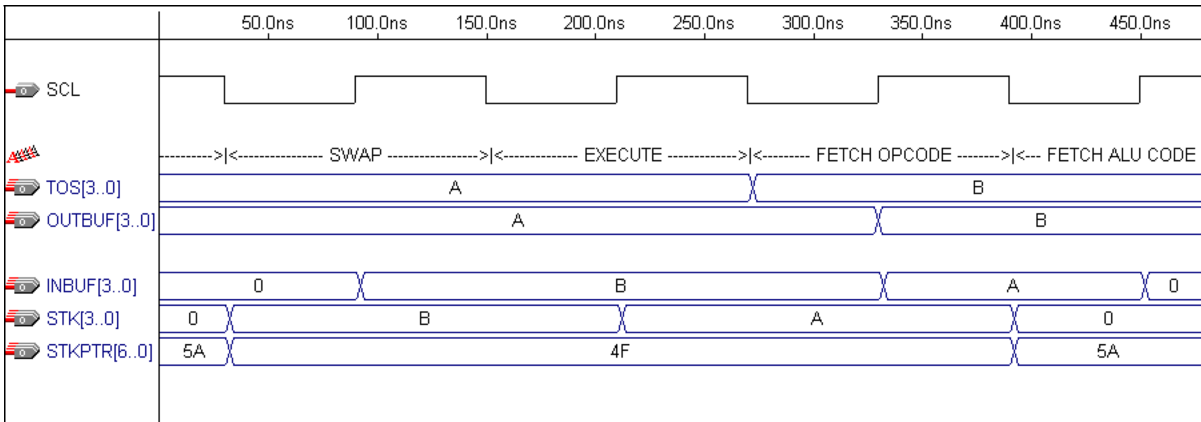


Figure 4-6. SWAP Instruction Timing. Stack Input Buffer Register loads on the positive clock edge during the SWAP state and holds the value until the positive clock edge during the FETCH OPCODE state. This holds the old STK value for loading to the TOS while the TOS value is written to the Stack RAM during the last half of the EXECUTE state. The Stack Output Buffer holds the old TOS value stable while it is being written to the Stack RAM during the second half of the EXECUTE state.

4.7 Input/Output Unit

The Input/Output Unit is shown in Figure 4-7. This simple unit consists of two eight-bit registers and event detection logic. The role of the I/O Unit is to interface the eight bi-directional pads. The direction of the pad is determined by its enable input, which is an asserted-low input. To drive the pad as an output, the enable input should be '0'. Otherwise the pad is just an input.

The Mask Register (MSKREG) is what determines the direction of the pads. The Set I/O Mask instruction (SEIO) writes the ALU CODE and INDEX 1 inputs to MSKREG. This register loads on the negative clock edge when the LDMR input is asserted. ALU CODE[3..0] is stored to MSKREG[7..4] and INDEX 1[3..0] is stored to MSKREG[3..0]. A '0' stored in the MSKREG bit configures the pad as an input while a '1' configures it as an output. A reset sets all pads as inputs.

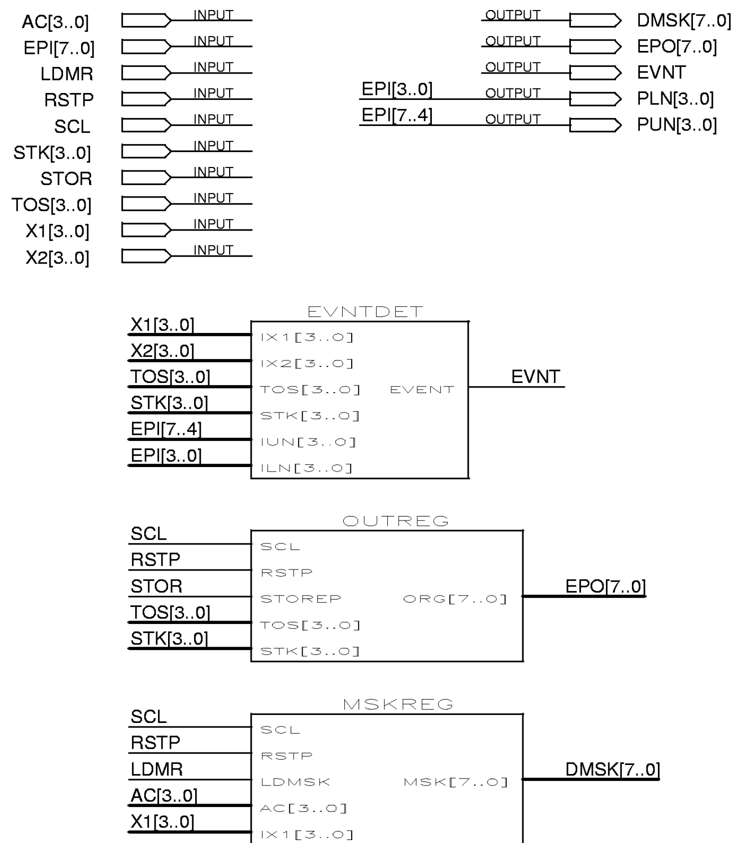


Figure 4-7. Input/Output Unit Interconnection Diagram

The Output Register (OUTREG) stores the output data for the bi-directional pads. If a pad is configured as an output, the bit stored in the OUTREG is driven to the output pad. During the Store to I/O port (STIO) instruction, OUTREG is loaded on the negative clock edge when the STOR input is asserted. TOS[3..0] is stored to OUTREG[7..4] and STK[3..0] is stored to OUTREG[3..0].

The Event Detection Logic (EVNTDET) is used during the WAIT instruction to determine that an event has taken place on the inputs and the Microdot should resume execution. During the Execute instruction state of the WAIT instruction, the Control Unit suspends the Microdot while the external inputs (EPI) are compared to the Microdot data

input (TOS and STK) according to the INDEX 1 and INDEX 2 inputs. INDEX 1 and INDEX 2 are the event mask. A '1' in the event mask means that, for that bit, a difference between the Microdot data input and the external input generates an event (EVNT = '1') and the Microdot resumes execution. A ‘0’ in the event mask bit means that the data input bit is not compared to the external input bit so no event can be generated by that bit. This lets the programmer select which inputs can generate events.

4.8 Memory Unit

The Memory Unit (MEMUNIT) is the interface for the 1024x4 SRAM that contains the Microdot instructions. During normal execution, the Memory Unit loads the program data to the instruction registers and advances the program counter. The normal control signals come from the Control Unit. During programming, however, special control signals and data come from the I^2C COM Unit. The Memory Unit Interconnection Diagram is shown in Figure 4-8. The four sub-blocks are explained below.

4.8.1 Memory Unit Sub-blocks. The Program Counter State Machine (PCSM) holds the current program memory address. The PCSM outputs feed the address inputs of the Program RAM and the external address output pads (EXADR) that allow off-chip memory interfacing. The Program Counter Logic (PCLOG) is responsible for computing the next PCSM value. PCLOG performs one of three functions specified by the Control Unit PC signals: PCIN, PCBR, and PCLD. If the PCIN input is asserted, PCLOG increments the current PCSM output by one. The PCBR signal is asserted when a conditional branch is taken. When this happens, PCLOG adds the eight-bit value in the INDEX 1 and INDEX 2 registers to the PCSM. This allows the program to conditionally branch forward up to 256 nibbles in the program. The PCLD signal is asserted when a JMP instruction is executed. In

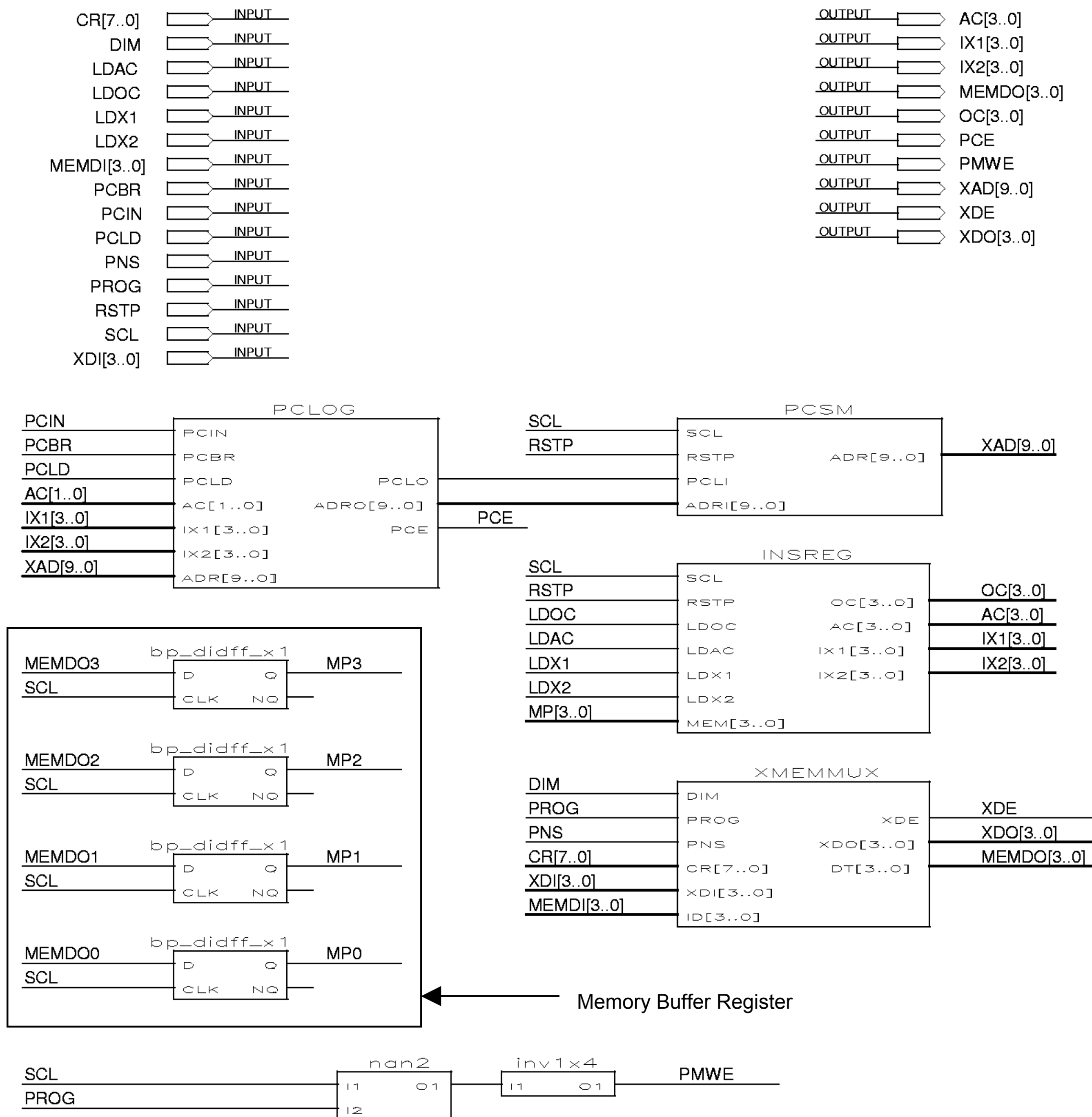


Figure 4-8. Memory Unit Interconnection Diagram

this case, PCLOG feeds the 10-bit value from the ALU CODE [1..0], INDEX 1, and INDEX 2 registers directly to the PCSM inputs. The PCLOG signals the PCSM to load when any of these PC input signals is asserted. The PCLOG also generates the PCE signal (Program Counter End) which signals the I^{2}C COM Unit that the memory is full and that programming should be terminated.

The Instruction Register (INSREG) holds all four parts of the current instruction, which are OPCODE, ALU CODE, INDEX 1, and INDEX 2. The OPCODE register is always loaded at the beginning of each instruction cycle. The Control Unit loads the remaining registers according to the OPCODE. Refer to Table 3-1 for the length of each instruction.

The External Memory Multiplexer (XMEMMUX) is a test structure that was created to interface an off-chip program memory. The External Data (EXDAT) ports are bi-directional ports for external memory interfacing. Combined with the External Address (EXADR) and External Memory Write Enable (EXMWE) output ports, they provide a complete interface to the off-chip memory. During normal operation, the Disable Internal Memory input (DIM) is low ('0') and the XMEMMUX selects the output of the internal Program RAM. Also during normal operation, the XMEMMUX configures the EXDAT ports as outputs and drives them with the value from the internal SRAM. This is another feature of the XMEMMUX created for testing purposes that allows viewing of the internal Program RAM during normal operation. When the DIM input is set to '1' to disable the internal program memory, the XMEMMUX configures the XDAT pads as inputs and selects XDAT as the memory input. A second function of the XMEMMUX is to multiplex the correct nibble from the COM Register to the Program RAM data input lines during programming. Recall that the COM Register is an eight-bit register that holds two program nibbles. The Program Nibble Select (PNS) input specifies which nibble is input to the Program RAM. During programming, the XMEMMUX configures the EXDAT ports as outputs and drives them with the value of the selected COM Register nibble to be written to memory. The Program Memory Write Enable (PMWE) output signals a write to the external

memory. This both programs the external memory (if present) and/or provides an external view of the programming process during testing. In a production version of the Microdot that did not contain the test structures, program nibble multiplexing would be the sole function of the XMEMMUX.

4.8.2 Memory Buffer Register. The positive-edge triggered flip-flops (bp_didff_x1) are the memory buffer register. This register holds the memory output stable around the falling edge of the clock when the instruction registers are loaded. This is designed to eliminate timing problems caused by loading the PCSM at the falling edge of the clock.

