AD-A232 554

NAVAL POSTGRADUATE SCHOOL
Monterey, California

DTIC
ELECTE
MAR 0 6 1991
S B D

THESIS

DESIGNING A VIRTUAL-MEMORY IMPLEMENTATION USING THE MOTOROLA MC68010 16-BIT MICROPROCESSOR WITH MULTI-PROCESSOR CAPABILITY INTERFACED TO THE VMEbus

by

David M. Sendek

June 1990

Thesis Advisor: Larry W. Abbott

Approved for public release; distribution is unlimited.

91 3 04 004

REPORT DOCUMENTATION PAGE

Form Approved
OMB No 0704-0188

1a REPORT SECURITY CLASSIFICATION: UNCLASSIFIED	1b RESTRICTIVE MARKINGS
2a SECURITY CLASSIFICATION AUTHORITY	3 DISTRIBUTION AVAILABILITY OF REPORT: Approved for public release; distribution is unlimited.
2b DECLASSIFICATION DOWNGRADING SCHEDULE	
4 PERFORMING ORGANIZATION REPORT NUMBER(S)	5 MONITORING ORGANIZATION REPORT NUMBER(S)
6a NAME OF PERFORMING ORGANIZATION: Naval Postgraduate School; 6b OFFICE SYMBOL (If applicable): EC	7a NAME OF MONITORING ORGANIZATION: Naval Postgraduate School
6c ADDRESS (City, State, and ZIP Code): Monterey, California 93943-5000	7b ADDRESS (City, State, and ZIP Code): Monterey, California 93943-5000
8a NAME OF FUNDING SPONSORING ORGANIZATION; 8b OFFICE SYMBOL (If applicable)	9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c ADDRESS (City, State, and ZIP Code)	10 SOURCE OF FUNDING NUMBERS: PROGRAM ELEMENT NO; PROJECT NO; TASK NO; WORK UNIT ACCESSION NO

11 TITLE (Include Security Classification): DESIGNING A VIRTUAL-MEMORY IMPLEMENTATION USING THE MOTOROLA MC68010 16-BIT MICROPROCESSOR WITH MULTI-PROCESSOR CAPABILITY INTERFACED TO THE VMEbus (U)

12 PERSONAL AUTHOR(S): Sendek, David, M.

13a TYPE OF REPORT	13b TIME COVERED	14 DATE OF REPORT (Year, Month, Day)	15 PAGE COUNT
Master's Thesis	FROM ____ TO ____	June 1990	175

16 SUPPLEMENTARY NOTATION: The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government

17 COSATI CODES			18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	MC68010 Microprocessor, VMEbus, Virtual-Memory, Dual-port Memory, Multi-processor

19 ABSTRACT (Continue on reverse if necessary and identify by block number)

The primary purpose of this thesis is to explore and discuss the hardware design of a bus-oriented microprocessor system. A bus-oriented microprocessor system permits it to be expanded to a multi-processor system. Through the use of a bus controller and bus arbiter, as discussed in this thesis, the necessary logic is in place to control bus access by system users. Bus access may be initiated to share another sub-system's resource, such as memory. To accommodate memory sharing between two systems, a dual-port memory controller can be used to resolve memory access between the two systems. This thesis discusses the design of a MC68010 microprocessor system integrated on the VMEbus with dual-ported memory capability. Additional features of the MC68010 microprocessor system include memory-management and interrupt control. The memory-management features permit protected memory and virtual-memory to be implemented on the system, while an interrupt handler is used to assist the MC68010 microprocessor in exception processing.

20 DISTRIBUTION AVAILABILITY OF ABSTRACT	21 ABSTRACT SECURITY CLASSIFICATION
☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS	Unclassified
22a NAME OF RESPONSIBLE INDIVIDUAL: Larry W. Abbott	22b TELEPHONE (Include Area Code): 713-483-8593; 22c OFFICE SYMBOL: EC/AT

DD Form 1473, JUN 86 — Previous editions are obsolete — S/N 0102-LF-014-6603

Approved for public release; distribution is unlimited.

DESIGNING A VIRTUAL-MEMORY IMPLEMENTATION USING THE MOTOROLA MC68010 16-BIT MICROPROCESSOR WITH MULTI-PROCESSOR CAPABILITY INTERFACED TO THE VMEbus

by

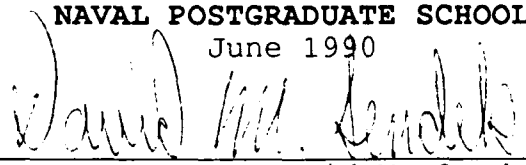
David M. Sendek
Lieutenant, United States Navy
B.S., The College of Charleston, 1981

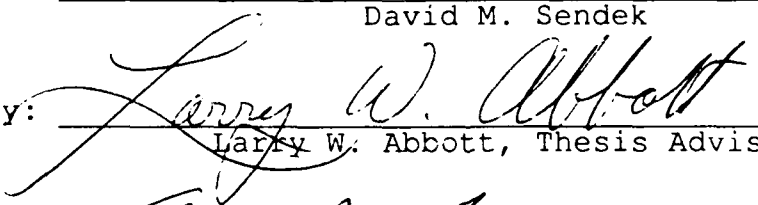
Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

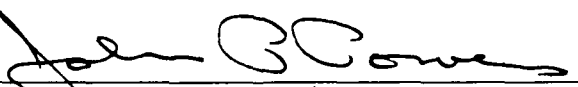
from the

NAVAL POSTGRADUATE SCHOOL
June 1990

Author: 
David M. Sendek

Approved by: 
Larry W. Abbott, Thesis Advisor

Fred W. Terman, Second Reader


John R. Powers, Chairman, Department of
Electrical and Computer Engineering

ABSTRACT

The primary purpose of this thesis is to explore and discuss the hardware design of a bus-oriented microprocessor system. A bus-oriented microprocessor system permits it to be expanded to a multi-processor system. Through the use of a bus controller and bus arbiter, as discussed in this thesis, the necessary logic is in place to control bus access by system users. Bus access may be initiated to share another sub-system's resource, such as memory. To accommodate memory sharing between two systems, a dual-port memory controller can be used to resolve memory access between the two systems. This thesis discusses the design of a MC68010 microprocessor system integrated on the VMEbus with dual-ported memory capability. Additional features of the MC68010 microprocessor system include memory-management and interrupt control. The memory-management features permit protected memory and virtual-memory to be implemented on the system, while an interrupt handler is used to assist the MC68010 microprocessor in exception processing.

Accession For
NTIS GRA&I
DTIC TAB
Unannounced
Justification
By
Distribution/
Availability Codes
Dist | Avail and/or Special
A-1

QUALITY INSPECTED 3

TABLE OF CONTENTS

I. INTRODUCTION ---------- 1

II. DESIGN CONCEPTS ---------- 6

A. VMEbus SPECIFICATION ---------- 7

1. Background ---------- 7

2. VMEbus Description ---------- 7

3. Configurations ---------- 9

a. Slave-Only Application ---------- 9

b. Master-Only Application ---------- 10

c. Master-Slave Application ---------- 12

4. Arbitration Protocols ---------- 14

B. MEMORY-MANAGEMENT ---------- 18

1. Memory Protection ---------- 18

2. Virtual-Memory ---------- 19

3. Dual-ported Memory ---------- 22

III. SYSTEM OVERVIEW ---------- 24

A. SYSTEM CONTROLLER CIRCUIT BOARD ---------- 24

1. Priority Bus Arbitration ---------- 24

2. Manual Reset ---------- 25

3. Interrupt Driver ---------- 26

B. MASTER CIRCUIT BOARD ---------- 26

1. Central Processor Unit ---------- 26

2. Dual Universal Asynchronous Receiver/ Transmitter ---------- 27

3. Erasable Programmable Read-Only Memory ---- 27

4. Random Access Memory ---------- 28

5. Memory Management Unit ---------- 28

6. Dual-port DRAM Controller ---------- 28

7. VMEbus Controller ---------- 29

8. Interrupt Handler ---------- 29

IV. DESIGN IMPLEMENTATION ---------- 31

A. MINIMAL SYSTEM ---------- 32

1. Memory Map ---------- 32

2. Hardware Interface ---------- 35

3. Software Support ---------- 38

a. Exception Vector Table and Monitor/ Debugger Program ---------- 38

b. Monitor/Debugger Commands ---------- 38

c. Programmable Logic Device Programming - 40

B. FULLY INTEGRATED SYSTEM ---------- 41

1. Memory Map ---------- 41

2. Master Circuit Board ---------- 44

a. Microprocessor ---------- 44

b. Halt and Reset Generation ---------- 45

c. Clock Generation ---------- 46

d. Local Bus Address Decoding ---------- 46

e. Memory Management Unit ---------- 46

f. Dual-port DRAM Controller ---------- 48

g. Dynamic Random Access Memory ---------- 50

h. EPROM and SRAM ---------- 51

i. Dual Serial Port ---------- 51

j. Interrupt Handler ---------- 51

k. Data Transfer Acknowledge and Bus Error Generation ---------- 53

l. VMEbus Controller ---------- 55

m. VMEbus Address Decoding ---------- 56

n. VMEbus Drivers ---------- 56

3. System Controller Circuit Board ---------- 57

a. Bus Arbiter ---------- 57

b. System Reset ---------- 57

c. VMEbus Drivers ---------- 57

V. RESULTS ---------- 59

VI. SUMMARY AND CONCLUSIONS ---------- 62

A. SUMMARY ---------- 62

1. Design Concepts ---------- 62

a. VMEbus Structure ---------- 62

b. Memory-Management ---------- 62

c. Interrupt Control ---------- 64

2. Design Implementation ---------- 65

a. Hardware Configurations ---------- 65

b. Erasable Programmable Logic Devices --- 66

B. CONCLUSIONS ---------- 67

APPENDIX A (MC68010 16-BIT MICROPROCESSOR) ---------- 69

APPENDIX B (MINIMAL SYSTEM EXCEPTION VECTOR TABLE AND MONITOR/DEBUGGER PROGRAM) ---------- 77

APPENDIX C (MINIMAL SYSTEM DIAGRAMS) ---------- 117

APPENDIX D (MINIMAL SYSTEM'S PROGRAMMABLE LOGIC DEVICE SOURCE CODE) ---------- 126

APPENDIX E (SYSTEM DIAGRAMS) ---------- 131

LIST OF REFERENCES ---------- 159

BIBLIOGRAPHY ---------- 161

INITIAL DISTRIBUTION LIST ---------- 162

LIST OF TABLES

I. Minimal System Memory Map ---------- 33

II. System Memory Map ---------- 42

III. Data Strobe Control of the Data Bus ---------- 71

IV. State and Address Space ---------- 74

LIST OF FIGURES

1.1 Generic Multi-Processor System ---------- 5

2.1 Slave-Only Subsystem ---------- 10

2.2 Master-Only Subsystem ---------- 11

2.3 Master-Slave Subsystem ---------- 13

2.4 Daisy Chain Arbitration ---------- 15

2.5 Parallel Arbitration ---------- 16

2.6 Virtual-Memory-Mapping ---------- 20

2.7 Mapping Mechanism ---------- 21

2.8 Dual-ported Memory ---------- 22

3.1 System Block Diagram ------- ---------- 25

4.1 Minimal System ---------- 31

A.1 MC68010 Signal Groups ---------- 70

C.1 Minimal System MC68010 Microprocessor Circuitry ---- 118

C.2 Minimal System HALT* and RESET* Generation Circuitry 119

C.3 Minimal System Clock Generation Circuitry ---------- 120

C.4 Minimal System Address Decode Circuitry ---------- 121

C.5 Minimal System DTACK* and BERR* Generation Circuitry 122

C.6 Minimal System EPROM and SRAM Circuitry ---------- 123

C 7 Minimal System Interrupt Request and Interrupt Acknowledge Circuitry ---------- 124