The memory buffer register synchronizes memory loading. The PCSM, which is the address input to the Program RAM, updates at the falling edge of the clock. The instruction registers are also loaded at the falling edge of the clock. This means that the Program RAM address is changing as the registers are loading. If the Program RAM output changes quicker than the instruction registers loads the memory output, incorrect data will be loaded. This situation can be caused by clock skew and other circuit delays that lead to the RAM outputs not meeting hold times for the instruction registers. The memory buffer register eliminates this timing problem by loading the memory at the rising edge of the clock. Assuming the rising edge occurs after the memory output is stable, the memory buffer register will load correctly. The memory buffer register then holds the memory output stable around the falling-edge of the clock when the instruction registers load. Memory loading with the memory buffer register is shown in Figure 4-9.

If the delay times can be verified to meet hold times, the memory buffer registers are not needed. However, for devices in an ionizing radiation environment, circuit delay times can be adversely affected. This leads to unpredictable delay times that cannot be guaranteed.

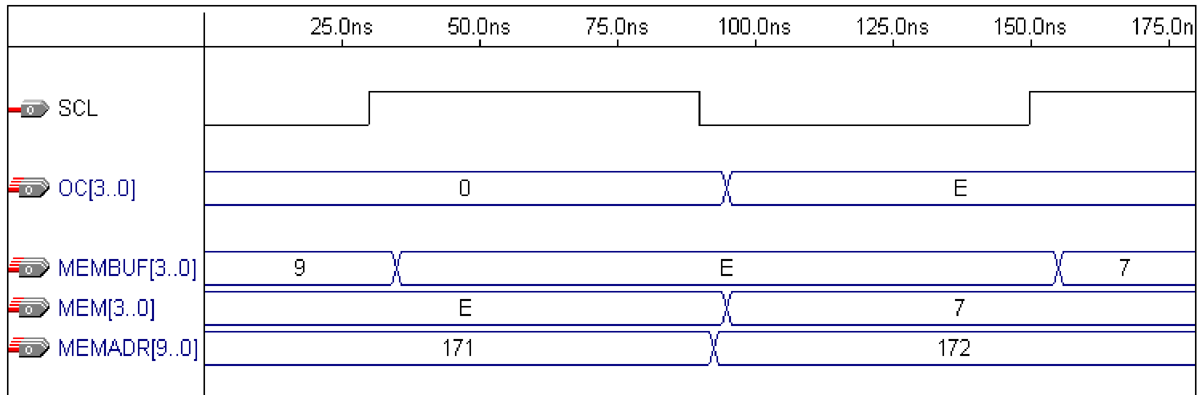


Figure 4-9. Memory Buffer Register Timing Diagram. Loading the memory buffer (MEMBUF) register on the rising edge of the clock keeps the OPCODE input stable when new data is being loaded. This eliminates timing problems caused if the memory output changes to quickly after the memory address changes.

Adding memory buffers reduces operating speed, but eliminates timing errors. Timing errors are eliminated by slowing the clock to meet the maximum delay time of the Program RAM.

Another method to eliminate timing problems is to load the PCSM on the rising edge of the clock. This would require changing the control logic for program counter increment. In retrospect, this would have been a better approach. This would have eliminated the need for the memory buffer register, which would have eliminated the area consuming flip-flops from the design.

4.9 I^2C Communication Unit

The Microdot is a simple microcontroller. The usefulness of the Microdot is not just as a simple and low-power microcontroller, but also as a microcontroller that is designed for distributed computing. Much of my design effort was spent on the distributed computing interface. Once the I^2C bus was chosen as the serial bus protocol, the effort focused on creating an I^2C Communication Unit that facilitated a useful and efficient distributed computing system for the Microdot. The next several sections explain the I^2C interface.

The I^2C Communication Unit (I^2C COM Unit) Interconnection Diagram is shown in Figure 4-10. It is the largest and most complicated functional block in the Microdot. Without it, however, the Microdot would just be a mini-microcontroller. It has three different state machines used to read the I^2C serial data and execute the I^2C distributed computing functions, which include data transfer, programming, and execution control. The I^2C COM Unit sub-blocks are described below.

The Start Detector (STDET) detects the start condition on the I^2C bus. The start condition indicates the start of a data transmission. It is defined as the high-to-low transition of the SDA line while the SCL line is high. STDET signals the COM Word State Machine to go into READ ADDRESS mode during which the I^2C COM Unit reads the address byte. The STDET output also resets the COM Bit State Machine. Regardless of the current state of the I^2C COM Unit, a start condition resets it to begin receiving an address byte.

The COM Bit State Machine (COMBS) keeps track of the serial bit count. When the COM Word State Machine is in IDLE mode, COMBS also remains idle and does not count. When the COM Word State Machine is out of IDLE mode, the COMBS counts clock cycles by nine. Every ninth clock cycle, the BSACK output signals the acknowledge bit state. The BSACK signal is used to synchronize the I^2C COM Unit. For instance, when COM Word State Machine is in READ ADDRESS mode the COM Buffer shifts serial data in until the BSACK signal is asserted. BSACK indicates that the address byte has completely loaded. The COM Word State Machine will change states at the end of the acknowledge cycle depending on the I^2C address and data direction bit.

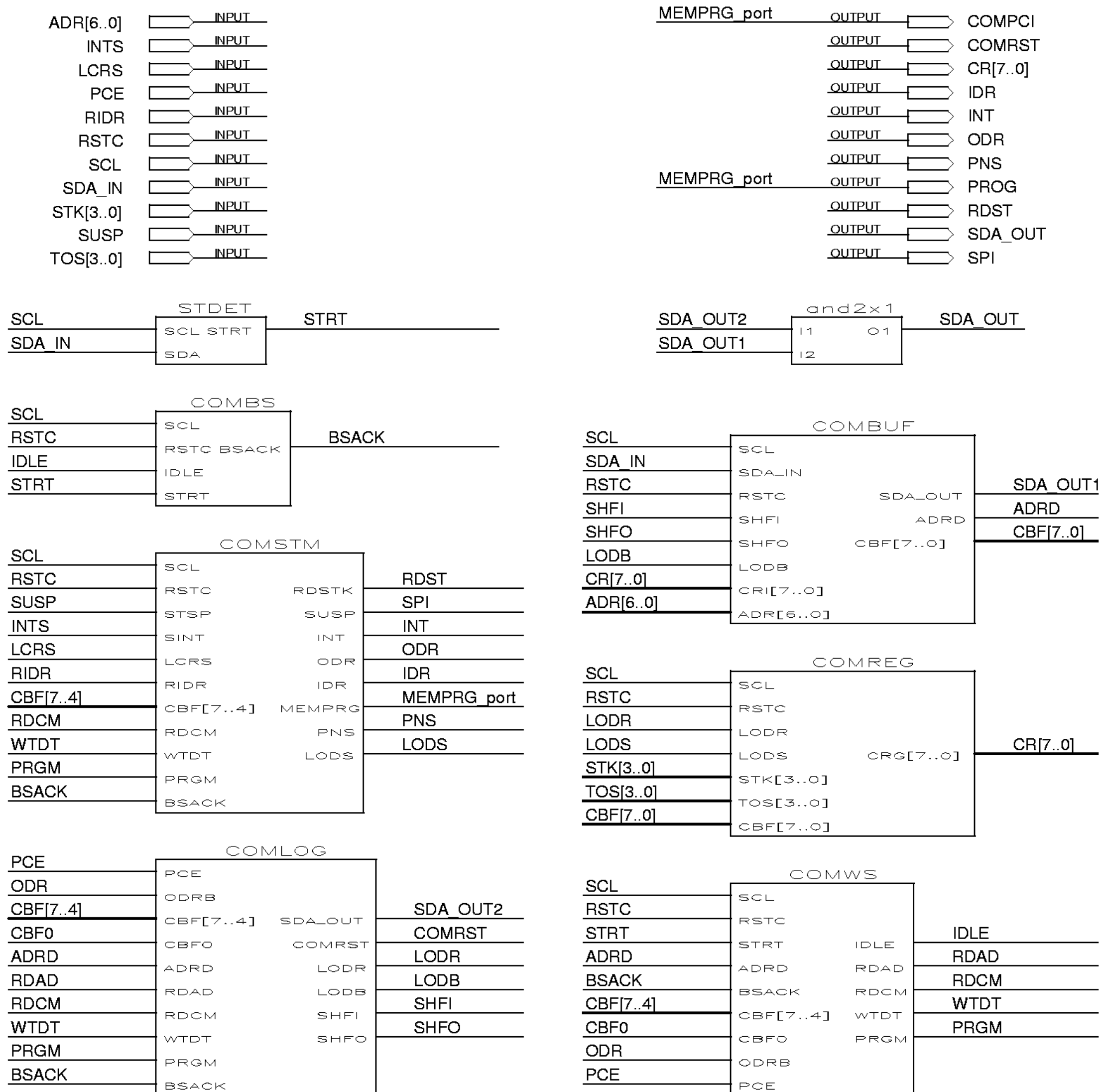


Figure 4-10. I^{2}C Communication Unit Interconnection Diagram

The COM State Machine (COMSTM) contains I^{2}C COM Unit status and control outputs. Status information is stored in SPI, IDR, ODR, and INT bits. SPI indicates that the I^{2}C COM Unit is has suspended the Microdot. The IDR bit indicates that the supervisor processor has loaded a nibble of data to the COM Register with the WRITE NIBBLE command. The ODR bit indicates that the Microdot has loaded new data to the COM Register. The INT bit drives the interrupt output pad that signals the supervisor processor

that the Microdot has data ready for transfer. The control outputs are used for executing I^2C COM Unit operations like reading the stack, programming, or storing data to the COM Register. The RDST output signals the Control Unit that the I^2C COM Unit is reading the stack. The LODS output is also used during a stack read to signal the COM Register to load the input data from the stack (TOS, STK). The MEMPRG_port and PNS outputs are used for programming. The MEMPRG output signals memory write and program counter increment. The PNS (program nibble select) output selects the correct nibble from the COM Register to write to the Program RAM. The I^2C COM Unit receives data in eight-bit blocks. The program memory is four-bits wide. For each program byte received, the I^2C COM Unit first writes the upper nibble (COMREG[7..4]) followed by the lower nibble (COMREG[3..0]).

The COM Logic (COMLOG) sub-block has two functions. Internal control logic controls loading and shifting of the COM Buffer and loading of the COM Register. SDA_OUT2 signals that the SDA line should be driven low during the acknowledge state. The COMRST signal initiates the Control Unit to reset the Microdot. It is asserted during the RESET and PROGRAM commands. LODR signals the COM Register to load from the COM Buffer. LODB signals the COM Buffer to load from the COM Register. SHFI and SHFO outputs control COM Buffer shifting. SHFI signals the COM Buffer to shift data in from the SDA input while SHFO signals the COM Buffer to shift data out to the SDA line.

The COM Buffer (COMBUF) is the serial-to-parallel data storage for the I^2C COM Unit. During read operations, data from SDA_IN input is shifted into bit 0 of the COMBUF. During write operations, data is loaded from the COM Register and then shifted out to SDA_OUT1. The COMBUF is also used to detect the I^2C address. The I^2C address input

(ADR[6..0]) is compared to COMBUF[7..1] and asserts the address detect (ADRD) signal if they match. After the address byte has been received an address detect will put the COM Word State Machine in either READ COMMAND or WRITE DATA mode based on the data direction bit ('0' = READ COMMAND, '1' = WRITE DATA). The data direction bit is placed in bit 0 of the address byte.

The COM Register (COMREG) is the data link from the I^2C COM Unit to the rest of the Microdot. During programming or a WRITE NIBBLE command, data is transferred from the COMBUF into the COMREG where it becomes available for loading to the Stack or Memory Units. During a READ DATA command, the COMREG is first loaded to the COMBUF before being shifted out to the SDA output.

The COM Word State Machine (COMWS) stores the operating mode of the I^2C COM Unit. Figure 4-11 is the state transition diagram for the COMWS. The IDLE mode indicates that nothing is happening. It essentially holds the I^2C COM Unit in a wait state. A start condition puts COMWS into READ ADDRESS (RDAD) mode during which the address byte is serially loaded into the COMBUF. If the Microdot I^2C address is detected, the data direction bit determines whether COMWS goes to READ COMMAND (RDCM) or WRITE DATA (WTDT) mode. If the address is not detected, COMWS returns to IDLE mode. At the end of RDCM mode, a program command puts COMWS into PROGRAM (PRGM) mode. PROGRAM mode loads program bytes and transfers them a nibble at a time to the Program RAM. The COMWS stays in PROGRAM mode until either the supervisor processor terminates programming by issuing a start condition or the Microdot terminates programming when Memory Unit asserts the Program Counter End (signal) indicating that the Program RAM is completely full.

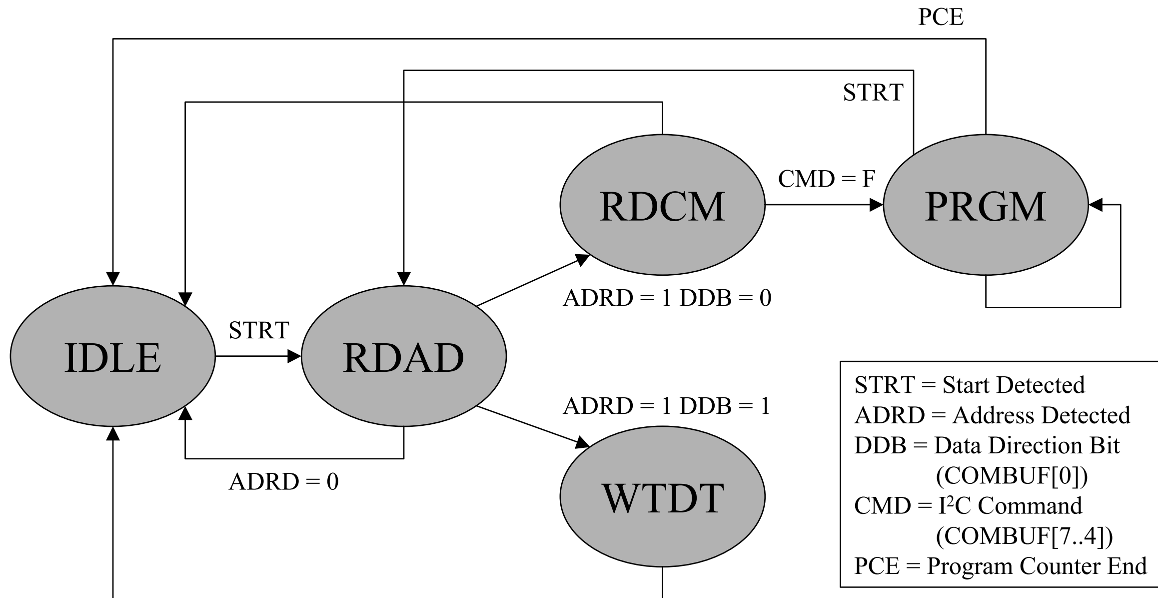


Figure 4-11. COM Word State Machine State Transition Diagram

The SDA AND gate (and2x1) takes in the SDA signals from COMBUF (data) and COMLOG (acknowledge) and sends the output to the SDA pad. If SDA_OUT is low, the bi-directional pad will pull the SDA output to ground signaling either a '0' data bit or I^2C byte acknowledge.

4.9.1 I^2C Commands. The supervisor processor controls the Microdot with commands sent over the I^2C bus. The command set is listed in Table 4-3 below. The RESET, SUSPEND, and RESUME commands allow the supervisor processor to control Microdot execution. The READ STACK command gives the supervisor processor full read access to the Microdot data memory. The WRITE NIBBLE command transfers a nibble from the supervisor processor to the Microdot. The PROGRAM command sets up the Microdot to receive a new program over the I^2C bus. The READ DATA command is a non-standard command that initiates a one-byte data transfer from the Microdot to the supervisor

Table 4-3. Microdot I^2C Commands

Command	Data Direction Bit	Command Byte	Description
READ STACK	0	0SSSSSSS	Load Top-of-Stack Register and Stack RAM element S[6..0] into the COM Register
RESET	0	1000XXXX	Reset the Microdot
SUSPEND	0	1001XXXX	Suspend Microdot execution
RESUME	0	1010XXXX	Resume Microdot execution
WRITE NIBBLE	0	1100DDDD	Write nibble D[3..0] to the COM Register and set the Input Data Ready (IDR) bit
PROGRAM	0	1111XXXX	Put Microdot into PROGRAM mode
READ DATA	1	None	Signal Microdot to write output data to I^2C bus

processor. There is no command byte because the '1' in the address byte data direction bit specifies a slave-to-master data transfer. The READ DATA command is examined further in section 4.9.1.2.

Figure 4-12 illustrates the standard I^2C command cycle. It shows the I^2C bus and COMBUF when a Microdot with address 5A (1011010) is sent the RESET command. After the start condition is detected, the I^2C COM Unit enters the READ ADDRESS mode and serially receives the I^2C address byte that contains the seven-bit address and the data direction bit. If the I^2C COM Unit detects that its I^2C address matches the address byte, it acknowledges its presence by driving the SDA line low during the acknowledge clock cycle and then enters the READ COMMAND mode. If the address does not match, the I^2C COM Unit returns to the IDLE mode. Following the address byte acknowledge, the command byte is sent and acknowledged. At the end of the command byte, the I^2C COM Unit logic detects

which command was sent and executes the command. The READ STACK, READ DATA, and PROGRAM commands are examined further in the sections below.

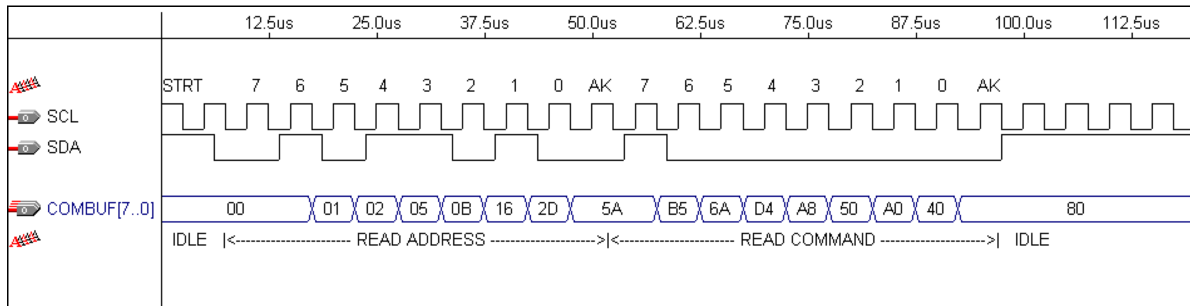


Figure 4-12. I^2C Read Command Cycle

I^2C Command Process (RESET, SUSPEND, RESUME)

1. Detect I^2C Start Condition
2. Read Address Byte
3. If I^2C address detected and data direction bit is '0' then load Command Byte
4. Execute command specified in command nibble (COMBUF[7..4])

4.9.1.1 Reading the Stack (READ STACK command). The READ STACK command was conceived to give the supervisor processor complete visibility into the Microdot data memory. I envisioned that the Microdot could be programmed to store data to specific Stack RAM addresses using the SWAP instruction. The supervisor processor then retrieves the Stack RAM data by sending the READ STACK command to the Microdot. The programmer would have to make sure that the program did not inadvertently overwrite the stack element to be read. This command was also conceived for testing purposes. At any point during execution, the supervisor processor could suspend the Microdot and then check any or all of its stack contents one nibble at a time.

The stack address width is only seven bits so the Stack RAM address of the element to be loaded is included in the command byte. Bit 7 of the command byte specifies a READ

STACK command with a '0'. All other commands have a '1' in Bit 7. After the READ STACK command byte is received, it is transferred to the COMREG. The lower seven bits of the COMREG are sent to the Stack RAM address inputs. The Microdot is temporarily suspended while the addressed Stack RAM element and current TOSREG are loaded into the COMREG. The Stack RAM element is loaded to COMREG[7..4] and the TOSREG is loaded to COMREG[3..0]. The READ STACK command only loads the COMREG and sets the Output Data Ready (ODR) bit. The data is actually sent to the supervisor processor with the READ DATA command as discussed in the next section.

When the I^2C COM Unit receives a READ STACK command, the COMLOG signals the COMREG to load from the COMBUF. At the same time, the COMSTM begins a READ STACK cycle, which is a two clock cycle event. During the first clock cycle, the COMSTM suspends the Control Unit. During the second clock cycle, the RDSTK output signals the Control Unit that, in turn, signals the Stack Unit to address the Stack RAM with the COMREG input. Finally, the LODS output signals the COMREG to load from the Stack Unit and the Output Data Ready (ODR) bit is set. The TOSREG and Stack RAM output are now loaded into the COMREG. Figure 4-13 shows exactly how the READ STACK command works. The address byte selects the Microdot and specifies a command byte is to be read. The command byte is read and recognized as a READ STACK command. During the command byte acknowledge, the COMBUF is transferred to the COMREG. The Stack RAM is addressed with COMREG and loads the Stack RAM element and TOSREG one clock cycle later. The COMREG now holds the data, which will be transmitted to the supervisor processor during the next READ DATA command cycle.

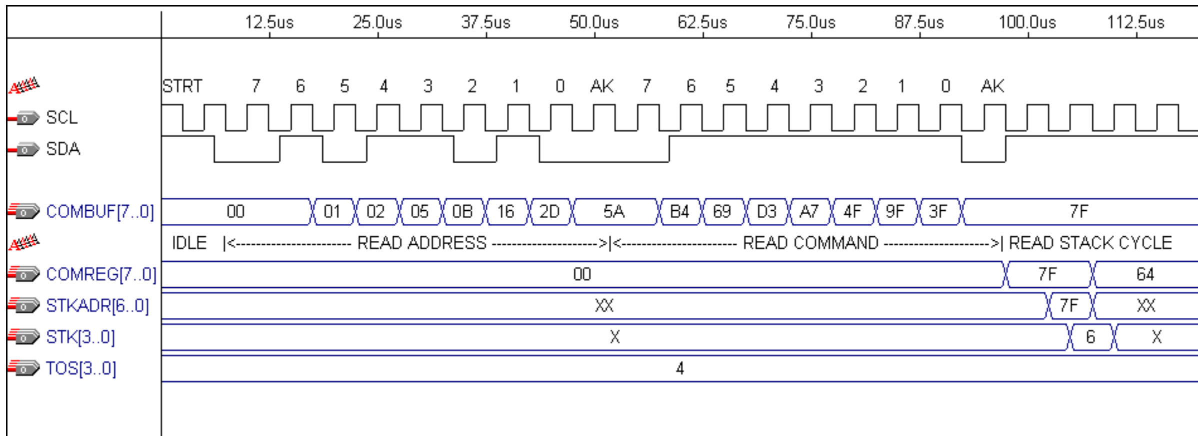


Figure 4-13. I^{2}C READ STACK Command Cycle

Read Stack Process

1. Detect I^{2}C Start Condition
2. Read Address Byte
3. If I^{2}C address detected and data direction bit is '0' then load Command Byte
4. If Bit 7 of Command Byte is '0', load COMBUF to COMREG
5. Suspend Microdot execution
6. Address Stack RAM with COMREG
7. Load TOSREG and Stack RAM to COMREG
8. Set Output Data Ready (ODR = '1')
9. Resume Microdot execution

4.9.1.2 Writing Output Data (READ DATA command). When the I^{2}C COM Unit has output data ready (ODR bit = '1'), the I^{2}C master receives it by sending a READ DATA command. The READ DATA command is simply an address byte with the data direction bit (Bit 0) set to '1'. The READ DATA command puts the I^{2}C COM Unit into WRITE DATA mode. During the address byte acknowledge state, the COMBUF is loaded from the COMREG. The loaded byte is then serially shifted out of the COMBUF to the SDA line. The READ DATA cycle is shown in Figure 4-14. As you can see, bit 0 of the address byte is '1', which signals a Microdot-to-supervisor transfer. During the address byte acknowledge, the byte in the COMREG is loaded into the COMBUF and the I^{2}C COM Unit enters the WRITE DATA mode. This signals the COMBUF to shift left and drive bit 7 to the

SDA line. The Microdot is only capable of writing a single byte at a time. The ODR bit indicates that the COMREG has data to be written. If the ODR bit is not set, the Microdot does not acknowledge the READ DATA command address byte and no data is sent. If the supervisor processor needs to test for correct functionality of the Microdot, it should issue the RESUME command and check for acknowledgement of the address and command bytes.

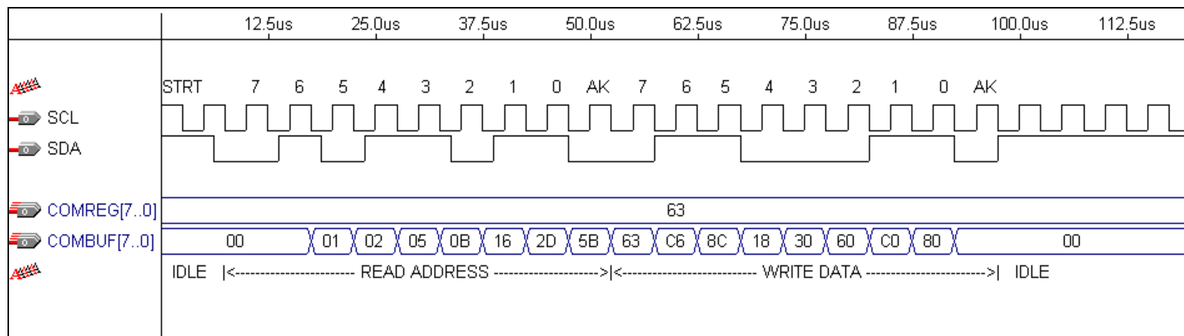


Figure 4-14. READ DATA Command Cycle

Read Data Process

1. Detect I^{2}C Start Condition
2. Read Address Byte
3. If I^{2}C address detected, data direction bit is '1', and output data is ready (ODR = '1') then load COMBUF from COMREG
4. Shift COMBUF data to SDA output (MSB first)

The ODR (output data ready) and IDR (input data ready) bits are the handshaking scheme for synchronizing data transfer to and from the Microdot. IDR indicates that input data is ready while ODR indicates that output data is ready. When the Microdot is expecting data from the supervisor processor, it checks the status of the IDR bit by executing the Set Status Register (STSR) instruction. When the supervisor processor is expecting data from a Microdot, it simply needs to periodically send a READ DATA command. If the ODR bit is set, the Microdot will acknowledge the READ DATA command address byte and then write

the data byte to the I^2C bus as shown in Figure 4-14. The ODR bit is set when the Microdot executes a Store to COM Unit (STCM) instruction. This instruction loads the COMREG with the TOSREG and Stack RAM output. The ODR bit is also set during the READ STACK command cycle. The Microdot loads the status of the ODR bit with the STSR instruction. Thus, the Microdot can wait until the I^2C COM Unit has written the data to the supervisor processor before writing new data to the COMREG. The STSR, BRCH, and JMP instructions let the programmer create programs that synchronize data transfer with the I^2C COM Unit. The code listing below in Figure 4-15 illustrates a program loop that checks for input data. The STSR instruction loads the status register with the I^2C COM Unit status bits (INTX, ODR, IDR). The BRCH instruction checks the IDR bit using mask value 1_{HEX}. If input data is ready, the BRCH instruction exits the loop by adding four to the program counter, which takes it past the JMP instruction. Otherwise, the JMP instruction sets the program counter back to address 039 to begin the check again.

```
ADDR  DATA  INSTRUCTION
039   3     STSR
03A   8
03B   D     BRCH IDR +4
03C   1
03D   0
03E   4
03F   E     JMP 039
040   0
041   3
042   9
```

Figure 4-15. Programming Example to Wait for Input Data

4.9.1.3 Programming (PROGRAM command). The supervisor processor starts the programming process by sending the PROGRAM command. During command byte acknowledge, the I^2C COM Unit resets the Microdot and then suspends it. The reset

returns the program counter to 000_{HEX}. Following the PROGRAM command, the supervisor processor sends the first program byte of the new program. When the byte is received, it is acknowledged and then transferred to the COMREG. The COMSTM then begins the program byte write cycle. This is a three clock cycle event. During the first clock cycle, the even program nibble (COMREG[7..4]) is written to program memory and the program counter is incremented at the end of the cycle. The next clock cycle is an idle state inserted to eliminate timing issues caused by writing to the program memory while the program counter is incremented. The third clock cycle contains the odd program nibble (COMREG[3..0]) write followed by the program counter increment. Programming continues until the memory is full or the supervisor processor ends programming by asserting a start condition on the I^2C bus. When the memory is full, the Program Counter End (PCE) signal will be asserted by the Memory Unit. When PCE is asserted, the I^2C COM Unit withholds the byte acknowledge for the last program byte and returns to the IDLE mode. After programming has ended, the Microdot remains suspended until the supervisor processor issues a RESET command, which returns the program counter to 000_{HEX} and starts execution of the new program.