C.8 Minimal System Dual-port Receiver/Transmitter Serial Port Circuitry ---------- 125

E.1 Master Circuit Board Functional Block Diagram ------ 132

E.2 System Controller Circuit Board Functional Block Diagram ---------- 133

E.3 MC68010 Microprocessor Circuitry ---------- 134

E.4 HALT* and RESET* Generation Circuitry ---------- 135

E.5 Clock Generation Circuitry ---------- 136

E.6 Local Bus Address Decode Circuitry ---------- 137

E.7 Memory Management Unit Circuitry (Page 1 of 2) ----- 138

E.8 Memory Management Unit Circuitry (Page 2 of 2) ----- 139

E.9 Dual-port DRAM Controller Circuitry (Page 1 of 3) -- 140

E.10 Dual-port DRAM Controller Circuitry (Page 2 of 3) -- 141

E.11 Dual-port DRAM Controller Circuitry (Page 3 of 3) -- 142

E.12 Dynamic Random Access Memory Circuitry
(Page 1 of 4) ---------- 143

E.13 Dynamic Random Access Memory Circuitry
(Page 2 of 4) ---------- 144

E.14 Dynamic Random Access Memory Circuitry
(Page 3 of 4) ---------- 145

E.15 Dynamic Random Access Memory Circuitry
(Page 4 of 4) ---------- 146

E.16 EPROM and SRAM Circuitry ---------- 147

E.17 Dual-port Asynchronous Receiver/Transmitter Serial
Port Circuitry ---------- 148

E.18 Interrupt Handler Circuitry ---------- 149

E.19 DTACK* and BERR* Generation Circuitry ---------- 150

E.20 VMEbus Controller Circuitry ---------- 151

E.21 VMEbus Address Decode Circuitry ---------- 152

E.22 Master Circuit Board VMEbus Drivers Circuitry
(Page 1 of 3) ---------- 153

E.23 Master Circuit Board VMEbus Drivers Circuitry
(Page 2 of 3) ---------- 154

E.24 Master Circuit Board VMEbus Drivers Circuitry
(Page 3 of 3) ---------- 155

E.25 VMEbus Arbitration Circuitry ---------- 156

E.26 SYSRESET* Generation Circuitry ---------- 157

E.27 System Controller VMEbus Drivers Circuitry --------- 158

I. INTRODUCTION

Economic pressure constantly forces computer design and technology to produce more cost-effective system implementations. Computers are made more cost-effective by lowering operating cost through increased speed and power and by lowering design, maintenance and upgrade costs through modular design techniques. Architectural innovations can accelerate this process. Hence, new innovations in system architecture are constantly sought after. Architecture is used here to mean the structuring of the modules which are organized into a computer system [Ref. 1:p. 1]. These modules include processors, memory and input/output (I/O) devices.

A uni-processor system consists of a single processor subsystem and various supporting modules integrated to form a system. In contrast, a multi-processor system is comprised of two or more processor subsystems connected into one interrelated functional system. In a multi-processor system, the interconnection of the processor subsystems must be done in such a way as to maintain control and manage the data flow of the entire system. This may be accomplished through multi-ported memory, a serial link or as in this thesis, by a system bus. A number of computer architectural designs that accommodate growing needs are examined in this thesis. Key architectural features of bus structures, memory-management and interrupt control are described in this chapter.

Bus structures allow for the integration of peripherals, memory and application-specific boards into one coherent system. Bus structures permit the exchange of data and control signals between circuit boards. This allows circuit boards to communicate with each other and to share resources. However, a strict adherence to protocols must be maintained so the integrity of information and control is preserved.

Memory-management features include memory protection and virtual-memory. Special memory schemes have been used to protect a system's integrity, to make more effective use of its physical memory's address range and to permit multi-ported memory so that the memory resource can be shared in a multi-processor system. A memory protection scheme prevents users from inadvertently or maliciously tampering with the operating system, its associated memory-mapped hardware or other users. To accomplish this, a portion of the processor's address range can be reserved for the operating system, while the remaining portion is allocated to system users. The operating system is protected because the user is not permitted to cross into the operating system's memory.

The virtual-memory aspect of memory-management permits a greater dynamic range and flexibility for user memory than actually exists with the system's physical memory. Virtual-memory allows each user to run programs as if he or she has full use of the processor's address range, independent of the memory used by the operating system or the other users. The user is unaware of how the physical memory in the system is allocated. Therefore, memory

resources can be allocated automatically and respond to the dynamic needs of the operating system and the users. In a system without virtual-memory, programs must be executed in a specific memory space and for large programs, the user must provide complex overlay schemes to circumvent the fixed user memory allocation. It is difficult for such a system to support several large programs concurrently. In a virtual-memory system, the operating system breaks up the user's program into segments called pages and moves these pages as needed between physical memory and a secondary storage device such as a hard disk. Thus, a virtual-memory system can easily support several large programs concurrently as long as each program only requires a modest amount of memory at any given time.

Multi-ported memory, such as dual-ported memory, allows a common memory resource to be shared between two or more processors or peripheral devices. Thus, different processes or different processors can communicate with each other via a multi-ported memory mailbox equipped with an accompanying semaphore to maintain access control and data integrity. Also, multi-porting provides a communication link between tightly coupled systems where there is a high degree of interaction.

Interrupts optimize the performance of a processor. An interrupt is a control signal generated asynchronously by a device, such as a serial port, requesting service from the processor. The processor is free to process other tasks between interrupts from devices requiring service [Ref. 2:pp. 220-223]. When it is ready

to service an interrupting device, the processor saves its current state and then performs the servicing tasks. When the servicing tasks are completed, the saved state of the processor is restored and the operation prior to the interrupt is resumed. Consequently, the processing power of the processor is increased because the overhead from polling peripheral devices for a service request is eliminated.

In a general sense, a generic multi-processor system can be viewed as illustrated in Figure 1.1. Various subsystems such as data processing, storage and data communications are integrated along a system bus to make up a complete system. Each subsystem is comprised of memory, I/O and processor modules configured to accommodate the unique requirements of the users of the multi-processor system. A system controller acts as the arbiter for the entire system. The system controller directs the information flow, much as a traffic policeman directs traffic, between the various subsystems along the system bus to ensure that the system is properly coordinated. In order for each subsystem to have access to the system bus, logic must be incorporated within each subsystem to allow it to interface to the system bus.

The main thrust of this thesis is to explore the concepts of bus structure, memory-management and interrupt control. These concepts are addressed in a greater depth than would be possible in a classroom environment.

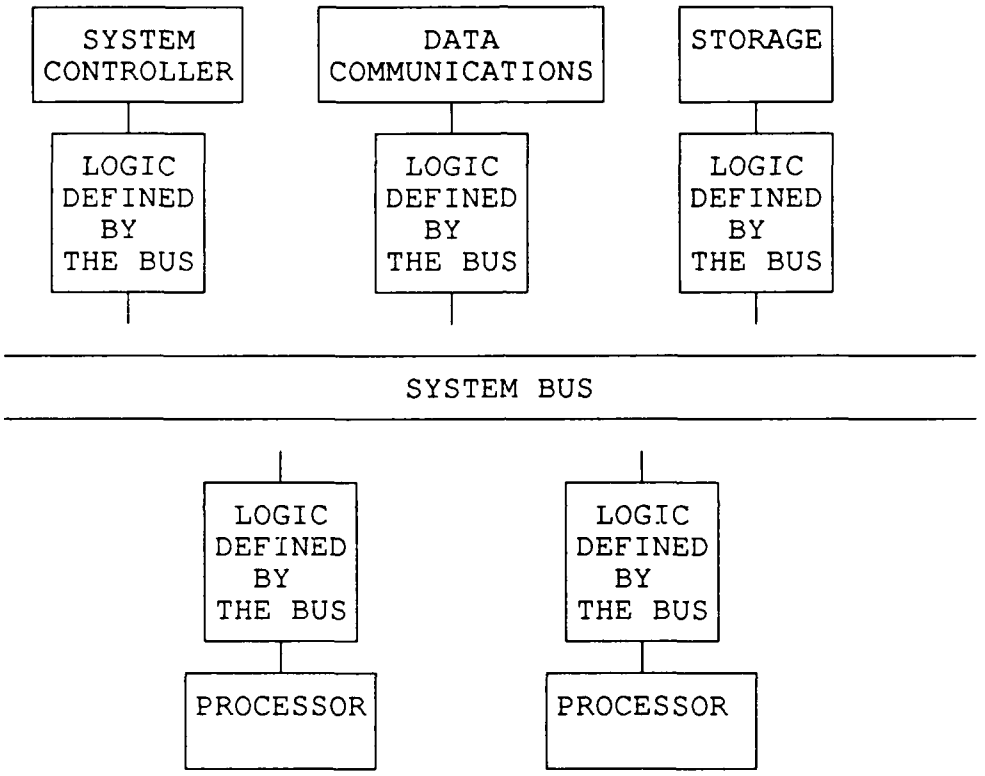


Figure 1.1: Generic Multi-Processor System

II. DESIGN CONCEPTS

The concepts addressed in this thesis are limited to bus structure organization, memory-management and interrupt control. These features are commonly used in today's processor systems. However, many options are available within each area. This thesis design is a virtual-memory implementation of a MC68010-based microprocessor system integrated on the VMEbus with dual-ported memory capability.

Borrill [Ref. 3] highlights several advantages of the VMEbus. The VMEbus, through its non-multiplexed address lines and data lines, does not have multiplexing delays as do other buses, nor does it have the transactional protocol overheads as do some other buses. In addition, the non-multiplexed address lines will support address pipelining. For interested readers, Borrill has made a detailed comparison of the features and performance of the VMEbus, Futurebus, Multibus II, Nubus and Fastbus [Ref. 3].

In addition to the advantages that Borrill highlights, the VMEbus structure was selected because of the relative ease of integrating Motorola and Signetics peripheral hardware devices. These hardware devices include a memory management unit, VMEbus controller, bus arbiter, interrupt handler hardware and dual-port dynamic random access memory (DRAM) controller.

The following discussion presents a broad overview of the VMEbus structure and memory-management. This should facilitate

understanding of the concepts that are incorporated into the final system (master circuit board) design.

A. VMEbus SPECIFICATION

1. Background

The VMEbus specification originated with Motorola's 68000 microprocessor products. The 68000 series was introduced to the marketplace in the late 1970s, using the VERSAbus specification. In the early 1980s, Motorola's European Microsystems group in Munich, Germany, introduced the Eurocard version of the VERSAbus, referred to as the VERSAbus-E specification. A joint agreement was reached to adopt the VERSAbus-E as the baseline bus specification for Motorola 68xxx devices with Mostek and Signetics as second-source suppliers of the 68xxx family of devices. The VERSAbus-E was renamed the VMEbus. The VMEbus specification [Ref. 4] delineates the mechanical and electrical characteristics of the bus and the protocols to interface devices on the VMEbus.

2. VMEbus Description

The VMEbus offers a versatile combination of timing strategies and support features. It also offers several data transfer sizes, several addressing modes and several arbitration methods. The VMEbus is an asynchronous, non-multiplexed bus that accommodates 8, 16 and 32-bit data transfers. [Ref. 5]

Asynchronous data transfers are flexible and do not impose timing control signals. Completion signals from the asynchronous devices ensure that adequate time is allowed for the data transfer. In contrast, synchronous data transfers impose a timing constraint

on the data transfer which must accommodate the slowest device attached to the bus.

A non-multiplexed bus is one that accommodates data transfers and address transfers as separate signals on separate lines of the bus. This contrasts with the multiplexing strategy where data signals and address signals share the same set of lines. As a simple description, during a write cycle, multiplexing address signals are gated on one clock cycle and data signals are gated on the same lines during a subsequent clock cycle. The non-multiplexing strategy speeds up data transfer by eliminating the second clock cycle.

The VMEbus can be used with 24 or 32 address lines depending on the microprocessor's requirements and it is easily adaptable to the entire family of Motorola 68xxx microprocessors and peripherals.