Figures 4-16 and 4-17 show the program process. Programming starts like any other I^2C command. The start condition is followed by the address byte. Following address byte acknowledge, the PROGRAM command (11110000) is sent. The Microdot acknowledges the command and enters PROGRAM mode. Figure 4-17 shows the programming process between the I^2C COM Unit and the program memory. Each program byte is received, acknowledged, and then transferred to the COMREG where control signals from the I^2C COM Unit load the Program RAM a nibble at a time and increment the program counter.

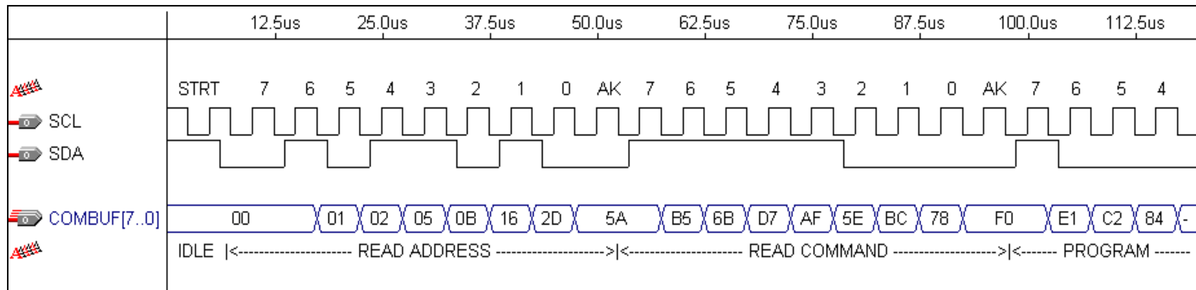


Figure 4-16. I²C Program Cycle. Address byte is followed by the PROGRAM command, which puts the I²C COM Unit into PROGRAM mode.

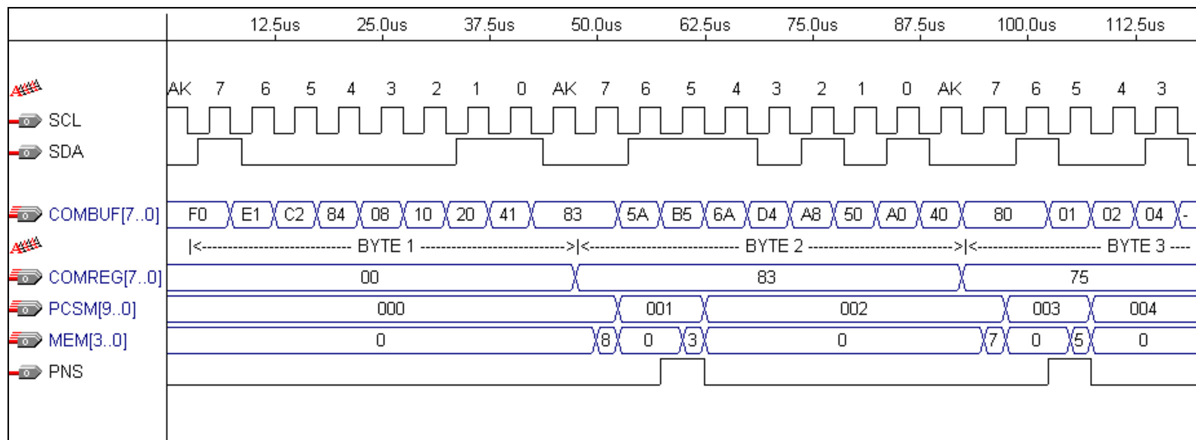


Figure 4-17. Continuation of Program Cycle. After each program byte is received, the byte is transferred to the COMREG and then written to program memory a nibble at a time.

Program Process

1. Detect I²C Start Condition
2. Read Address Byte
3. If I²C address detected and data direction bit is ‘0’ then load Command Byte
4. If command nibble (COMBUF[7..4]) is ‘1111’, then enter PROGRAM mode
5. Suspend the Microdot
6. Reset the Program Counter
7. Load a Program Byte
8. Transfer Program Byte from COMBUF to COMREG
9. Write program nibble 1 (COMREG[7..4]) to Program RAM
10. Increment Program Counter
11. Write program nibble 2 (COMREG[3..0]) to Program RAM
12. Increment Program Counter
13. Return to Step 7 (Load a Program Byte) until I²C Start Condition detected or PCE signals the Program RAM is full

4.9.2 Interrupt. The interrupt is an open-drain bi-directional port for synchronization between the Microdot and the supervisor processor. The interrupt was conceived to give the Microdot the capability to interrupt the supervisor processor if it has data ready to send. Imagine a Microdot is interfaced with a temperature sensor that is monitoring a critical device. If the Microdot reads a temperature above a certain threshold, it should alert the supervisor processor so corrective action can be taken. When the temperature is exceeded, the Microdot asserts the interrupt. The supervisor processor services the interrupt by reading data from the Microdot which indicates an overheat condition on the critical device and the supervisor processor reacts accordingly. A Microdot can signal the supervisor processor by executing the interrupt instruction (INT), which pulls the interrupt line low. The interrupt line is connected to each Microdot and the supervisor processor through a pull-up resistor. This creates a wired-or configuration for the Microdots. There is no address information to the interrupt so when the interrupt line is pulled low, the supervisor processor must poll each Microdot until it finds the Microdot(s) that are interrupting. The supervisor processor polls the Microdots by sending a READ DATA command to each Microdot until the interrupt line returns high. If a Microdot has output data ready, the Microdot will acknowledge the address byte and transmit its output byte. When the output byte is sent, the Microdot will reset its interrupt output. The Microdot cannot set its interrupt unless it has output data ready. Consequently, in order for the Microdot to pull the interrupt line low, the program must have a STCM instruction prior to the INT instruction so the ODR bit will be set when the INT instruction is executed. If there is not output data ready, the INT instruction has no effect.

4.10 Microdot Top Level

Figure 4-19 shows the top-level interconnection diagram for the Microdot. The interconnection signals are defined in the signal table in Appendix A. This figure shows the integration of the Microdot functional units without the stack and program SRAM units. The interconnection diagram for the complete integrated version of the Microdot that contains the SRAM units and pads is shown in Appendix C. Figure 4-18 is the interface diagram for the test version of the Microdot. The test version has 42 pins. The EXADR, EXDAT, and EXMWE ports are the external memory interface. The DIM (Disable Internal Memory) input selects the external memory as the instruction memory and disables the internal Program RAM. The MADR port is the input for the seven-bit I^2C address. The SCL and SDA signals are the interface to the I^2C serial bus. The interrupt (INT) output is used for interrupting the supervisor processor. The SSEL inputs are the select lines for the Status Multiplexer as defined in section 3.11. The SMX outputs are the Status Multiplexer output. The eight bidirectional data ports are defined as EP[7..0]. RST is simply the master reset signal.

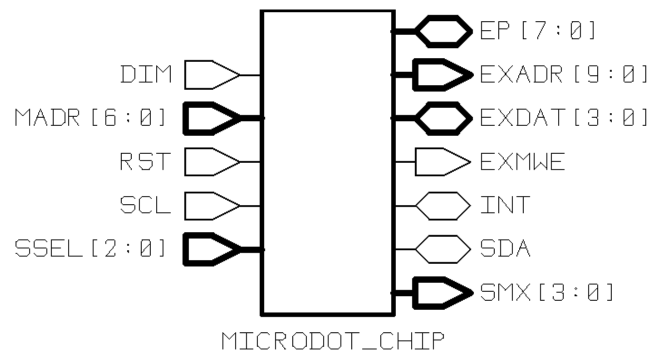
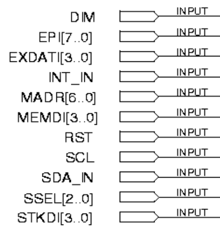
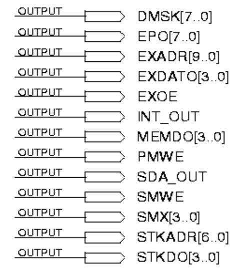
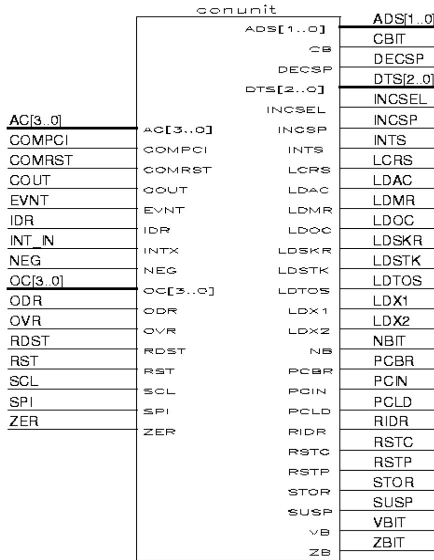
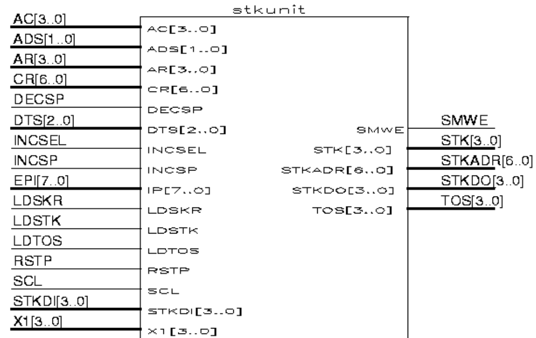


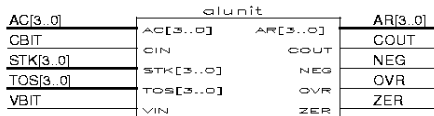
Figure 4-18. Microdot Interface Diagram

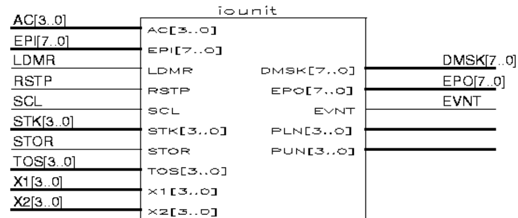


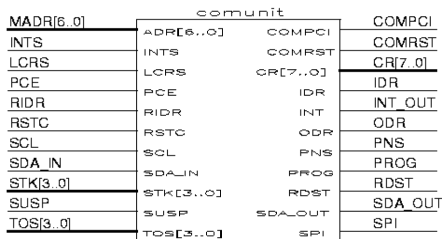


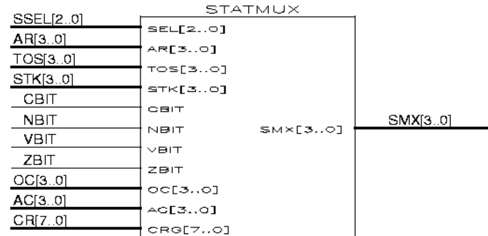












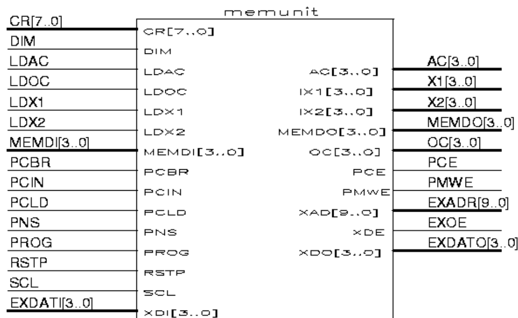


Figure 4-19. Microdot Interconnection Diagram

4.11 Microdot Production Design

The production version of the Microdot would only have 14 pins or pads. Figure 4-20 shows the pinout for a production version.

1. Serial/System Clock (SCL)
2. Serial Data (SDA)
3. Reset
4. I/O Port 0
5. I/O Port 1
6. I/O Port 2
7. Ground
8. I/O Port 3
9. I/O Port 4
10. I/O Port 5
11. I/O Port 6
12. I/O Port 7
13. Interrupt
14. Vcc

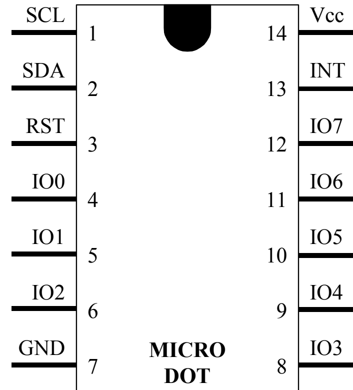


Figure 4-20. Microdot Production Version Pinout

The I^2C address of each Microdot would be "burned" in after fabrication. The address bit structure is a combination pull-up/pull-down structure in top layer metal that feeds the seven-bit address input. The address is defined by laser-cutting each address bit structure to make it a pull-up or pull-down and thus define the address bit as a '1' or '0'.

4.12 Summary

In this chapter, I have provided a look inside the Microdot. I have examined each functional block and its sub-blocks, covering the overall functionality and interconnections. I have also gone in depth into the design and operation of the I^2C serial bus interface.