The VMEbus is composed of four sub-buses that play unique roles within the overall VMEbus functional structure. These include the data transfer bus (DTB), the data transfer arbitration bus, the priority interrupt bus and the utility bus. The VMEbus functional specification describes how each sub-bus interacts and the rules which govern the behavior of each sub-bus [Ref. 4:pp. 15-194]. The DTB provides the pathways for the data signals, the address signals and their associated control signals. The process of resolving bus ownership takes place on the data transfer arbitration bus. The priority interrupt bus is used to accommodate processes which request servicing from another subsystem. An

interrupt stops normal bus activity until the interrupt is serviced. The utilities bus is sometimes referred to as a "miscellaneous functions bus". It includes a system reset line, an alternating current (AC) power failure line, a system failure line and a system clock [Ref. 2:p. 475].

The design in this thesis uses the VMEbus controller and the interrupt handler hardware devices which are designed for use with the VMEbus.

3. Configurations

In a multi-processor VMEbus-based system with a variety of peripheral devices, each subsystem can fulfill one of three primary roles. The subsystem can serve as a slave-only, as a master-only or as a master-slave combination. A subsystem can also have the role of direct memory access (DMA) in a master-slave configuration. (To limit the size and complexity of this thesis, the DMA master-slave configuration is not discussed.) These roles determine the way the subsystem is integrated to the system bus.

a. Slave-Only Application

In the slave-only configuration, the subsystem is slaved to the VMEbus. In other words, this subsystem is incapable of making a request to obtain access and control of the VMEbus. The slave subsystem is a device which other subsystems utilize. Examples of slave subsystems include communication ports and stand alone memory boards. If intelligence (logic) is added, the subsystem can evolve into an input/output (I/O) channel or a mass storage subsystem. Figure 2.1 shows the simplicity of a slave

subsystem interfaced to the VMEbus. The 74LS245s octal-bus transceivers with 3-state outputs provide the drive capability for transmitting signals onto the VMEbus and the receiver capability for receiving signals from the VMEbus. If desired, the 74LS245s can also be disabled to isolate the slave subsystem from the VMEbus.

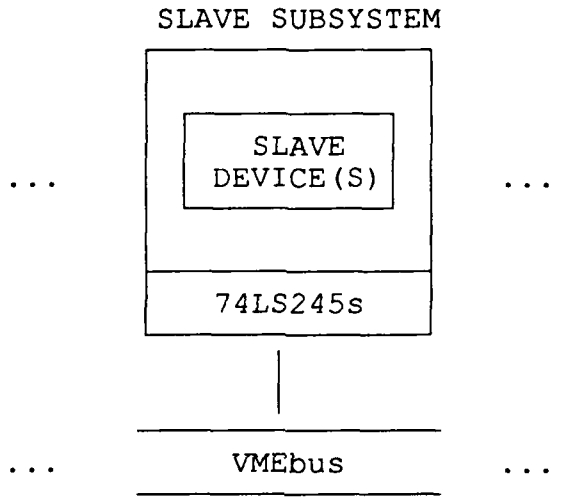


Figure 2.1: Slave-Only Subsystem

b. Master-Only Application

In the master-only configuration, the subsystem has the ability to gain control of the VMEbus. A master-only subsystem has an onboard central processor unit (CPU) with or without local slave devices. It is interfaced to the VMEbus with a bus controller. When the subsystem has gained control of the VMEbus, this subsystem is said to be in a master role. Figure 2.2 gives a simplified illustration of a VMEbus system with a master-only subsystem

attached to it. Comparison of Figures 2.1 and 2.2 shows the added complexity required in a subsystem which can gain control of the VMEbus. In addition, a system controller is included in Figure 2.2 to illustrate the added system complexity required to control bus accesses.

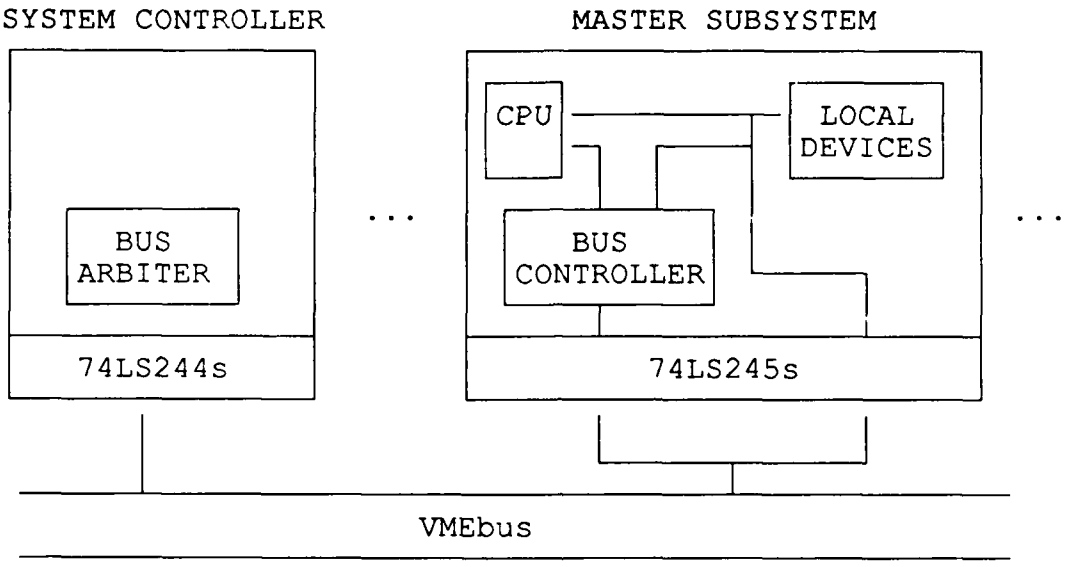


Figure 2.2: Master-Only Subsystem

Given a request by the CPU, the bus controller generates a bus request signal through an 74LS245 to the system controller's bus arbiter. (The abilities of the 74LS245 were described in the slave-only subsystem.) The bus arbiter receives requests from subsystems on the VMEbus through the 74LS244 octal-buffers and line drivers with 3-state outputs. The function of the bus arbiter is to resolve prioritized requests from the subsystems and to generate a bus grant signal through the 74LS244 to the

highest priority requesting subsystem. The subsystem's bus controller maintains system integrity by ensuring that a bus grant signal is received prior to permitting a data transfer. The requesting subsystem, after receiving the bus grant signal, negates its bus request and asserts the bus busy signal so that other subsystems cannot gain control of the bus while the data exchange is in process. Also, the bus busy signal informs the bus arbiter that a data exchange is currently in progress and that the bus arbiter can release the bus grant signal. The requesting device is now the bus master. When the data exchange is complete, the requesting device releases the bus busy signal to allow the bus arbiter the opportunity to grant the bus to another subsystem.

If the bus is in use and a higher priority bus request is asserted, the bus arbiter asserts the bus clear line. The bus clear signal informs the current bus master that another subsystem with a higher priority is requesting bus ownership. Each potential bus master should accommodate either a "release when done" or a "release on request" strategy to resolve pending higher priority requests for bus access.

c. Master-Slave Application

A master-slave configuration combines the master-only and slave-only capabilities into a single subsystem. As illustrated in Figure 2.3, the CPU residing on the master-slave subsystem has the ability to gain control of the VMEbus. The system controller and bus arbiter perform the same roles as described in the master-only subsystem.

Shared slave devices are onboard the master-slave subsystem. These devices can be accessed by another subsystem when it has control of the VMEbus (Fig. 2.3). The bus controller isolates the shared slave devices from the CPU by putting the 74LS244s outputs into a high impedance state, whenever another subsystem accesses the shared slave devices. When this happens, the shared slave devices become a global asset to the system. The 74LS245s not only act as line drivers and receivers,

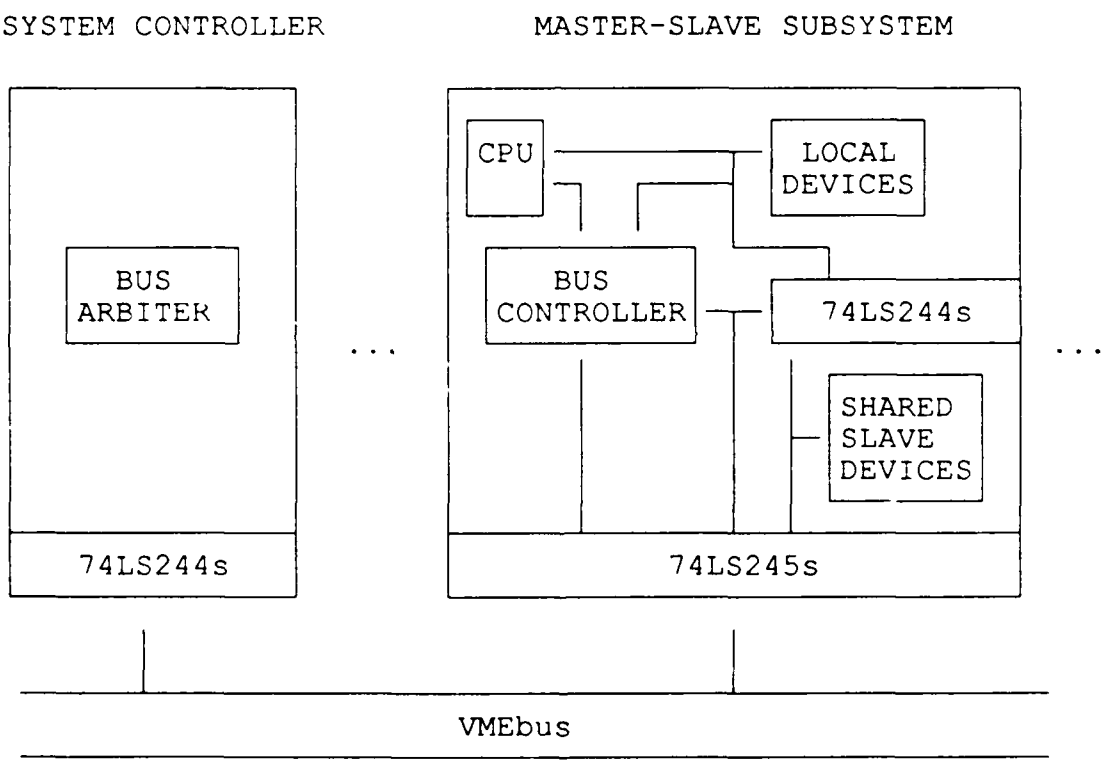


Figure 2.3: Master-Slave Subsystem

they also prevent access from the VMEbus to shared slave devices when the appropriate control signal is asserted by the bus

controller. Whenever the local master (in this case the CPU) is accessing the shared slave devices, these devices become a local asset. As discussed in the master-only application, the bus controller preserves the VMEbus protocol.

4. Arbitration Protocols

Arbitration protocols ensure conflict-free access to the system bus from all subsystems and are crucial in a multi-processor environment [Ref. 6:p. 100]. An arbitration protocol ensures that only one bus master has access to the bus at a time, thus safeguarding the bus from collisions in which information is transferred on the bus by multiple sources. The VMEbus supports both serial and parallel arbitration schemes or a combination of both methods. These two method are described in the following paragraphs.

Daisy chaining is a method of arbitrating a shared communication bus by serial prioritization. Figure 2.4 illustrates daisy chain arbitration. If the bus is in use, any subsystem requesting ownership must wait till the present bus master relinquishes control of the bus. A subsystem requests access to the bus by asserting the bus request (BR) signal. The bus arbiter or other controlling device acknowledges the bus request by asserting a bus grant (BG) signal to the bus grant input (BGIN) of SUBSYSTEM1, the first subsystem in the daisy chain. If SUBSYSTEM1 is requesting the bus, it asserts the bus busy (BBSY) signal and it continues to negate its bus grant output (BGOUT) signal. SUBSYSTEM1 can now begin data transfer. If the bus request was

made by any subsystem other than SUBSYSTEM1, the BG signal is passed by SUBSYSTEM1 to the next subsystem in the chain (SUBSYSTEM2). The BGOUT signal from SUBSYSTEM1 becomes the BGIN signal to the next subsystem in the chain (SUBSYSTEM2). This process is repeated until the highest priority requesting subsystem receives the BGIN signal. SUBSYSTEM1 has a higher priority than SUBSYSTEM2. The last subsystem in the chain (SUBSYSTEMn) has the lowest priority.