5. Testing and Analysis

5.1 Introduction

In this chapter, I review the testing and analysis of the Microdot design. Testing was completed at each level of design starting with the behavioral models for the Microdot and its functional blocks and sub-blocks. The fabrication costs and limitations to the RHBD gate array library prevented fabrication of the Microdot. In lieu of fabrication, I constructed and tested a prototype using an Altera field programmable gate array (FPGA). Consequently, most of the simulation and testing was completed with the Altera FPGA software and hardware.

5.2 Altera Overview

The Altera Max+ Plus II development software is a complete design flow tool for developing logic circuits on Altera FPGAs. It supports both schematic capture and hardware description languages including Verilog and VHDL. It also offers complete simulation and timing analysis. Behavioral and structural designs implemented in Synopsys can be ported directly to the Altera FPGA. A gate library was created for the Altera FPGA to match the RHBD gate array library. The Microdot design for the RHBD gate array library was created with Synopsys Design Analyzer and then ported to the Altera FPGA for testing.

The Altera device I used was the EPF10K20RC240-4. It is part of the Altera University Program UP-1 design laboratory package shown in Figure 5-1. The FLEX 10K20 is a SRAM-based FPGA that has 20,000 gates and 12,288 RAM bits. Considering that Microdot uses less than 1,000 gates and 4.5K RAM bits, it was more than enough for me to fully implement and test the complete design. The extra capacity of the

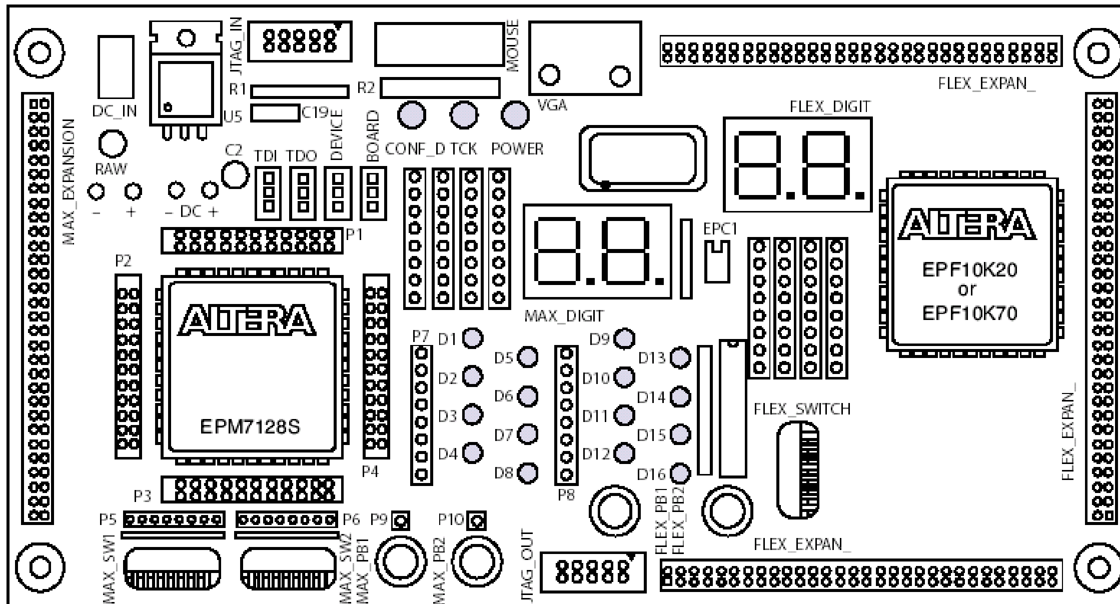


Figure 5-1. Altera UP-1 Design Laboratory Package [21]

FLEX 10K20 also allowed test units to be implemented on the same chip. Thus, the entire Microdot design, including a simulated supervisor processor interface, could be tested with only a single chip.

Another advantage to using the FLEX 10K20 was the embedded array blocks (EAB). The EABs are blocks of RAM that can be configured to create complete RAM units of varying size and depth. The FLEX 10K20 has six EAB with each containing 2048 bits. The Max+ Plus II software has EAB megafunctions that allow easy creation of complete Random Access Memory designs of variable width and depth. Both the stack memory and program memory were created with EAB megafunctions. Another advantage of the EABs is that the memory can be pre-loaded using a memory initialization file. This allowed for easy testing of the chip without having to use external memory or external circuits for downloading programs.

The 10K20 is an array of 1152 Logic Elements. The FLEX 10K Logic Element is shown in Figure 5-2. The logic actually consists of the programmable four-input lookup table. The element is configured for combinational or sequential operation by selecting the Register Bypass or Programmable Register. The Carry and Cascade structures are specifically designed to create fast adders and counters. Programming the 10K20 is done by downloading a configuration bit stream to the device. The bit stream programs the lookup table and sets the configuration of each Logic Element. Fortunately, the Max+ Plus II software simplifies the design process by synthesizing the behavioral or structural logic into the configuration for each Logic Element. The software also configures the programmable interconnects to connect the Logic Elements. A SRAM-based FPGA loses its configuration whenever power is cutoff. In permanent designs, the configuration must be stored in a programmable read-only memory (PROM) connected to the FPGA. Every time the power is cycled, the FPGA loads its configuration bit stream from the PROM.

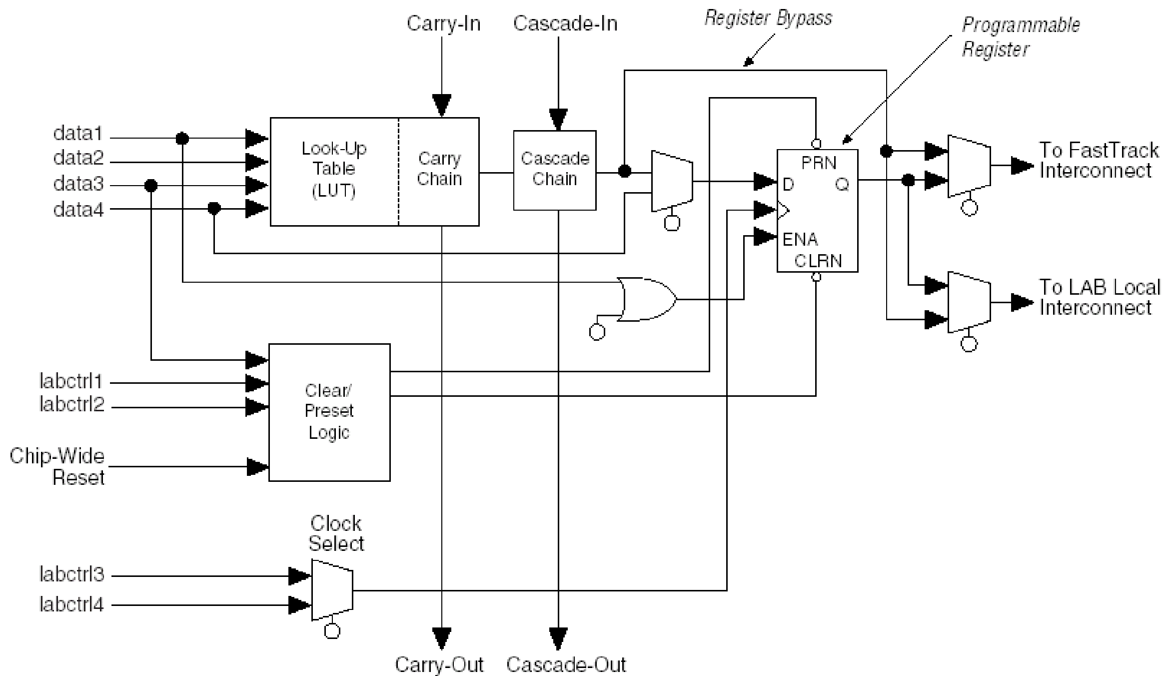


Figure 5-2. Altera FLEX10K Logic Element [22]

5.3 Behavioral VHDL Testing

The design entry started with behavioral VHDL. The behavioral VHDL defines only the function of each block with standard programming constructs. At this point, no structural design information is included. Each sub-block was implemented in behavioral VHDL and then tested for correct functionality with the Synopsys VHDL Simulator. The next level of design hierarchy is the functional block level. Each functional unit was a structural VHDL implementation of its sub-blocks. Each functional block was then tested for correct functionality. After verifying correct operation of each functional block, the complete design was integrated with a top-level structural VHDL file and then tested.

After successfully testing the behavioral design with Synopsys VHDL simulator, the design and testing moved to Altera Max+ Plus II design software. I found the Max+ Plus II software much more useful for creating simulation files. Unlike Synopsys, simulation files could be created with a graphical timing diagram. This made the simulation process much more intuitive. The behavioral VHDL was once again tested. This time, however, the testing included the FPGA timing. The Max+ Plus II software synthesizes the behavioral VHDL into a structural layout for the selected FPGA chip. It then analyzes the structural layout using the known logic block timing delays to compute the overall timing of each sub-block. Behavioral testing with timing verifies functionality with realistic timing constraints. It was during this testing that several timing errors were uncovered that required some redesign. Specifically, timing errors with read and write operations to the two memories (program and stack) were causing incorrect data to be loaded from and stored to the asynchronous RAMs. The errors stemmed from the fact that all control signals and registers were synchronized with the falling edge of the clock. These errors were corrected by using

synchronization buffer registers that loaded on the rising edge of the clock so data was held stable at the falling edge of the clock.

5.4 Structural VHDL Testing

After verifying the correct operation of the Microdot behavioral design, the next step was to convert the behavioral VHDL to logic gates. For large and complex designs, a logic synthesis tool is required to translate behavioral VHDL into its logic gate equivalent. Synopsys Design Analyzer was the logic synthesizing tool used for converting the Microdot behavioral VHDL to the RHBD gate array implementation. Design Analyzer uses size and timing information in the RHBD gate array design library to synthesize logic for selectable size and performance constraints. For the Microdot I synthesized logic for minimum area, as performance was not the first concern. The structural design was ported to the Altera design software. In order to port the structural VHDL, I first had to port the RHBD gate array library to the Max+ Plus II software. The structural VHDL for each of the functional units was tested with the simulation files created for the behavioral VHDL. After the top-level design had been successfully simulated, the design was downloaded to the UP-1 board for final design verification.

5.5 Full FPGA Testing

The final design verification step was loading the design on actual FPGA hardware and running a test program to confirm correct functionality. The Microdot design was downloaded to an Altera EPF10K20RC240-4 on the UP-1 Design Laboratory Package. Test monitoring was accomplished with a Hewlett-Packard 1630D Logic Analyzer. The HP1630D can record 1024 state measurements on 43 channels at up to 25 MHz. It is also

capable of timing measurements up to 100 MHz. The 25.175 MHz oscillator on the UP-1 board was stepped down to 3.15 MHz for full testing. A test program was written to test the instruction set. Figure 5-3 shows the FPGA test configuration.

(a) (b)

Figure 5-3. Microdot FPGA Test Setup. Included (a) HP1630D Logic Analyzer and (b) Altera UP-1 design board

5.6 I²C Driver

The processor portion of the Microdot was quite simple to test. Test programs were downloaded to program memory as part of the FPGA configuration and the logic analyzer could monitor program execution. However, testing the I²C interface required additional circuitry because the I²C bus is controlled by an external supervisor processor. There was no supervisor processor available for testing so a test circuit was created to act as the I²C bus master (supervisor processor). Because of the extra logic capacity of the 10K20 (Microdot uses only 38% of logic resources), the test circuit could easily be included as part of the overall FPGA design. The I²C Driver circuit tested the entire I²C command set including downloading a new program.

5.7 FPGA Results

The FPGA implementation of the Microdot required 435 out of the 1152 (37.7%) Logic Elements. The program and data memory requires 3 of the 6 (50%) Embedded Array Blocks. The timing and power consumption are presented in the next two sections.

5.7.1 FPGA Timing. The overall performance of both the FPGA design and RHBD gate array design is limited by the speed of the program memory. All timing delays in this discussion are maximum simulated timing delays for the EPF10K20RC240-4. The PCSM, which is loaded on the falling edge of the clock, has a timing delay of 19.9 ns. The PCSM addresses the Program RAM, which has timing delay of 35.1 ns. The memory output is loaded to the buffer registers on the rising edge of the clock. Therefore, in order to guarantee that the memory loads correctly, there needs to be delay of 55 ns between the falling edge and rising edge of the clock. For a 50% duty cycle, the minimum clock period would be 110 ns (9.1 MHz). All other logic blocks meet timing requirements within a 110 ns clock cycle. For instance, the critical path in the Control Unit is the LDSKR output which controls loading of the stack buffer registers on the rising edge of the clock. The maximum delay from negative clock edge to LDSKR would be 40.9 ns. A 55 ns half-clock period provides a 14 ns margin. All other control signals have the full 110 ns to propagate through before the next falling clock edge. The maximum delay of the Control Unit is 53 ns. The timing results of the FPGA design are presented in Table 5-1. Test program simulation showed that the design functioned at 9.1 MHz without timing errors. Hardware tests actually showed that the Microdot could be clocked at up to 12.6 MHz before timing errors were detected. These tests were not exhaustive, though, so this number cannot be guaranteed.

Table 5-1. FPGA Timing Results

Min Clock Period	110 ns
Max Frequency	9.1 MHz
Functional Unit	**Max Delay (ns)**
ALUNIT	46.4
MEMUNIT	49.5
STKUNIT	26.9
IOUNIT	29.2
CONUNIT	53
Program RAM	35.1
Stack RAM	30.1

5.7.2 FPGA Power Consumption. For the FPGA, power consumption is estimated based on the size of the design and the operating frequency. The formula is [22]:

$$I_{CCACTIVE} = \mathrm{K} \times f_{MAX} \times \mathrm{N} \times tog_{LC} \quad \frac{\mu A}{MHz}$$

Where K is a multiplication constant for each device. For the EPF10K20, K is 89. The maximum frequency is given as f_{MAX}. The maximum frequency for the Microdot is 9.1 MHz. N is the number of logic elements in the design. The Microdot design required 435 logic elements. The tog_{LC} variable is the estimate of the average number of logic elements toggling each clock cycle. The data sheet provides a typical number at 12.5%. An analysis of the Microdot shows that most registers change state once every 4-5 clock cycles. Thus, the maximum average toggle rate should never exceed 25%. The current consumption at max operating frequency (9.1 MHz) for the Microdot is 88 mA, which equates to a power dissipation of 440 mW. This is orders of magnitude above the typical operating currents of the four-bit microcontrollers presented in Table 2-1. Maximum current is not really a fair comparison. If we look at the current consumption at slower speeds, we see the design compares more favorably with the typical four-bit microcontrollers.

Table 5-2 shows the supply current, power dissipation, and performance over the range of operating frequencies. At the target operating frequency of 32 kHz, we see that the supply current is 310 μA with a power dissipation of 1.55 mW. This is still quite higher than the extremely low-power microcontrollers presented in Table 2-1, but is still considered low-power. The numbers presented so far are for the 5-Volt EPF10K20 device on the UP-1 board. If I wanted to minimize power consumption in the FPGA implementation, I would select the 3.3-Volt EPF10K10A. This device consumes 1/5 the current of the EPF10K20. This would reduce 32kHz current and power consumption to 59 μA and 196 μW, respectively. These numbers are on par with the low-power four-bit microcontrollers currently available.

Table 5-2. Microdot FPGA Supply Current, Power Dissipation, and Performance (Altera EPF10K20RC240-4)

Frequency	Supply Current (mA)	Power Dissipation (mW)	Performance x1K instr/sec
32 kHz	0.3104	1.552	8.9
100 kHz	0.97	4.85	27.8
500 kHz	4.85	24.25	139
1.0 MHz	9.7	48.5	277
2.0 MHz	19.4	97	556
3.0 MHz	29.1	145.5	833
4.0 MHz	38.8	194	1111
7.0 MHz	67.9	339.5	1944
9.0 MHz	87.3	436.5	2500

The decision against using Altera SRAM-based FPGAs for distributed and reconfigurable low-end computing in space-borne applications is their susceptibility to SEU. If SRAM-based FPGAs were radiation-hardened against total ionizing dose and SEU, they

would offer far more flexibility for reconfigurable control and data collection on microdevices than would small microcontrollers. The designer would not be limited to eight I/O ports. Nearly every pad/pin could be configured as an input or output. Also, the internal logic of the FPGA could be more efficiently tailored to the microdevice that it interfaces.

5.8 RHBD Gate Array Results

The Microdot was designed with a RHBD gate array standard cell library. The behavioral design of each functional block was synthesized and/or hand-designed into a structural design. The structural designs were then used to create the gate-array layouts with the GARDS place-and-route tool. I did this at Mission Research Corporation, Microelectronics Division in Albuquerque, NM.

Table 5-3 RHBD Microdot Design Summary

Components	782	Area
Gates	666	33 %
Flip-flops	116	67 %
Gate Array Cells	3684	0.88 mm^2
Functional Unit	**Width (µm)**	**Height (µm)**
ALUNIT	180	385
COMUNIT	580	580
CONUNIT	385	390
IOUNIT	375	385
MEMUNIT	575	580
STKUNIT	570	435
STATMUX	180	140
MICRODOT CORE	1440	1280

Table 5-3 is a summary of the RHBD gate array design. You can see that the SEU-hardened flip-flops require 67% of the cell area, but are less than 15% of the total components. The dimensions of each functional block and the Microdot core are also presented in Table 5-3. This does not include the RAM blocks for the stack or program memory. Memory size is discussed in section 5.8.2.

5.8.1 RHBD Gate Array Timing. The Synopsys Design Library for the RHBD gate array contains elements for timing analysis. Overall timing delays are computed from individual gate delays and the fanout of each gate. While this level of timing analysis is somewhat coarse, it does provide good estimates of what the actual timing delays would be in a fabricated design. I expected the RHBD gate array design to be much faster than the FPGA design and it is.

Table 5-4. RHBD Gate Array Timing Results

Min Clock Period	43 ns
Max Frequency	23.3 MHz
Functional Unit	**Max Delay (ns)**
ALUNIT	9.2
MEMUNIT	8.6
STKUNIT	4.4
IOUNIT	1.1
CONUNIT	2.6
I^2C COMUNIT	4.3
Program RAM	20.7
Stack RAM	20.7

Table 5-4 shows the timing performance for the RHBD gate array version of the Microdot. The limiting factor for performance is still the memory. A 128x4 memory

structure created with RHBD gate array components had a simulated timing delay of 20.7 ns. Since the PCSM has delay of 0.8 ns, the total delay for the memory output is 21.5 ns. Since 21.5 ns is the minimum half-clock cycle, the total clock cycle time would be 43 ns which is a frequency of 23.3 MHz. All the other functional units operate well within the 43 ns clock cycle.

5.8.2 RHBD Gate Array Memory Design. One of the limitations to implementing the Microdot with the RHBD gate array is there is no compact memory structure for implementing SRAM units. The only way to create memory structures is to use existing latches and flip-flops that are not area efficient. A 128x4 RAM implemented with available RHBD gate-array components would require a cell area of 5137 (1.23 mm^2) with non-SEU-hardened latches and 6673 (1.60 mm^2) with SEU-hardened latches. The units of cell area are gate array cells, which are 240 μm^2. The microdot without SRAM units requires a cell area of 3684 (0.88 mm^2). These are minimum areas that do not factor in extra area needed for interconnect routing which could be as much as 125%. The two RAM units (Program and Stack) dominate the total cell area and ultimately will control the size of the Microdot. Considering that a full program memory will be nearly eight times as large as the 128x4 SRAM (9.84 mm^2 for non-SEU-hard, 12.8 mm^2 for SEU-hard), creating large memories with the gate array library wastes too much space and does not make sense for the Microdot design. The smaller the size of the SRAM cell, the smaller the Microdot will be. It would be advantageous to develop smaller RHBD SRAM cells that could be efficiently packed to create smaller SRAM designs that still have an acceptable level of total ionizing dose and SEU hardness.

5.9 Summary

The Microdot was successfully tested with an Altera FLEX 10K20 FPGA. Simulations and testing showed that the FPGA Microdot could be clocked at speeds up to 9.1 MHz. The current and power consumption at the base frequency of 32 kHz is low, but still does not come close to the low-power four-bit microcontrollers currently available. By using improved versions of the FLEX 10K20 FPGAs or fabricating the RHBD gate array layout the Microdot can achieve better performance and lower power consumption. A partial layout was created with a RHBD gate array library. Simulations on this design showed a performance of 23.3 MHz. The partial layout without RAM units requires 5.8 mm^2. A full layout in RHBD gate array requires a minimum area of 15.28 mm^2. Unfortunately, this does not meet the design goal for small area.

6. Conclusions and Future Work

6.1 Summary and Accomplishments

I designed, implemented, and tested the Microdot, a four-bit microcontroller designed for distributed microdevice control and data collection in satellites. Normally, a single processor interfaces a large number of microdevices and must execute individual subroutines to service each device. As the number of devices in a design increases, the processing capability needs to be increased. The concept of the Microdot is to distribute the needed processing capability among a network of small microcontrollers dedicated to each microdevice instead of using a single processor for all processing. Delegating processing serves to reduce the level of processing required by a single processor, which can also reduce the power consumption and increase reliability. Another element of the Microdot concept is reconfigurability. The Microdot makes satellite design flexible because each Microdot can be reprogrammed on-the-fly.

In this thesis, I expanded previous research completed at both the Air Force Research Laboratory Space Electronics Branch (AFRL/VSSE) and the Air Force Institute of Technology. The major design improvement I gained with this version of the Microdot was the addition of the I^2C serial bus interface for distributed computing. The previous versions of the Microdot introduced the distributed computing concept, but did not really provide an efficient way to create a distributed network. The I^2C bus uses only two signal lines to connect up to 128 Microdots to a single central processor. My work with the Microdot design transformed it from just a small microcontroller to a more flexible microcontroller designed specifically to operate as a distributed computing element within an overall distributed computing system.

I designed the Microdot microcontroller for fabrication in a radiation-hardened by design (RHBD) gate array standard cell library for the Taiwan Semiconductor (TSMC) 0.35 micrometer CMOS process. The RHBD gate array design of the Microdot core requires 666 combinational gates and 116 flip-flops with an area of 1.84 mm^2. Simulation using 128x4 RAM units for both the program and stack memory showed a maximum clock speed of 23.3 MHz. The test version of the Microdot requires 44 pins. The production version of the Microdot requires 14 pins.

I implemented the full Microdot design in an Altera FLEX 10K20 FPGA. During my simulation and hardware testing, I showed that the maximum clock speed is 9.1 MHz, which equates to 2.5 million instructions per second (MIPS). At this frequency, the circuit draws 87.3 mA and dissipates 436.5 mW. At the low-power operating frequency of 32 kHz, the Microdot executes 8.9 thousand instructions per second (KIPS) while consuming 310 μA and dissipating 1.552 mW of power.

6.2 Conclusions and Lessons Learned

Designing for a radiation-hardened library requires an initial understanding of the layout and performance of the individual cells. For instance, it is important to understand the area penalty of using SEU-hardened flip-flops. As noted earlier, state machine design was one area where I learned that using a one-hot design strategy may simplify logic, but it increases area and makes a design more susceptible to SEU. In RHBD designs, the strategy must be to reduce the number of flip-flops. Ultimately, to increase reliability in SEU environments, redundancy and error detection and correction (EDAC) must be applied to memory cells so reducing the number of flip-flops reduces the area penalty of designing in SEU hardness.

It is important to obtain the design libraries early in the process. After analyzing the RHBD design, I discovered the areas where the design could be improved. The major shortcoming of the RHBD gate array library was the lack of area-efficient SRAM cells. Therefore, the RHBD gate array library should not be used if a full version of the Microdot is fabricated. Since memory ultimately determines the size of the design it would be better to design the Microdot for a more area efficient standard cell library with smaller SRAM cells.

Timing is one of the hardest things to get right and has to be correctly engineered from the earliest stages of design and properly monitored for improvement. The clocking strategy used for the Microdot was single phase clocking. The advantage of single phase clocking is that only a single clock signal has to be routed. However, clock skew between different areas on the chip can cause timing errors that have to be corrected. In my case, it cost me 12 extra flip-flops, which was a significant area penalty. Two phase clocking can eliminate clock skew errors and probably should have been considered as the clocking strategy.

Another lesson learned was to get your design tools in place as early as possible. At the start of my research, AFIT did not have any FPGA design tools for VLSI research. Because of the uncertainty of the fabrication process due to cost and timeline, I wanted a way to prototype and test my designs with more than just simulation tools. This is why I got AFIT started with the Altera University Program. The design and simulation process improved greatly after I got the Altera FPGA tools. I was able to create simulation files and run the simulations very quickly. Also, the reconfigurable SRAM-based chip permitted me to quickly download a design to the chip and run a hardware test. If I had been able to use the Altera software and hardware from the start of my research, it would have sped up the

design evolution and I could have recognized design problems earlier and got a better working design in the end. The Altera FPGA tools are very useful to VLSI design and looking back, I am glad that I went to the trouble to get them.

6.3 Future Research

The Microdot is essentially a small reconfigurable computing element. My work with the Altera FPGAs showed me that SRAM-based FPGAs are the ultimate in reconfigurability. I believe future research in distributed low-end computing for space applications should look at how to develop small, low-power SRAM-based FPGAs for robust operation in the space radiation environment. Conventional SRAM-based FPGAs are not suitable for operation in a space environment mainly due to their SEU susceptibility. Since the configuration is stored in SRAM cells, an upset of a configuration bit could alter the functionality of the device and cause failure. Radiation-hardened SRAM-based FPGAs would offer designers a much greater amount of flexibility for interfacing microdevices. They would not be limited to eight I/O pins on a single device, nor would they be limited to the capabilities of the Microdot instruction set. Each FPGA could be more uniquely tailored to the device it interfaces which would be much more efficient. If a device required a greater amount of processing capability than a single FPGA could provide, multiple FPGAs could be used. I do believe an I^2C serial bus would still be very useful for creating a distributed architecture. It would be important to have a standard interface for downloading configurations as well as providing a data path back to the supervisor processor.