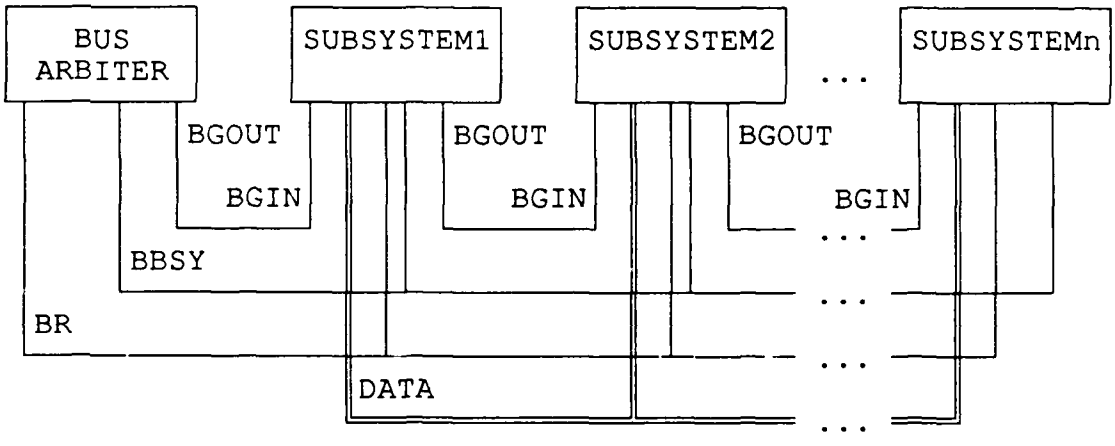


Figure 2.4: Daisy Chain Arbitration

The BR and BBSY signals are wire-ORed (open collector-active low), i.e., the logic is tied together at a wire connection. Consequently, the BR signal will cause the BBSY signal to be asserted once the BGIN signal is received through the daisy chain.

Parallel arbitration is a method of arbitrating a shared communication bus by priority levels. An example of a three-level parallel arbitration scheme is shown in Figure 2.5. In Figure 2.5,

bus request zero (BR0) has the lowest priority level, while bus request two (BR2) has the highest priority level. The highest priority subsystem with a pending request is granted access to the bus. In this parallel arbitration scheme, the subsystems desiring use of the bus make bus requests (BRx) through the bus arbiter. The bus arbiter or other controlling device then sends out a bus grant (BGx) onto the bus to the highest priority subsystem with a pending bus request.

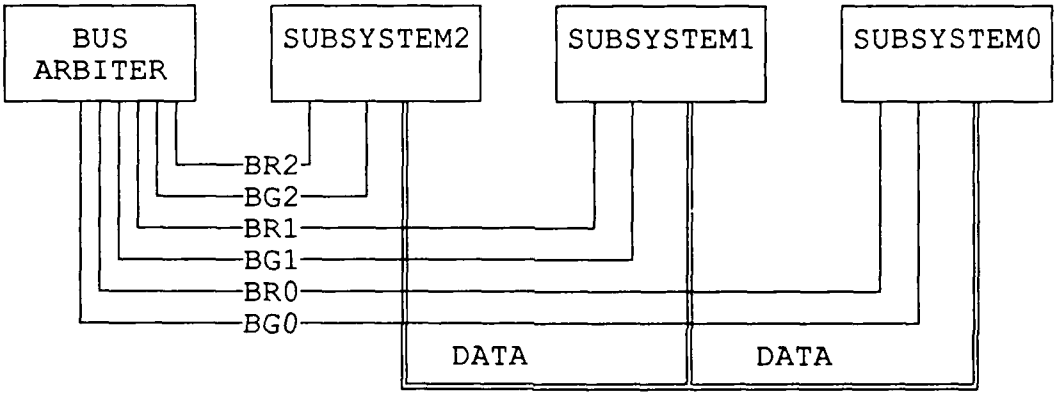


Figure 2.5: Parallel Arbitration

The main advantage of the daisy chain arbitration scheme over the parallel arbitration scheme is that subsystems can be inserted sequentially, one after the other. Consequently, new subsystems are easily added to the system.

The main advantage of the parallel arbitration scheme over the daisy chain arbitration scheme is that arbitration can be performed faster. Parallel arbitration does not propagate a bus grant signal down a chain, but rather the bus grant signal is sent

directly to the highest priority subsystem requesting service. However, the parallel arbitration scheme limits the number of subsystems that the bus arbiter can accommodate.

Any fixed priority arbitration cannot ensure that the subsystem with the lowest priority level will be serviced if higher priority subsystems make frequent requests. The daisy chain arbitration and parallel arbitration methods may need to be modified or a controller may need to be incorporated to ensure each subsystem can be serviced fairly.

The VMEbus uses a serial-parallel combination for bus arbitration with only one bus arbiter. VMEbus arbitration uses a scheme with four parallel priority levels similar to Figure 2.5. Each priority level, however, can have subsystems daisy-chained as illustrated in Figure 2.4. In other words, the bus arbiter grants bus access to a given level and then the daisy chain at that level determines which subsystem actually gets the bus.

The VMEbus arbitration process includes the BBSY signal (as shown in Figure 2.4) and the bus clear (BCLR) signal. The BBSY and BCLR lines are added to the bus arbiter and all subsystems on the VMEbus. The VMEbus BBSY signal is asserted by the subsystem which is granted bus access. The BCLR output signal informs all subsystems on all priority levels that a subsystem on a higher priority level than the current bus master has requested access to the VMEbus. As mentioned earlier, the requesting subsystem should accommodate a "release when done" or "release on request" strategy to resolve pending higher priority requests for bus access.

B. MEMORY-MANAGEMENT

Memory-management can employ a combination of methods to organize the physical memory associated with a microprocessor or system. These methods effectively free the programmer using the system, from being concerned where the program code and program data will reside in memory. This thesis addresses the memory-management concepts of memory protection, virtual-memory and dual-ported memory.

1. Memory Protection

One method used to organize the address range of a microprocessor is to divide its address space into two or more blocks. Each block of the address space can be designated for a specific purpose, such as supervisor memory or user memory.

The MC68010 microprocessor has two modes of operation. These modes are the user mode and the supervisor mode. The user mode provides an instruction set for the programmer to accommodate a majority of applications. The supervisor mode provides additional instructions and privileges for use by the operating system and other system-related software [Ref. 7:p. 1-1].

The user memory is the area designated for non-privileged individuals to use. Such an individual executes programs in the user mode. The address range for the user is normally limited because it does not include the addresses associated with the operating system and the memory-mapped peripherals. Additionally, the user is restricted from executing privileged supervisor instructions. In contrast, the operating system executes programs

in supervisor mode and can address supervisory memory and memory-mapped peripherals as well as user memory. This segregation of the supervisor and the user precludes the user from reconfiguring the system, but still allows the user access to part of the physical memory and to the computational power of the microprocessor. Typically, the user must request the operating system to perform operations which the user is not allowed to perform.

2. Virtual-Memory

Virtual-memory allows programs to be executed which require more memory space than is physically resident. Therefore, the maximum program size is not limited by the size of physical memory. Originally, this method was designed to reduce and more effectively use memory.

A virtual address is an address located within the address space of the microprocessor. Consequently, with the MC68010 microprocessor, there exists 16 megabytes of virtual-memory. A virtual-memory implementation groups the virtual addresses into blocks called pages. Figure 2.6 shows such a grouping with zero through N pages of virtual-memory but with only enough physical memory to accommodate two virtual pages in physical memory. In Figure 2.6, virtual PAGE 1 and virtual PAGE N are mapped into separate physical pages.

When the CPU generates a virtual address, the virtual address is translated into a physical address. The address translation process includes fairly sophisticated memory protection so that tasks cannot interfere with each other or access resources

not allocated to them. Figure 2.7 illustrates a simplified memory-mapping mechanism. The high order virtual address bits are referred to as a virtual page number. The virtual page number references a location of the translation table. The translation table has as its contents a physical page number which references the starting location of the physical memory's page address. The low order virtual address bits give the relative address offset of the desired address within the physical page selected.

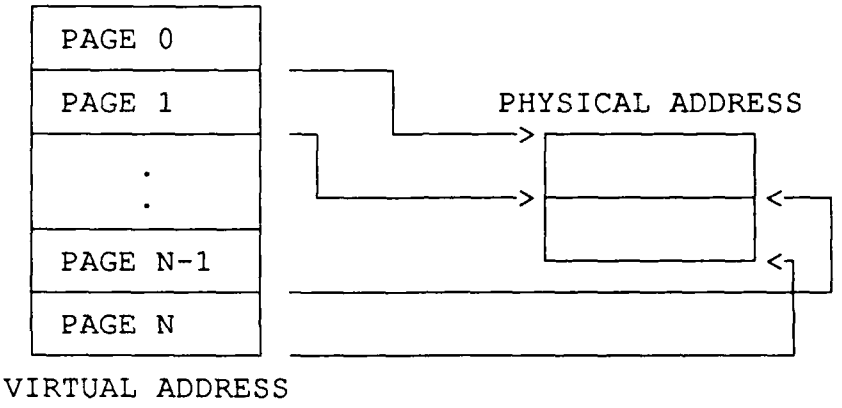


Figure 2.6: Virtual-Memory-Mapping

Generally, each processing task has its own translation table similar to Figure 2.7. These tables are switched whenever the active task changes which avoids interference between processing tasks.

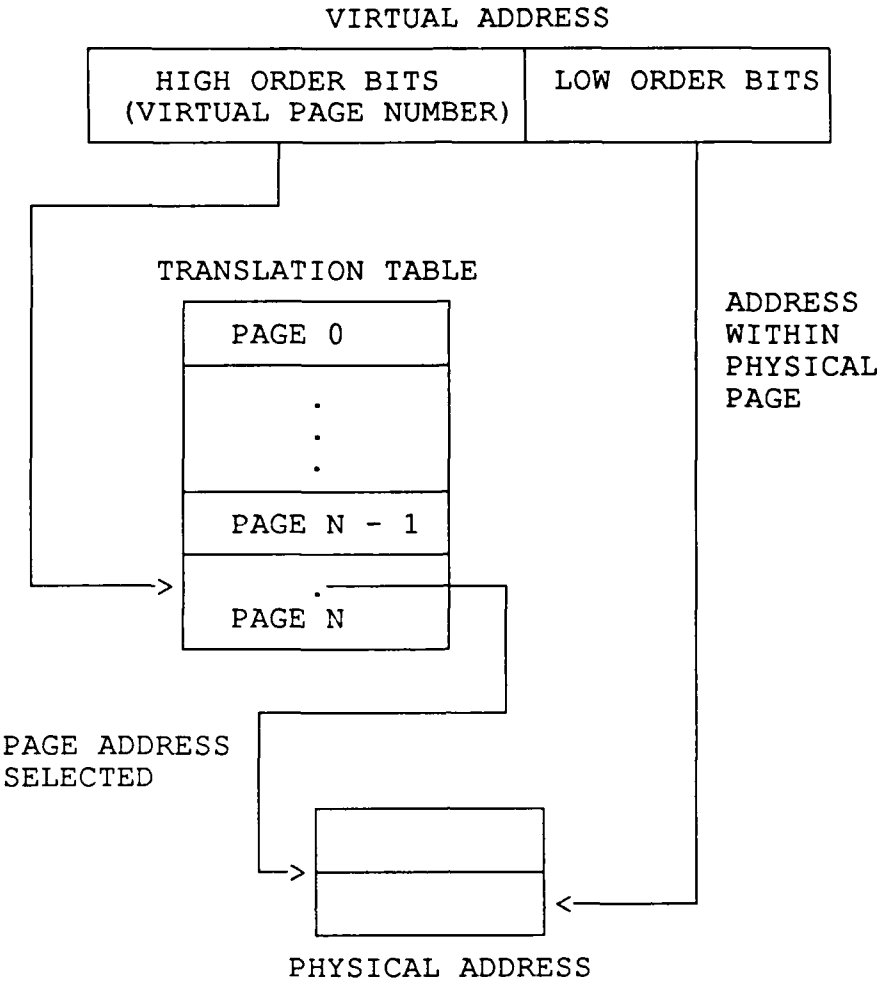


Figure 2.7: Mapping Mechanism

When the CPU generates a virtual address in a page that is not present in physical memory, for instance PAGE 2 as in Figure 2.7, the memory manager senses that fact and generates a page fault. The page fault triggers a chain of events which ultimately retrieves the desired page of the program from secondary storage and places it in physical memory. The instruction which caused the page fault is then continued or restarted. [Ref. 2:pp. 326-330]

3. Dual-ported Memory

Dual-ported memory permits two nearly simultaneous accesses to the memory resource without conflict. Figure 2.8 illustrates a typical configuration of a dual-port memory device. One approach to arbitrating concurrent memory requests in a dual-ported random access memory (RAM) is to sample one request line on the rising clock edge and the other on the falling clock edge. A PORT 1 REQUEST is assumed to be sampled on the rising clock edge.

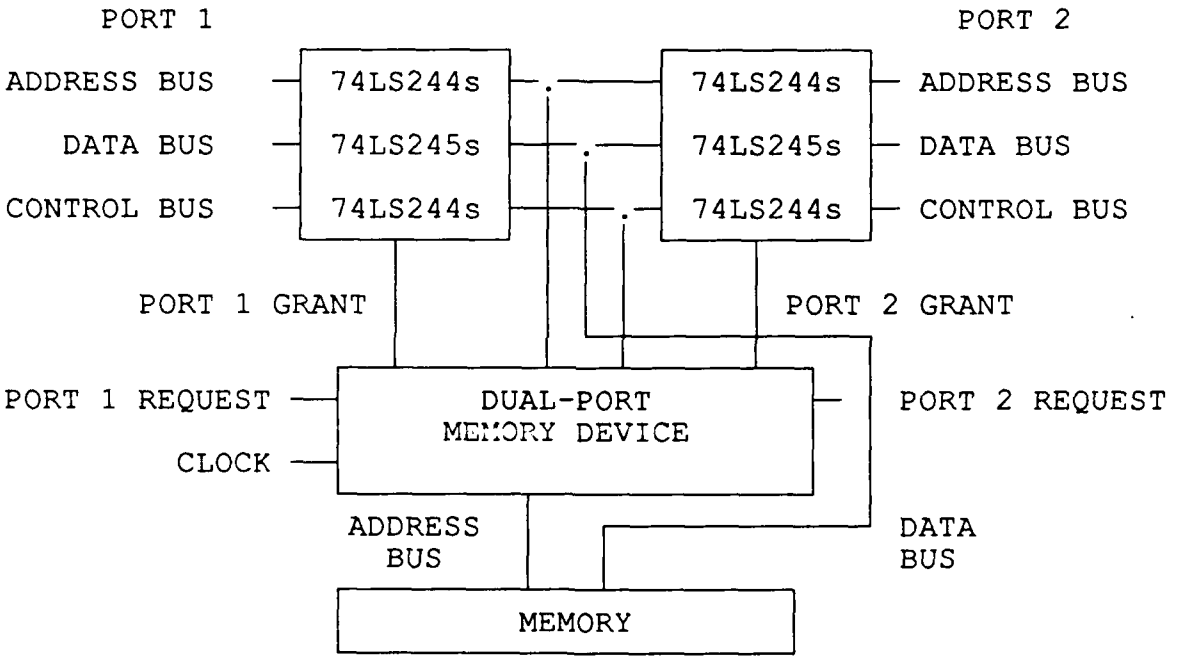


Figure 2.8: Dual-ported Memory

If a PORT 1 REQUEST is asserted, a PORT 1 GRANT is generated which gates the PORT 1 address, data and control lines through the left-hand 74LS244s and 74LS245s in Figure 2.8. The address and control signals are sent to the dual-port memory device and the data

signals are sent directly to memory. The dual-port memory device then gates the address lines to memory. While the PORT 1 GRANT is active, the PORT 2 GRANT cannot be asserted. PORT 2 is thus locked out from gaining access to memory. In contrast, if a PORT 2 REQUEST is asserted and PORT 1 is inactive, a PORT 2 GRANT is generated. This causes PORT 2 to gate the control and address lines through the other 74LS244s to the dual-port memory device and to gate the data lines directly to memory via the 74LS245s.

In the event that both request lines are active, a PORT 1 GRANT will be generated on the rising clock edge or a PORT 2 GRANT will be generated on the falling clock edge. The other request is locked out until the request line of the recognized port is no longer asserted. The other port will then gain access on the appropriate clock edge.

III. SYSTEM OVERVIEW

This thesis seeks to design a system that satisfies the design requirements for a system that can be expanded to a multi-processor system. Additionally, the subsystem design is interrupt-controlled with both virtual-memory and dual-ported memory support. This chapter gives a system perspective on the hardware associated with the system controller circuit board and master circuit board (Fig. 3.1) integrated to the VMEbus.

A. SYSTEM CONTROLLER CIRCUIT BOARD

The VMEbus specification describes the system controller as a board which resides in slot one of the VMEbus back plane [Ref. 4: pp. 5]. The system controller circuit board design provides priority bus access arbitration, a manual system reset and a interrupt acknowledge (IACK*) daisy chain driver. The system controller subsystem uses line drivers to buffer the arbitration signals and IACK* signal on the VMEbus.

1. Priority Bus Arbitration

The Motorola MC68452 bus arbitration module (BAM) peripheral device [Ref. 8] was selected to perform the VMEbus access arbitration. The BAM is configured to accommodate four bus request (BRx*) inputs and four bus grant (BGx*) outputs. After parallel arbitration, a bus grant signal is generated by the BAM at the level of the highest priority bus request. The bus grant signal is then daisy chained down on the level of the highest

priority bus request. This VMEbus arbitration method combines the advantages of both the daisy chain arbitration and parallel arbitration methods discussed in Chapter II.

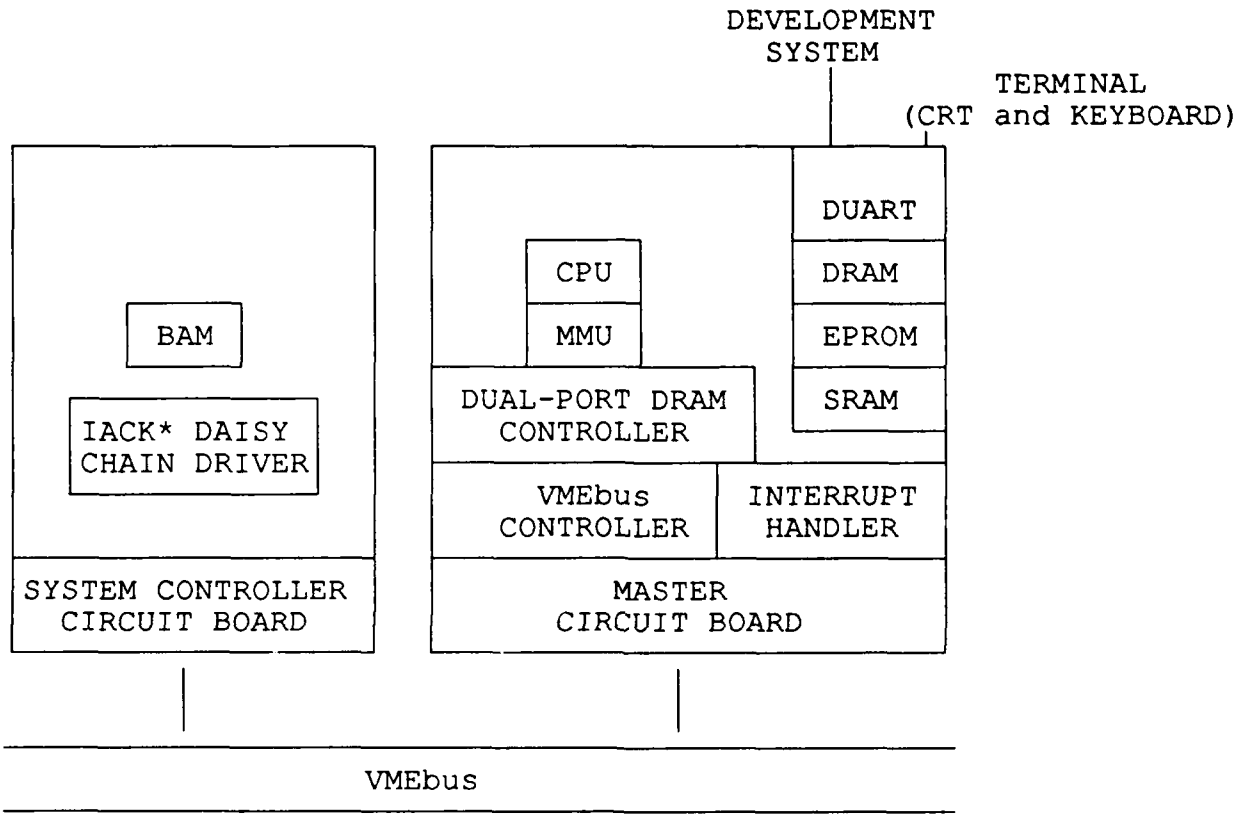


Figure 3.1: System Block Diagram

2. Manual Reset

The manual system reset provides a system-wide master reset of all devices within all subsystems. Resetting the system re-initializes various devices within it. This is necessary in order to restart the system after system failure.

3. Interrupt Driver

The VMEbus structure provides the IACK* signal daisy chain. However, a driver is provided on the system controller circuit board to drive the IACK* signal onto the VMEbus.

B. MASTER CIRCUIT BOARD

The master circuit board is the primary design focus of this thesis. As shown in Figure 3.1, the master circuit board subsystem is composed of nine functional blocks. These functional blocks are the central processor unit (CPU), dual universal asynchronous receiver/transmitter (DUART), dynamic random access memory (DRAM), static random access memory (SRAM), erasable programmable read-only memory (EPROM), memory management unit (MMU), dual-port DRAM controller, VMEbus controller and interrupt handler. The master circuit board is configured in a master-only role as discussed in Chapter II.

1. Central Processor Unit

The Motorola MC68010, 16-bit CPU, was selected to be the processing element because it has the necessary features to support virtual-memory but lacks the added complexity of a 32-bit architecture. It also affords easier wire-wrap assembly than the other Motorola CPUs supporting virtual-memory because wire-wrap is better supported for a dual in-line package (DIP) and there are fewer data and address signals. The signals and programming capabilities of the MC68010 microprocessor are discussed in further detail in Appendix A.

2. Dual Universal Asynchronous Receiver/Transmitter

Two asynchronous serial (RS-232) ports are implemented with the Motorola MC68681 DUART. One serial port is configured to drive a terminal, while the second serial port is used to down-load files from an IBM XT/AT compatible computer. The first serial port is used to permit a human interface to the system. The intent of the second serial port is to provide the ability to develop software on an IBM XT/AT compatible computer with a cross assembler and then to down-load the software through the second serial port to the master circuit board's random access memory (RAM) for testing, debugging and execution.