Appendix A. Microdot Functional Blocks and Signals

Table A-1. Microdot Design Hierarchy

Abbreviation	Name	Function
MICRODOT_CHIP	Microdot Chip Design	Top-level design that contains microdot processor, RAM units, and external pads
MICRODOT	Microdot Processor Design	Processor design contains the functional blocks without the RAM units or external pads
CONUNIT	**CONTROL UNIT**	Control logic for microdot functional units
CONTSTM	Control Unit State Machine	Control Unit state plus Status Register
CONCNT	Internal Control Logic	Control Unit State Machine Logic
MEMCNT	Memory Control Logic	Memory Unit Control Signals
STKCNT	Stack Control Logic	Stack Unit Control Signals
IOCNT	I/O and I^2C COM Control Logic	I/O and I^2C COM Unit Control Signals
MEMUNIT	**PROGRAM MEMORY UNIT**	Microdot Program Memory
PCSM	Program Counter	Holds current program memory address
PCLOG	Program Counter Logic	Either increments program counter or adds instruction register increment during BRCH instruction or loads from instruction register during JUMP instruction
INSREG	Instruction Register	Holds all elements of the current instruction (OPCODE, ALU CODE, INDEX 1, and INDEX 2)
XMEMMUX	External Memory Mux	Selects program memory input based on the state of the disable-internal-memory input
RAM1024X4	Program Memory	Program SRAM
STKUNIT	**STACK UNIT**	Microdot Data Memory
TOSREG	Top of Stack Register	Register for the Top of Stack
TOSMUX	Top of Stack Data Multiplexer	Selects the data input for Top of Stack based on the current instruction
STKADRMUX	Stack Address Multiplexer	Selects stack RAM address from stack pointer, stack adder, or instruction register based on the current instruction

STKPTR	Stack Pointer	Holds the current stack RAM address -- Adding elements to the stack decrements the stack pointer
STKPTRLOG	Stack Pointer Logic	Increments or decrements the stack pointer based on the control signals
STKADD	Stack Address Adder	Adds the stack offset (ALU Code or Index 1) to the current stack pointer
RAM128X4	Stack Memory	Stack SRAM
ALUNIT	**ARITHMETIC AND LOGIC UNIT**	Microdot arithmetic and logic functions
ALUCONT	ALU Controller	Selects the active ALU operation based on the current ALU CODE
ALUNOT	ALU NOT Function	Not Logic
ALUSHF	ALU SHIFT Function	Shift Right and Shift Left Logic
ALUAND	ALU AND Function	And Logic
ALUOR	ALU OR Function	Or Logic
ALUXOR	ALU XOR Function	Exclusive-or Logic
ALUADD	ALU ADD Function	Add/Subtract Logic
ALUSTR	ALU Status Detector	ALU Status Logic (Carry, Negative, Overflow, and Zero)
IOUNIT	**INPUT/OUTPUT UNIT**	Interface to the 8 Input/Output Ports
OUTREG	Output Register	Output Register written to during the Store I/O (STIO) Instruction
MSKREG	Mask Register	Holds the state of the I/O lines as input or output (0 = input, 1 = output)
EVNTDET	Event Detection Logic	Detects if an event has taken place on the I/O lines during a WAIT instruction
COMUNIT	**I^2C COMMUNICATION UNIT**	Interface to the I^2C serial bus
COMBUF	Serial Communication Buffer	Serial-in/out, Parallel-in/out Shift Register for interfacing the I^2C bus and detecting I^2C address
COMREG	Serial Communication Register	Register for interfacing between the processor and the I^2C Com Unit
COMWS	Word State	Part of the I^2C Com State Machine -- Determines if I^2C is Reading Address/Reading Command/Writing Data/Programming

COMBS	Bit State	Part of the I^2C Com State Machine -- Synchronizes data load and acknowledge
COMSTM	I^2C Communication State Machine	I^2C Control State Machine -- Asserts control signals to execute I^2C commands
COMLOG	I^2C Communication Logic	Internal and External Control Logic for the I^2C COM Unit
STDET	Start Detector	Detects a Start condition on the I^2C bus
STATMUX	**STATUS MULTIPLEXER**	Test multiplexer for viewing internal data lines

Table A-2. Microdot Signal Table

Name	Bits	Source	Description	Destinations	Other names
AC	4	MEMORY	ALU CODE	CONUNIT, ALUNIT, IOUNIT, STKUNIT, STATMUX	
ADR	7	EXTERNAL	Microdot I^2C Address	I^2C COMUNIT	
ADS	2	CONUNIT	Stack Address Select	STKUNIT	
AR	4	ALUNIT	ALU Result	STKUNIT, STATMUX	
CB	1	CONUNIT	Status Register Carry Bit	ALUNIT, STATMUX	CIN
COMPCI	1	I^2C COMUNIT	Increment Program Counter	CONUNIT	
COMRST	1	I^2C COMUNIT	I^2C COM Unit Reset	CONUNIT	
COUT	1	ALUNIT	Carry Out	CONUNIT	
CR	8	I^2C COMUNIT	I^2C COM Register	MEMUNIT, STKUNIT	
DECSP	1	CONUNIT	Decrement Stack Pointer	STKUNIT	
DIM	1	EXTERNAL	Disable Internal Memory	MEMUNIT	
DMSK	8	IOUNIT	Direction Mask	EXTERNAL PADS	
DTS	3	CONUNIT	Top of Stack Data Select	STKUNIT	
EPI	8	EXTERNAL	I/O Port Input	IOUNIT	
EPO	8	IOUNIT	I/O Port Output	EXTERNAL	
EVNT	1	IOUNIT	Event Detected	CONUNIT	
IDR	1	I^2C COMUNIT	Input Data Ready	CONUNIT	
INCSEL	1	CONUNIT	Increment Select	STKUNIT	
INCSP	1	CONUNIT	Increment Stack Pointer	STKUNIT	
INT	1	I^2C COMUNIT	Interrupt Output	EXTERNAL PAD	
INTS	1	CONUNIT	Set Interrupt	I^2C COMUNIT	
INTX	1	EXTERNAL	Interrupt Input	CONUNIT	
LCRS	1	CONUNIT	Load I^2C COM Register from STK	I^2C COMUNIT	
LDAC	1	CONUNIT	Load ALU CODE	MEMUNIT	
LDMR	1	CONUNIT	Load Mask Register	IOUNIT	
LDOC	1	CONUNIT	Load OPCODE	MEMUNIT	

Name	Bits	Source	Description	Destinations	Other names
LDSKR	1	CONUNIT	Load Stack Buffer Register	STKUNIT	
LDSTK	1	CONUNIT	Load Stack RAM	STKUNIT	
LDTOS	1	CONUNIT	Load Top of Stack Register	STKUNIT	
LDX1	1	CONUNIT	Load INDEX 1 Register	MEMUNIT	
LDX2	1	CONUNIT	Load INDEX 2 Register	MEMUNIT	
MEMDI	1	PROGRAM RAM	Program RAM Input Data	MEMUNIT	
MEMDO	4	MEMORY	Memory Data Output	PROGRAM RAM	
NB	1	CONUNIT	Status Register Negative Bit	STATMUX	
NEG	1	ALUNIT	Negative Result	CONUNIT	
OC	4	MEMUNIT	OPCODE	CONUNIT, STATMUX	
ODR	1	I^2C COMUNIT	Output Data Ready	CONUNIT	
OVR	1	ALUNIT	Arithmetic Overflow	CONUNIT	
PCBR	1	CONUNIT	Branch Program Counter	MEMUNIT	
PCE	1	MEMUNIT	Program Counter End	I^2C COMUNIT	
PCIN	1	CONUNIT	Increment Program Counter	MEMUNIT	
PCLD	1	CONUNIT	Load Program Counter	MEMUNIT	
PLN	4	IOUNIT	I/O Port Internal (Lower)	STKUNIT	IP[3..0]
PMWE	1	MEMUNIT	Memory Write Enable	PROGRAM RAM	
PNS	1	I^2C COMUNIT	Program Nibble Select	MEMUNIT	
PROG	1	I^2C COMUNIT	Programming	MEMUNIT, CONUNIT	
PUN	4	IOUNIT	I/O Port Internal (Upper)	STKUNIT	IP[7..4]
RDST	1	I^2C COMUNIT	Reading Stack	CONUNIT	
RIDR	1	CONUNIT	Reset Input Data Ready	I^2C COMUNIT	
RST	1	EXTERNAL	System Reset	I^2C COMUNIT, IOUNIT, CONUNIT, STKUNIT, MEMUNIT	
RSTC	1	CONUNIT	Reset I^2C COM Unit	I^2C COM UNIT	

Name	Bits	Source	Description	Destinations	Other names
RSTP	1	CONUNIT	Reset Microdot	MEMUNIT, STKUNIT, IOUNIT, CONUNIT	
SCL	1	EXTERNAL	I^2C Serial Clock/System Clock	I^2C COMUNIT, IOUNIT, CONUNIT, STKUNIT, MEMUNIT	
SDA	1	EXTERNAL	I^2C Serial Data	I^2C COMUNIT	SDA_IN
SDA_OUT	1	I^2C COMUNIT	Serial Data Output	EXTERNAL PAD	
SMWE	1	STKUNIT	Stack RAM Write Enable	STACK RAM	
SMX	4	STATMUX	Status Multiplexer	EXTERNAL	
SPI	1	I^2C COMUNIT	Microdot Suspended	CONUNIT	
STK	4	STKUNIT	Stack Memory Output	ALUNIT,I^2C COMUNIT, IOUNIT, STATMUX	
STKADR	7	STKUNIT	Stack RAM Address	STACK RAM	
STKDI	4	STACK RAM	Stack RAM Input Data	STKUNIT	
STKDO	4	STKUNIT	Stack RAM Output Data	STACK RAM	
STOR	1	CONUNIT	Store to I/O Port	IOUNIT	
SUSP	1	CONUNIT	Set Suspend	I^2C COMUNIT	
TOS	4	STKUNIT	Top-of-Stack Register	ALUNIT,I^2C COMUNIT, IOUNIT, STATMUX	
VB	1	CONUNIT	Status Register Overflow Bit	ALUNIT, STATMUX	VIN
X1	4	MEMUNIT	INDEX1	STKUNIT, IOUNIT	
X2	4	MEMUNIT	INDEX2	IOUNIT	
XAD	10	MEMUNIT	Program Counter	EXTERNAL, PROGRAM RAM	
XDE	1	MEMUNIT	External Data Output Enable	EXTERNAL PADS	
XDI	1	EXTERNAL	External Data Input	MEMUNIT	
XDO	4	MEMUNIT	External Data Output	EXTERNAL	

Name	Bits	Source	Description	Destinations	Other names
ZB	1	CONUNIT	Status Register Zero Bit	STATMUX	
ZER	1	ALUNIT	Zero Result	CONUNIT	

Appendix B. Microdot Design Summary -- Radiation Hardened By Design Gate Array Library

Table B-1. Microdot RHBD Gate Array Cell Breakdown

Cell	Number	GA Cells	Function
AND2X1	30	2	2-input AND
AND3X1	15	2	3-input AND
AND4X1	1	3	4-input AND
AND5X1	7	3	5-input AND
AO211X1	2	3	4-input AND-OR
AO21X1	2	3	3-input AND-OR
AO221X1	1	3	5-input AND-OR
AO222X1	18	4	6-input AND-OR
AO22X1	6	3	4-input AND-OR
AOI21	11	2	3-input AND-OR INVERT
AOI211	4	2	4-input AND-OR-INVERT
AOI22	79	2	4-input AND-OR-INVERT
AOI222	9	3	6-input AND-OR-INVERT
BL_ZBUF1X1	18	3	Tri-state Buffer with 1X drive
BL_ZBUF1X2	6	4	Tri-state Buffer with 2X drive
BN_DIDFF_X1	39	18	Negative-edge D flip-flop
BN_DIDFFC_X1	64	24	Negative-Edge D flip-flop with Clear
BN_DIDFFP_X1	1	24	Negative-Edge D flip-flop with Preset
BP_DIDFF_X1	12	18	Positive-Edge D flipflop
BUF1X1	17	1	Buffer with 1X drive
BUF1X2	8	2	Buffer with 2X drive
BUF1X3	1	2	Buffer with 3X drive
INV1X1	132	1	Inverter with 1X drive
INV1X2	15	1	Inverter with 2X drive
INV1X3	4	2	Inverter with 3X drive
INV1X4	3	2	Inverter with 4X drive
MAJI3	2	3	3-input Majority
MUX2X1	26	3	2-input Multiplexer
NAN2	60	1	2-input NAND
NAN3	5	2	2-input NAND

NAN4	9	2	2-input NAND
NOR2	52	1	2-input NOR
NOR3	14	2	2-input NOR
NOR4	13	2	2-input NOR
OAI21	20	2	3-inputOR-AND-INVERT
OAI211	7	2	4-input OR-AND-INVERT
OAI22	26	2	4-input OR-AND-INVERT
OAI222	4	3	6-input OR-AND-INVERT
OR2X1	9	2	2-input OR
OR3X1	6	2	3-input OR
XNOR2	5	3	2-input Exclusive-NOR
XOR2	19	3	2-input Exclusive-OR
Total	**782**	**3684**	

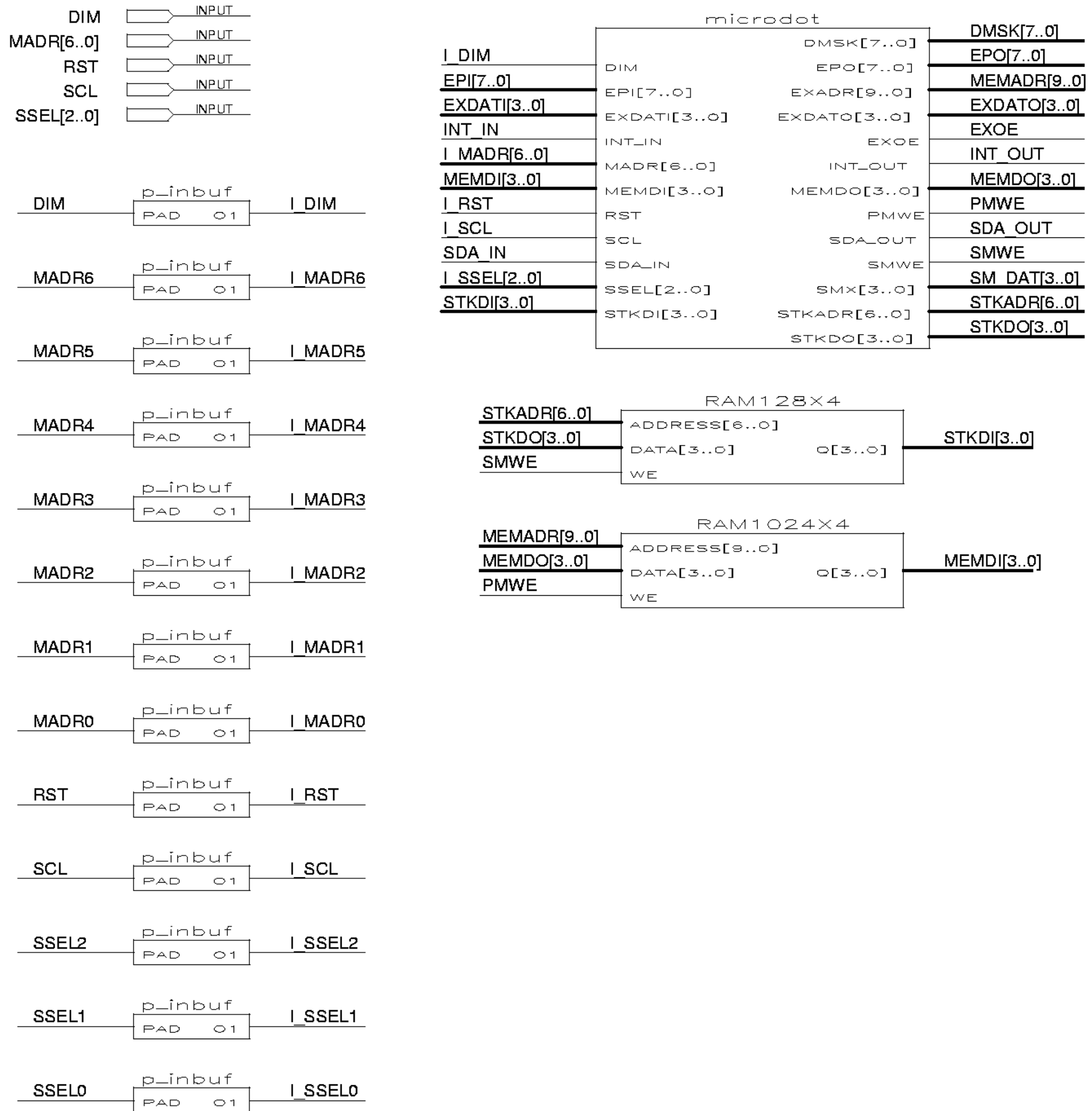


Figure C-1. Microdot Test Chip Diagram (1 of 3).
External inputs and input pads, microdot and memory blocks.