3. Erasable Programmable Read-Only Memory

The EPROM in this thesis design, contains the exception vector table and the monitor/debugger program. The exception vector table contains the addresses of the routines to be executed as a result of an interrupt or other exception. The monitor program configures the subsystem when it is powered up and handles communications with the terminal for interaction between the microprocessor and the user. It also provides debugging commands and coordinates the previously mentioned down-loading of files. Sixty-four kilobytes of EPROM are provided in the master circuit board.

Once an operating system is developed, it would not be desirable to freeze the interrupt part of the exception vector table into read-only memory (ROM). It should be noted that the

design of an operating system to take advantage of the system's hardware features is beyond the scope of this thesis.

4. Random Access Memory

Sixteen kilobytes of SRAM and one megabyte of DRAM are provided on the master circuit board.

5. Memory Management Unit

The use of the Motorola MC68451 MMU affords several advantages to the microprocessor system. The MMU provides the advantages of virtual-memory and a sophisticated memory protection scheme (both previously discussed in Chapters I and II). The MC68451 provides the capability to:

- Translate logical addresses to physical addresses.
- Provide segment descriptors to implement memory protection.
- Detect page faults and other situations requiring operating system intervention.
- Aid the operating system in managing the virtual-memory system efficiently (by use of the segment status registers).

6. Dual-port DRAM Controller

The Signetics 74F765 dual-port DRAM controller provides access to the DRAM by either a local bus master or a global bus master. If DRAM is accessed by the local bus master, i.e., the CPU on the master circuit board subsystem, it becomes a local asset. It is not desirable for the local CPU to access DRAM via the VMEbus because long access times would be the result. If DRAM is accessed by a global bus master, i.e., another subsystem controlling the VMEbus, it becomes a global asset. The ability to access DRAM locally or globally is desirable for a system that includes

subsystems that interact closely with one another. In addition, the dual-port DRAM controller provides refresh cycles to the dynamic memory integrated circuit chips.

The global memory accesses in this master circuit board subsystem design, use physical addresses to permit the implementation of mailboxes with attached semaphores as discussed in Chapter I. An operating system needs to lock the mailbox page in physical memory at a specified physical address.

7. VMEbus Controller

The Signetics SCB68172 VMEbus controller preserves the VMEbus data transfer and VMEbus access protocols. The VMEbus controller and the MC68010 CPU are configured in a master-only role as illustrated in Figure 2.2 and discussed in Chapter II. The VMEbus controller provides the necessary logic to interface the master circuit board subsystem to the VMEbus.

8. Interrupt Handler

The Signetics SCB68155 interrupt handler is used in the master subsystem design to assist the CPU with interrupt processing. The interrupt handler receives global and local interrupt requests and arbitrates their priority. The arbitration priority is non-maskable interrupts, first, then local interrupts and finally global interrupts.

The interrupt handler acts as a mediator between the CPU and the interrupting device or between the CPU and the interrupting subsystem. Once a local interrupt is generated by the DUART or MMU, control signals are sent between the interrupting device and

the interrupt handler as well as between the interrupt handler and the CPU. The DUART or the MMU responds with a pre-programmed status/ID vector as an interrupt response.

A subsystem can request an interrupt at any time by asserting the appropriate interrupt request line. On detecting an interrupt request, the interrupt handler sends a control signal to the VMEbus controller to request the VMEbus during the interrupt acknowledge cycle. The subsystem making the request then sends the status/ID vector to the master circuit board's CPU.

IV. DESIGN IMPLEMENTATION

This chapter discusses the design of the minimal system and of the fully integrated system (master circuit board and system controller circuit board). The minimal system provides the foundation of core resources necessary to construct a computer system. The fully integrated system design can be implemented by integrating additional resources to the minimal system. For comparison, the fully integrated system is illustrated in Figure 3.1, while the minimal system is illustrated in Figure 4.1.

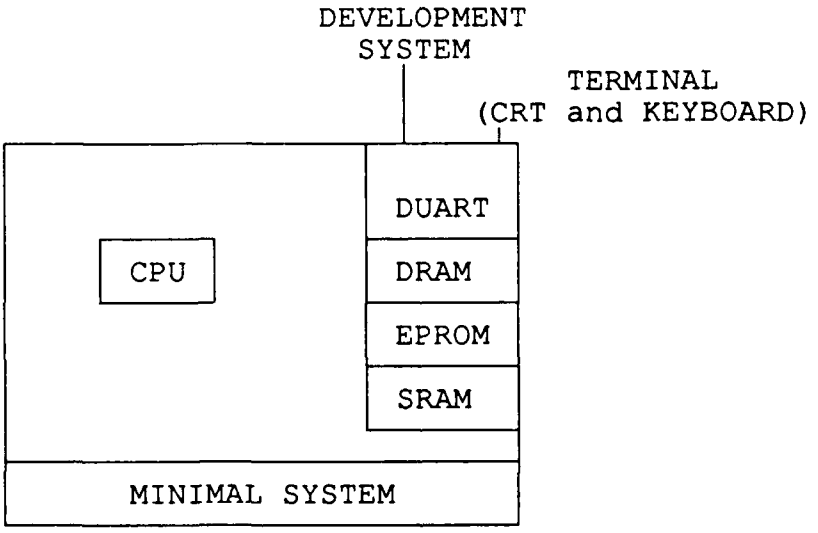


Figure 4.1: Minimal System

A. MINIMAL SYSTEM

Currently at the Naval Postgraduate School (NPS), there exists no computer-aided design (CAD) tools which can simulate the fully integrated system designed in this thesis. This is in part due to the inability of the CAD vendors to keep pace with the profusion of extremely complex very large scale integrated (VLSI) circuit chips. The CAD systems at NPS, Valid Inc.'s SCALD and Futurenet's CAD50, do not support all the peripheral devices incorporated within this thesis. Consequently, a step-by-step progression was made to fully integrate the system. The first stage, referred to as the minimal system, includes the core resources which form the foundation to which more complex devices can be added. When more complexity is added to the minimal system, operational testing can be conducted to insure proper integration of the new devices into the system.

1. Memory Map

Memory-mapping determines how the microprocessor accesses physical memory and peripheral devices. The Motorola MC68010 microprocessor has 23 address lines, A1 through A23. The upper data strobe (UDS*) and lower data strobe (LDS*) lines collectively determine address bit A0. Effectively, there are 24 address lines giving an virtual address range of 16 megabytes. Physical memory elements such as static random access memory (SRAM), dynamic random access memory (DRAM) and read-only memory (ROM) are mapped into this 16 megabyte range as are the memory-mapped peripherals.

The memory-mapped peripheral devices have multiple internal registers. The high order physical address bits are used to select a particular peripheral device. The low order physical address bits are decoded inside the peripheral device and subsequently select one of the internal registers. These registers are programmed to configure the device to meet desired performance specifications.

Table I displays the specific locations of the minimal system's memory-mapped devices and the physical memory components within the address space of the MC68010 central processor unit (CPU).

TABLE I: MINIMAL SYSTEM MEMORY MAP

PHYSICAL ADDRESS	
$000000	
	64K BYTES OF EPROM
$00FFFF	
$010000	
	16K BYTES OF STATIC RAM
$013FFF	
$014000	
	NOT USED
$7F6FFF	
$7F7000	
	MC68681 DUART
$7F7FFF	
$7F8000	
	NOT USED
$FFFFFF	

The 64k bytes of erasable programmable read-only memory (EPROM) contain the exception vector table and the monitor/debugger program. Appendix B gives the source code listing of the exception

vector table and the monitor/debugger program. The 2500AD MC68010 cross assembler [Ref. 9], running on an IBM XT/AT compatible computer, was used to cross assemble the monitor/debugger source code into a Motorola S-record format [Ref. 10:pp. A-1 - A-4]. In order to program the S-record code into the EPROM, a Data I/O System 29 Universal Programmer was configured to accept Motorola S-records. The S-record file was then sent from the IBM XT/AT to the Data I/O System 29 via an RS-232 interface. Finally, the EPROM programming process was initiated on the Data I/O System 29.

The 16K bytes of SRAM are used to test development software. Files can be down-loaded to the SRAM for debugging. SRAM is used in the minimal system design instead of DRAM to avoid the additional logic necessary to generate refresh cycles for the DRAM.

The MC68681 dual universal asynchronous receiver/ transmitter (DUART) is a communications peripheral device that can accommodate two independent full-duplex (receiver/transmitter) ports. The operating mode and data format of each port can be programmed independently. One port of the DUART is configured by the monitor/debugger program to accommodate the down-loading of files from an IBM XT/AT compatible computer. The other port of the DUART is configured to communicate with the terminal. The memory map (Table I) delineates a physical address range of $7F7000 through $7F7FFF for the DUART. A chip select signal will be generated for the DUART when a physical address is in the range $7F7000 through $7F7FFF. The physical addresses in the range

$7F7010 through $7F7FFF are multiple maps for the DUART. Multiple maps provide valid addresses to chip select the DUART. They also permit address decoding logic to be simplified. However, to avoid ambiguity, only the physical addresses $7F7000 through $7F700F are used to address the DUART.

2. Hardware Interface

Appendix C illustrates the circuitry involved in the minimal system. Figures C.1 through C.8 illustrate the minimum system in its entirety.

Figure C.1 illustrates the MC68010 microprocessor used in the minimal system design.

Figure C.2 illustrates the HALT* and RESET* generation circuitry. The NE555 timer provides an automatic system reset when the system is powered up. There is also a manual system reset switch (push button). Resetting the system initializes the internal circuitry of the CPU and DUART. A two-input OR gate in the reset circuitry has one input grounded, so it acts as an unneeded buffer. However, in the fully integrated system (discussed later in this chapter), this input is tied to the VMEbus system reset (SYSRESET*) line. This permits a system-wide reset to the master circuit board illustrated in Figure 3.1.

Figure C.3 illustrates the clock generation circuitry. The 8 MHz CPU clock signal is produced by using a 74LS161 binary counter to divide a 16 MHz signal from a crystal controlled oscillator. A 4 MHz signal from the 74LS161 provides the clock

input for the shift register which is used to help generate the data transfer acknowledge (DTACK*) and bus error (BERR*) signals.

Erasable programmable logic devices (EPLDs), specifically Altera EP310s, were used to reduce the chip count in the minimal system. EPLDs were used for address decoding, generating DTACK and BERR signals, performing interrupt control and generating SRAM write enable and RAM and ROM output enables.

Figure C.4 shows the EPLD implementation for the minimal system address decoder. The minimal system address decoder implements the memory map of Table I. Listing D.1 in Appendix D presents the Abel software program for the address decoder. Abel software will be discussed in the next section.

Figure C.5 shows the logic of the circuitry which generates the DTACK* and BERR* signals to the CPU. The circuitry prior to the 74LS05 open collector inverters, is implemented by an EPLD. The DTACK and BERR signals are passed through the 74LS05s to give the open collector outputs and the proper assertion levels (DTACK* and BERR*). In the event that the MC68010 microprocessor tries to address a location not supported by the design, a bus error (BERR*) time-out signal is generated after two microseconds. The BERR* signal causes the CPU to begin bus error exception processing. This invokes the routine whose address is in the longword at address $000008. The circuit which generates the delay time for BERR* is referred as a watchdog timer. Listing D.2 in Appendix D presents the Abel description of the DTACK and BERR signals.

The circuitry for EPROM and SRAM is illustrated in Figure C.6. Since random access memory (RAM) and ROM cannot generate a DTACK* signal to the CPU, additional circuitry is required. The DTACK* signal informs the CPU that the data transfer has been completed by the slave device. The 74LS164 shift register generates the data transfer delay times for the RAM and the ROM and the bus time-out delay for a bus error condition (Fig. C.5). A 250 nanosecond delay is provided to ensure an adequate time for data transfer between the CPU and the RAM. A 500 nanosecond delay is provided for data transfer between the CPU and the ROM. These transfer times accommodate the data propagation delay, the system address decoding delay and the internal address decoding delay of the RAM and the ROM. The logic for the output enable and the write enable signals are implemented on an EPLD. Listing D.3 in Appendix D presents the Abel description of the SRAM write enable and RAM and ROM output enable signals.

Figure C.7 shows the logic for the interrupt priority level (IPL0* through IPL2*) and the interrupt acknowledge (IACK681*) signal. A level one interrupt request (HHL) is sent to the MC68010 CPU when the MC68681 DUART asserts its interrupt request output (low). An IACK681* signal is sent to the DUART when a level one interrupt acknowledge is output by the CPU. The logic for the IACK681* and the IPL0* through IPL2* signals are actually implemented with an EPLD. Listing D.4 in Appendix D presents the Abel description of the IACK681* and IPL0* through IPL2* signals.

Figure C.8 illustrates the circuitry which supports the dual serial ports. As mentioned earlier, one port (Port A) of the DUART is configured to communicate with the terminal. The other port (Port B) is configured by the monitor/debugger program to accommodate the down-loading of files from an IBM XT/AT compatible computer.

3. Software Support

a. Exception Vector Table and Monitor/Debugger Program

The exception vector table contains the addresses of routines to be executed when an exception (trap or interrupt) is detected. The monitor program sets up communications with the terminal, provides debugging commands as well as a down-load command. The exception vector table and the monitor/debugger program (Appendix B) reside in the EPROM starting at physical address $000000. The exception vector table occupies physical addresses $000000 through $0003FF [Ref. 7:p. 4-5]. Physical addresses $000400 through $001FFF are not used and the monitor/debugger program begins at the arbitrarily selected physical address $002000.

The monitor/debugger program was developed on the Motorola Educational Computer Board (ECB) [Ref. 10]. After a system reset, the microprocessor's program counter is initially loaded with address $002000 to start the monitor/debugger program.

b. Monitor/Debugger Commands

The monitor/debugger program provides a user with six commands. These commands are not intended to be comprehensive, but

they do provide assistance in program development and debugging. The user commands are as follows:

- GO address <,break point address>
- MM start address <,end address>
- MD start address <,end address>
- RCH {Axx, Dxx, PC, US, SP, SR}
- REG
- LOAD

where <...> implies optional
{...} implies select one entry

The GO command is used to execute a program that resides in the system's memory. The program can be placed in memory by using the memory modify command or by down-loading a program from an IBM XT/AT compatible computer. The address in the GO command gives the location where program execution will begin. An optional break point address can be added within the GO command. The break point will stop program execution at the address specified. This is particularly useful if one desires to know the state of the machine, i.e., memory contents or register contents, at that point.

The memory modify command (MM) is used to modify the contents of an address or, if desired, a range of addresses. This command can modify code or data residing in RAM.

The memory display command (MD) is used to display the contents of an address or a range of addresses, if desired.

The change register command (RCH) is used to modify the contents of an address register (Axx), a data register (Dxx), the program counter (PC), the user stack pointer (US), the system stack

pointer (SP) or the status register (SR). One of these options must be specified with the RCH command.

The display register command (REG) displays the contents of the address registers, data registers, program counter, user stack pointer, system stack pointer and status register. This information gives the state of the MC68010. This command is particularly useful when a breakpoint is reached in the debugging process.

The down-load command (LOAD) permits the minimal system to receive software that was developed on an IBM XT/AT compatible computer. After code has been assembled and linked using software such as the 2500AD MC68010 cross assembler, it can be down-loaded to the absolute address (or addresses) specified during the linking process.

c. Programmable Logic Device Programming

As already mentioned, EPLDs are used to reduce the chip count on the printed circuit board. The Data I/O Abel [Ref. 11] program was used to compile a high-level language representation of desired digital logic. The output of Abel is a joint electron device engineering council (JEDEC) standard file for programming the EPLDs. This file is then down-loaded to the Data I/O System 29 Universal Programmer to program the EPLDs. Appendix D shows the Abel source code that generates the logic implementations discussed in this chapter and illustrated in Figures C.4, C.5, C.6 and C.7.

B. FULLY INTEGRATED SYSTEM

The intent of this thesis is to design a hardware system so that at some future date an operating system could be developed to control its hardware facilities. These facilities accommodate virtual-memory, protected memory, serial communications, interrupt control and multi-processor abilities interfaced to the VMEbus. A hard disk controlled by a direct memory access (DMA) controller would be needed to implement the paging function required to support virtual-memory. The operating system would use the memory management unit (MMU) to implement user/supervisor memory allocations (protected memory) and virtual-memory. Considerations for a future operating system will be discussed throughout the following sections.

The fully integrated system is composed of the master circuit board subsystem and the system controller subsystem (Fig. 3.1). Each subsystem is decomposed into functional units. The functional units for the master circuit board subsystem are shown in Figure E.1 and the functional units for the system controller subsystem are shown in Figure E.2. Each of the functional units for the subsystems is discussed in the following sections.

1. Memory Map

The memory map (Table II) of the master circuit board's physical address space contains the memory-mapped peripheral devices and the physical memory. This mapping is an enhanced version of the minimal system's physical memory map (Table I).

TABLE II: SYSTEM MEMORY MAP

PHYSICAL ADDRESS	
$000000	
	64K BYTES OF EPROM
$00FFFF	
$010000	
	16K BYTES OF SRAM
$013FFF	
$014000	
	OFF-BOARD RESOURCE
$7F4FFF	
$7F5000	
	MC68451 MMU
$7F5FFF	
$7F6000	
	SCB68155 INTERRUPT HANDLER
$7F6FFF	
$7F7000	
	MC68681 DUART
$7F7FFF	
$7F8000	
	OFF-BOARD RESOURCE
$7FFFFF	
$800000	
	ONE MEGABYTE OF DRAM
$8FFFFF	
$900000	
	OFF-BOARD RESOURCE
$FFFFFF	

The memory map allocates 64K bytes of ROM to include the interrupt vector table, monitor/debugger program and operating system. The interrupt vector table and monitor/debugger program perform the same roles as described in the minimal system. However, an operating system would have to be incorporated to handle the enormous code requirements to manage user/supervisor memory allocations (protected memory), page faults (for virtual-memory) and an operating system kernel. The intent is for the core of the operating system to reside in ROM, since a mass storage

device is not incorporated in this subsystem design. A design of a multi-disk control module for a VMEbus-based system was presented in an earlier thesis [Ref. 12].

The 16k bytes of SRAM retains upward compatibility with the minimal system. The SRAM will be used until the DRAM can be incorporated into the master circuit board subsystem. However, if an operating system requires more that the 64K byte size of ROM, which is a likely possibility, any range spanning the physical addresses $010000 through $7F4FFF could be allocated for more ROM or RAM. This would require changing the address decoding logic and adding ROM or RAM chips to the master circuit board subsystem design.

The MC68451 MMU [Ref. 13] is memory-mapped because its internal registers must be programmed for the desired virtual-memory configuration and address translation. By using the MC68010's function codes (see Appendix A) along with the desired address translation scheme, an operating system can separate the supervisor's address space from the user's address space, thus implementing a memory protection scheme.

The SCB68155 interrupt handler hardware [Ref. 14:pp. 2-369 - 2-385] is memory-mapped so that it can be initialized for the desired mode of operation. The interrupt handler can accommodate local interrupts from the DUART and the MMU as well as interrupts from global bus masters.

The MC68681 DUART [Ref. 15] provides the interface to two RS-232 serial links. One link is used for communications with the

terminal, while the other link is used for communications with an IBM XT/AT computer. The DUART is configured to provide the desired serial communications characteristics such as baud rate, parity and stop bits.

One megabyte of DRAM is provided for the master circuit board subsystem. The operating system would manage this resource by assigning virtual pages to physical memory. It is intended that a portion of the DRAM's physical address range map to the same virtual address range. This will permit global memory access to pass semaphores and messages between the master circuit board and other subsystems, as discussed in Chapter I.

It is important to note that if an address falls into the ranges of $014000 through $7F4FFF, $7F8000 through $7FFFFF or $900000 through $FFFFFF, the CPU is accessing an off-board device.

2. Master Circuit Board

a. Microprocessor

The MC68010 CPU (Fig. E.3) is the processing element of the master circuit board subsystem. The signals of the CPU can be organized into functional groups (see Appendix A) which describe the role of the signals within the subsystem.

The CPU has two bi-directional open collector pins, HALT* and RESET*, which require pull-up resistors to ensure that the signals are not asserted until the appropriate events occur.

The only bus master on the subsystem is the MC68010. Hence, the bus request (BR*) and the bus grant acknowledge (BGACK*)

signals require a pull-up resistor to ensure that the CPU does not perform bus arbitration.

No Motorola M6800 peripherals are used in the master circuit board design. Hence, the valid peripheral address (VPA*) signal is tied to a logical one.

The circuitry to generate the DTACK* and BERR* signals (discussed later) are open collector signals. Hence, pull-up resistors are used to ensure that these signals are not inappropriately asserted.

b. Halt and Reset Generation

The HALT* and RESET* generation circuitry (Fig. E.4) provides manual and automatic power-on subsystem reset to the CPU and peripheral devices. The NE555 timer provides an automatic power-on reset to the subsystem. The NE555 timer is configured as a one-shot to generate the power-on reset signal. This automatic reset occurs within the first few tenths of a second after the subsystem is powered on. An external system reset can also reset the subsystem. This system reset is generated from the system controller subsystem via the VMEbus. A debounced switch is used to cause a manual reset of the subsystem.

A reset causes the CPU to read into the SP register and PC register the longword (32-bits) contents of physical addresses $000000 and $000004, respectively. Recall that ROM begins at physical address $000000. Consequently, the two longwords beginning at physical address $000000 are retrieved from non-volatile memory. The initial PC vector at physical address $000004 contains the

value $002000, so when this value is read into the PC, execution of the monitor/debugger program is started.

c. Clock Generation

The clock generation circuitry (Fig. E.5) provides clocking signals to the CPU and to the peripheral devices. A 74LS161 binary counter is used to divide the 16 MHz signal from the crystal oscillator into rates that accommodate the CPU, the MMU, the dual-port DRAM controller and the interrupt handler hardware. A 4 MHz signal is sent to additional circuitry to help generate the DTACK* and BERR* signals.

d. Local Bus Address Decoding

Once a virtual address is mapped to a physical address, the local bus address decode circuitry (Fig. E.6) is used to generate chip select signals for RAM, ROM or a peripheral device based upon the system memory map (Table II). Two Altera EP310 EPLDs [Ref. 16:pp. 2-57 - 2-62] were used in the design to be programmed via Abel software [Ref. 11]. As mentioned earlier, Abel is software developed by Data I/O Corporation that permits a high-level language description of the logic function to be programmed on a EPLD, programmable array logic (PAL) or similar logic device.

e. Memory Management Unit

The MMU circuitry (Figs. E.7 and E.8) provides the subsystem with virtual-memory support and memory protection. The address translation from a virtual-address-to-physical-address is done by this device. Once the MC68451 MMU has been configured by the operating system, the address translation is performed

internally within the MMU and is thus hidden from the subsystem unless a page fault occurs. The internal details of the MMU are given in its reference manual [Ref. 13].

A page fault (FAULT*) signal is generated if the MMU detects a write violation or if address translation cannot be performed successfully. The write violation occurs if an attempt is made to write to a write-protected portion of physical memory. If address translation cannot be performed, this denotes to the operating system that a new memory page may need to be brought into memory from a hard disk or that there is a system error. The operating system configures the MMU to write-protect memory segments and to implement virtual-memory-mapping by the MMU.

The circuitry to inhibit virtual-address-to-physical-address translation during an interrupt cycle is illustrated in Figure E.7. The mapped address strobe (MAS*) and ALL input signals to the MMU are generated during an interrupt acknowledge cycle.