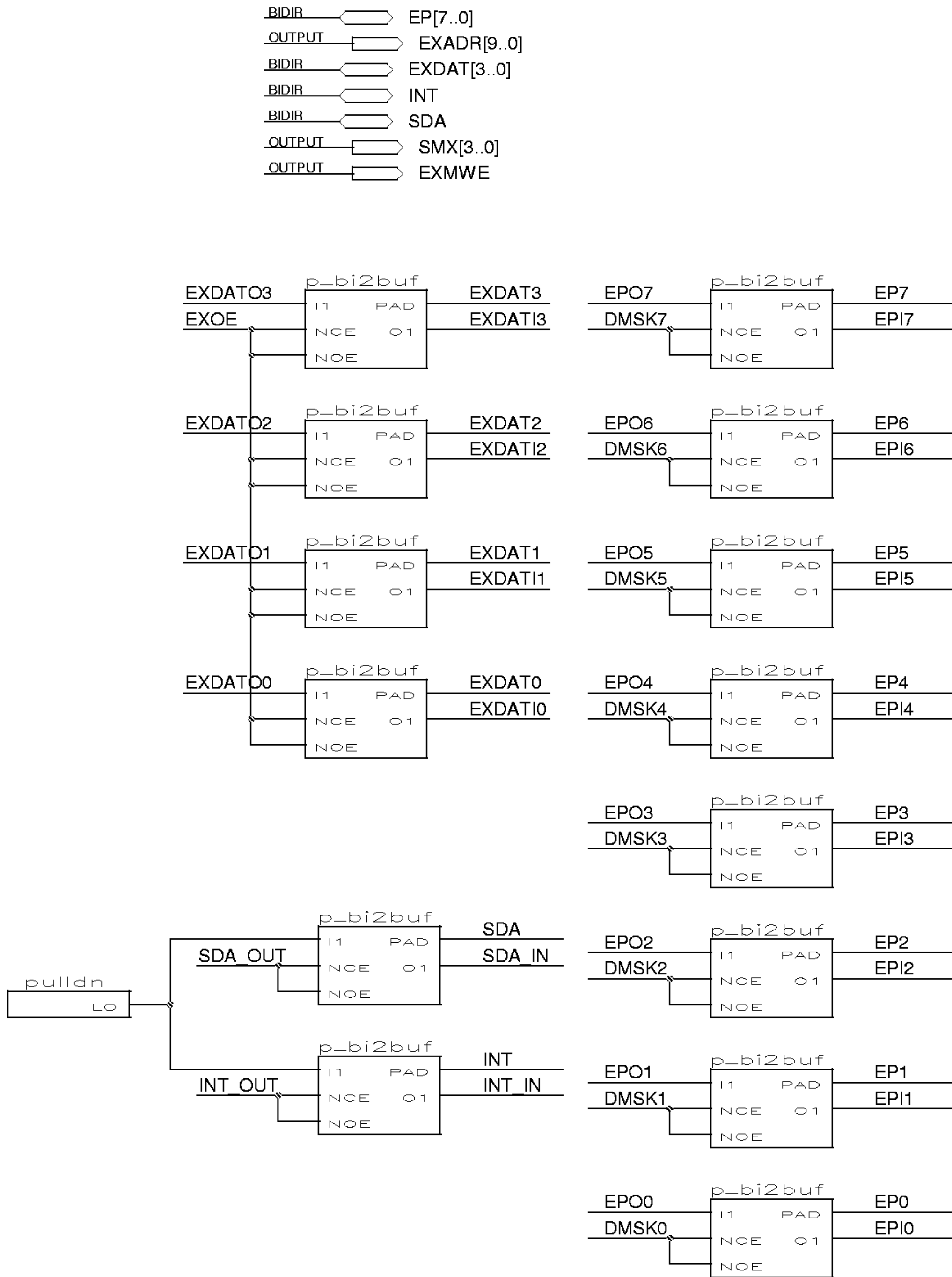


Figure C-2. Microdot Test Chip Diagram (2 of 3).
External outputs and bi-directional pads.

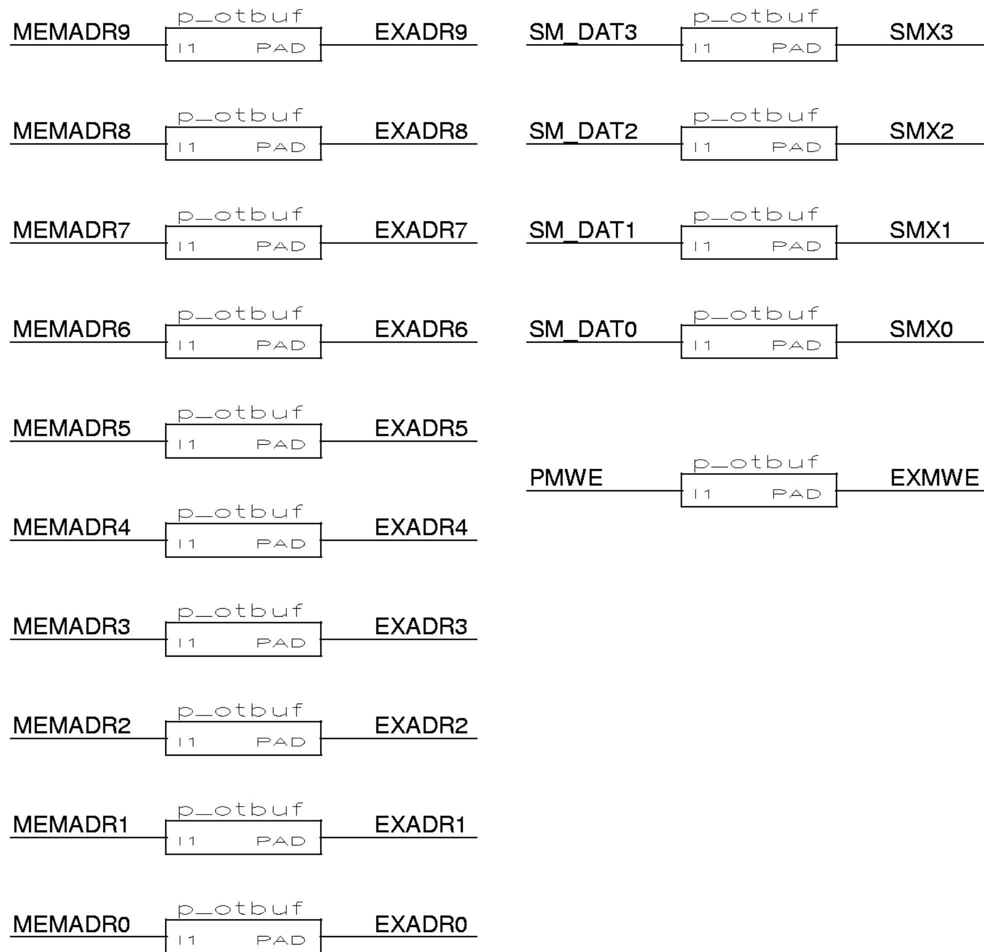


Figure C-3. Microdot Test Chip Diagram (3 of 3).
Output pads.

Appendix D. List of Abbreviations

AFOSR – Air Force Office of Scientific Research

ALU – Arithmetic and Logic Unit

ASIC – Application Specific Integrated Circuit

BJT – Bipolar Junction Transistor

CAD – Computer Aided Design

CISC – Complex Instruction Set Computer

CMOS – Complementary Metal Oxide Semiconductor

DARPA – Defense Advanced Research Project Agency

EDAC – Error Detection and Correction

EEPROM – Electrically-erasable Programmable Read-only Memory

FPGA – Field Programmable Gate Array

HEX – Hexadecimal

I/O – Input/Output

I^2C – Inter-IC Communication

Kbps – Kilobits per second

KIPS – Thousand Instructions per second

LCD – Liquid Crystal Display

LET – Linear Energy Threshold

Mbps – Megabits per second

MDOT – Microdot

MIPS – Million Instructions per second

MOS – Metal Oxide Semiconductor

MRC – Mission Research Corporation

MSB – Most Significant Bit

NFET – N-channel Field Effect Transistor

Nibble – Four-bit Word

PFET – P-channel Field Effect Transistor

PROM – Programmable Read-only Memory

Rad – Radiation Absorbed Dose

RAM – Random Access Memory

RHBD – Radiation Hardened By Design

RISC – Reduced Instruction Set Computer

ROM – Read-only Memory

SCL – I^2C Serial Clock

SCR – Silicon Controlled Rectifier

SDA – I^2C Serial Data

SEE – Single Event Effects

SEL – Single Event Latchup

SEU – Single Event Upset

SOI – Silicon-on-Insulator

SPI – Serial Peripheral Interface

SRAM – Static Random Access Memory

TID – Total Ionizing Dose

TMR – Triple Modular Redundancy

TSMC – Taiwan Semiconductor Manufacturing Company

Vdd – Supply Voltage

VHDL – Very High Speed Integrated Circuit Hardware Description Language

VLSI – Very Large Scale Integration

References

1. Watson, Kirby M. *Microdot, A 4-bit Synchronous Microcontroller for Space Applications*. MS Thesis, AFIT/GE/ENG/01M-20. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, March 2001.

2. Donohoe, Greg W. *Microdot 4-bit Stack Machine: Architectural Description*. Unpublished. June 2000.

3. Donohoe, G. W., J.C. Lyke, S. Cannon, "Microdot: a Tiny Microcontroller for Distributed Sensor Interfacing," *Proc. 2nd International Conference on Integrated MicroNano Technology*. April 1999.

4. Kang, S.M., Y. Leblebici, *CMOS Digital Integrated Circuits: Analysis and Design, Second Edition*. Boston: McGraw-Hill, 1999.

5. Hamacher, C.V., Z.G. Vranesic, S.G. Zaky, *Computer Organization, Fourth Edition*. New York: McGraw-Hill, 1996.

6. Stevens, Roger L. *Serial PIC'n: PIC Microcontroller Serial Communications*. Kelseyville, California: Square 1 Electronics, 1999.

7. Awtrey, Dan. “Transmitting Data and Power Over a 1-Wire Bus,” http://pdfserv.maxim-ic.com/arpdf/AppNotes/onewirebus.pdf. 1997.

8. Philips Semiconductors. “The I^2C-Bus Specification, Version 2.1,” http://www.semiconductors.philips.com/acrobat/various/... ...I2C_BUS_SPECIFICATION_3.PDF. 2000.

9. Palmintier, B., R. Twiggs, C. Kitts, “Distributed Computing on Emerald: A Modular Approach for Robust Distributed Space Systems,” *2000 IEEE Aerospace Conference Proceedings*, 7: pages 211-222, 2000.

10. Sharma, Ashok K. *Semiconductor Memories: Technology, Testing, and Reliability*. Piscataway, New Jersey: IEEE Press, 1997.

11. Jet Propulsion Laboratory, Radiation Effects Group. “Space Radiation Effects on Microelectronics,” http://nppp.jpl.nasa.gov/docs/Radcrs_Final2.pdf. 2000.

12. van Vonno, N., J. Barth, L. Lorence, D. Beutler, E. Petersen, J. Swonger, “Applying Computer Simulation Tools to Radiation Effects Problems,” *1997 IEEE NSREC Short Course*. 1997.

13. Winokur, Peter S. “Why Semiconductors Must Be Hardened for Space Deployment,” http://www.ieee.org/organizations/pubs/newsletters/npss/june2000/semi.htm. 2000.

Vita

Captain Anthony R. Woodcock was raised in Washington, Kansas and graduated from Washington High School in May 1992. He entered undergraduate studies at Kansas State University in Manhattan, Kansas where he graduated with a Bachelor of Science degree in Computer Engineering in May 1997. He was commissioned through Detachment 270 at Kansas State University.

His first assignment was at the Air Force Research Laboratory, Wright-Patterson AFB where he was a development engineer for combat identification systems in the Reconnaissance, Strike & Combat ID branch. In August 2000, he entered the Graduate School of Engineering and Management, Air Force Institute of Technology. Upon graduation, he will be assigned to the Air Force Information Warfare Center (AFIWC) at Lackland AFB, Texas.