The physical data strobe generation circuitry (Fig. E.8) is used to generate the physical upper data strobe (PUDS*) and the physical lower data strobe (PLDS*) signals. The PUDS* and PLDS* signals are generated during normal virtual-address-to-physical-address translation. Normal address translation is the mapping of a virtual address to a physical address without a page fault occurring. The physical data strobes will not be generated if there is a write cycle for a write-protected segment. This is accomplished by the write inhibit (WIN*) signal generated by the MMU.

The physical address strobe circuitry (Fig. E.8) generates a physical address strobe (PAS*) signal to denote that the address translation has taken place and the physical address is valid and stable.

f. Dual-port DRAM Controller

The dual-port DRAM controller circuitry (Figs. E.9, E.10 and E.11) provides two paths into RAM [Ref. 17]. The local bus master (the CPU) can be ported to the RAM or a global bus master can be ported to the RAM via the VMEbus. Two paths into RAM are especially useful because processor subsystems can pass information-carrying semaphores. Also, The 74F764 dual-port DRAM controller provides DRAM refresh.

The 3-state capability of the 74LS244s (Fig. E.9) octal-buffers and line drivers with 3-state outputs are used to isolate one port access to the dual-port DRAM controller from the other port. The port is selected by the appropriate clock edge and control signal to the request input (REQ1* or REQ2*) of the 74F764 dual-port DRAM controller.

The control signal for REQ1* of the 74F764 (CS764REQ1*) is generated by the local bus address decoder and the control signal for REQ2* of the 74F764 (CS764REQ2*) is generated by the VMEbus address decoder. If CS764REQ1* is active on a rising clock edge and SEL2* is not asserted, the local master is granted access to the 74F764. The dual-port DRAM controller then asserts SEL1* to enable the 74LS244s and 74LS245s on the local bus side.

If CS764REQ2* is active on a falling clock edge and SEL1* is not asserted, the global bus master is granted access into the 74F764. The dual-port DRAM controller then asserts SEL2* to enable the 74LS244s and 74LS245s for the global bus side. In each case, the select line is released after the request signal is no longer asserted.

If both request lines are asserted and neither select line is asserted, on the next (rising or falling) clock edge, the select signal will be generated for the appropriate port access. The request that is locked out cannot gain access to the dual-port DRAM controller until the other port has completed its task and is no longer asserting its request signal.

The 74LS245s octal-bus transceivers with 3-state outputs, illustrated in Figure E.10, are used to buffer the data signals. Data can be sent between the CPU and the VMEbus, between the CPU and the DRAM or between the DRAM and the VMEbus. The data enable signal (DATAEN*) enables data to flow between the CPU and the VMEbus. The select port one (SEL1*) signal enables data to flow between the CPU and the DRAM, while the SEL2* signal enables data to flow between the DRAM and the VMEbus. The data flow direction to the 74LS245s is controlled by the read/write (R/W*) signal during local DRAM accesses, while the global R/W* signal (GR/W*) controls the direction for global DRAM accesses. The data direction enable (DDEN) signal controls the data direction flow between the CPU and the VMEbus.

The 74F764 can only effectively accommodate 18 address lines. Consequently, additional logic illustrated in Figure E.11 must be incorporated to handle address bit A19, which is required to give access to the desired one megabyte of RAM.

When the row address strobe (RAS*) signal becomes inactive, the data transfer acknowledge output from the 74F764 (DTACK764) is asserted. The DTACK signal of the 74F764 signals that data has been transferred to or from memory.

g. Dynamic Random Access Memory

The dynamic random access memory circuitry (Figs. E.12, E.13, E.14 and E.15) provides one megabyte of DRAM for the master circuit board subsystem. The DRAM is divided into two 512k byte blocks. The odd bytes are stored in one 512k byte block (Figs. E.12 and E.13), while the even bytes are stored in the other 512k byte block (Figs. E.14 and E.15).

The DRAM receives refresh cycles from the dual-port DRAM controller. Although the 74F764 dual-port DRAM controller seizes control of the DRAM during refresh cycles, a bus arbitration process is not needed. An 8 MHz clock pulse (RCP) is divided by 64 to produce a refresh request internal to the 74F764. If no request signal (REQ1* or REQ2*) is asserted on the 74F764, a nine-bit counter internal to the 74F764 is incremented. The counter value which represents the row in memory to be refreshed is then placed on output lines MA0 through MA8 of the 74F764. The RAS* signal is then asserted for four clock cycles to refresh a row in memory.

Finally, the RAS* signal is released and the refresh cycle is complete.

h. EPROM and SRAM

The EPROM and SRAM circuitry (Fig. E.16) provide 64k bytes of ROM and 16k bytes of SRAM. The EPROM contains the resident exception vector table and the monitor/debugger program. The SRAM is upward compatible from the minimum system. If additional memory is required by a resident operating system, a modification to the local bus address decoding logic would permit the size of ROM or RAM to be increased.

i. Dual Serial Port

The MC68681 dual universal asynchronous receiver/ transmitter serial port circuitry (Fig. E.17) is used to provide serial communications with the terminal and the IBM XT/AT computer. Port A is dedicated to the terminal and Port B is dedicated to the IBM XT/AT computer. The 3.6864 MHz crystal is used to generate the baud rates for data transmission for both ports. The terminal provides an interface to the system for the user. The IBM XT/AT is used to down-load files into the master circuit board subsystem's memory.

j. Interrupt Handler

The interrupt handler circuitry (Fig. E.18) provides the necessary logic to accommodate interrupts from devices residing on the master circuit board subsystem and global devices residing on other subsystems. The SCB68155 interrupt handler can

accommodate six local interrupts, seven global interrupts and a non-maskable interrupt (NMI).

Local interrupts (LRQ1* through LRQ6*) have a higher precedence than the global interrupts (IRQ1* through IRQ7*). The local interrupt signal LRQ6* has the highest priority, while local interrupt signal LRQ1* has the lowest priority. The global interrupt signal IRQ7* has the highest priority, while global interrupt signal IRQ1* has the lowest priority. The NMI signal has priority over local and global interrupts and it is provided for a catastrophic occurrence such as an alternating current (AC) power failure.

Local interrupts are generated by the DUART and the MMU. The DUART is programmed to provide an interrupt request when a port buffer full condition is met. The buffer full condition of the MC68681 DUART occurs whenever a character is received from the terminal keyboard or from the IBM XT/AT. The local interrupt generated by the MC68451 MMU occurs when the interrupt bit of the page status register is set during normal address translation.

When a local or global interrupt occurs, the interrupt handler hardware generates an interrupt priority level output on lines IPL0* through IPL2* to the CPU. The CPU responds by acknowledging the interrupt with the interrupt acknowledge signal (IACK*) and places the interrupt level on address lines A1 through A3. The interrupt handler hardware reads the interrupt level on address lines A1 through A3 to determine which level is being acknowledged. If the interrupt was from a local device, the

interrupting device provides the vector number on the local data bus. If the interrupt was from another subsystem on the VMEbus, the interrupt handler hardware generates a bus interrupt acknowledge (BIACK*) signal to the VMEbus controller and the VMEbus. The VMEbus controller obtains control of the data transfer bus (DTB) so that an interrupt vector can be obtained from the interrupting subsystem. The BIACK* signal is only generated if the bus interrupt level is not masked (within the interrupt handler) and a local interrupt is not pending.

Once the local CPU has acknowledged the (local or global) interrupt request and has obtained an interrupt vector, the local CPU saves the state of the machine and transfers control to the appropriate interrupt handling routine. This prepares the CPU to perform an interrupt handling routine. After completion of the interrupt handling routine, the stored state of the machine is restored and the CPU resumes processing where it left off at the interrupt. [Ref. 7:pp. 4-3 - 4-16; Ref. 18:pp. 5-1 - 5-15]

k. Data Transfer Acknowledge and Bus Error Generation

The data transfer acknowledge and bus error generation circuitry (Fig. E.19) provides control signals to the CPU. This circuitry physically resides within a Altera EP310 EPLD. The DTACK* signal denotes that a data transfer has been completed by the slave device addressed. The MC68681 DUART, MC68451 MMU, SCB68172 VMEbus controller, SCB68155 interrupt handler and 74F764 dual-port DRAM controller peripheral devices possess the necessary

logic to generate their own DTACK* signal to acknowledge receipt or availability of data.

The master circuit board's RAM and ROM chips cannot generate their own DTACK* signals so external circuitry must do it for them. The DTACK* generation circuitry for the SRAM and ROM must allow adequate time for the data transfer. All these DTACK* signals are ORed together to produce the MC68010 DTACK* input.

If the CPU on the master circuit board makes an off-board access using the off-board (OFFBOARD*) signal to the VMEbus controller, the DTACK* signal (DTACK172*) is generated from the VMEbus controller. The off-board device provides a global DTACK* signal (GDTACK*) to the VMEbus controller (Fig. E.20) via the VMEbus DTACK* line. In turn, the VMEbus controller would provide the DTACK172* signal for the DTACK* circuitry. This arrangement permits long access times on the VMEbus.

If the master circuit board's DRAM is being accessed as a global asset, the GDTACK* signal is generated by the SEL2* and DTACK764 signals as illustrated in Figure E.11.

The BERR* signal is generated under one of three conditions. First, the BERR* signal is generated when the maximum allowable SRAM and ROM data transfer time has been reached and a DTACK* signal has not been received by the CPU. Secondly, a global bus error (BERR172*) signal can be received from a VMEbus watchdog timer if the master circuit board subsystem has control of the VMEbus. Finally, if a page fault signal (FAULT*) is generated by the MMU, this also causes a bus error condition.

The bus error condition causes exception processing to occur. The current state of the machine is saved. Information from the saved state of the machine can be used to determine the cause of the bus error. This is handled by the bus error exception routine as part of an operating system.

If the first port of the dual-port DRAM controller is not active and a refresh cycle is not taking place, a global bus master can have access to the DRAM. The master circuit board's CPU is unaware of the access to the DRAM through the second port. Consequently, the burden is placed upon a global master or a VMEbus watchdog timer to provide a global BERR* signal (GBERR*) on the VMEbus BERR* line, when appropriate, to the VMEbus controller. The GBERR* signal is sent to the BERR* circuitry (Fig. E.19) via the BERR172* signal.

1. VMEbus Controller

The VMEbus controller circuitry (Fig. E.20) provides the necessary logic for the master circuit board subsystem to gain access to the VMEbus. The SCB68172 VMEbus controller provides control signals (VMEEN*, DATAEN* and DDEN) to the master circuit board subsystem's drivers and transceivers. The purpose of the VMEbus enable (VMEEN*) signal is to enable the bus drivers only when there is an off-board (OFFBOARD*) access. In addition, the data flow (DATAEN*) and its direction (DDEN) are controlled. Parallel jacks are provided which permit jumper selection of the master circuit board subsystem's priority on the VMEbus.

m. VMEbus Address Decoding

The VMEbus address decode circuitry (Fig. E.21) permits access of a global bus master to the second port of the dual-port DRAM controller and ultimately into DRAM. Any subsystem, which has gained control of the VMEbus, has the ability to access the designated (by the operating system) area of DRAM for semaphore passing. The VMEbus address decoder provides the chip select signal CS764REQ2* to the dual-port DRAM controller (Fig. E.9). If the CS764REQ2* is asserted when clock edge falls and SEL1* signal of the 74F764 is not asserted, the isolation drivers are enabled to permit the flow of data and addresses from the global resource to the DRAM.

n. VMEbus Drivers

The circuitry for the master circuit board's VMEbus drivers (Figs. E.22, E.23 and E.24) provides control of signals from the local bus to the VMEbus and from the VMEbus to the local bus. The VMEbus controller controls the direction of the signal flow as requested by the CPU. Whenever the local bus master, the CPU, is not in control of the VMEbus, all signals from the local bus are isolated at the drivers by the VMEbus controller. Thus, in this case, no signals are gated onto the VMEbus from the local bus. However, another subsystem, if in control of the VMEbus, has direct access to the DRAM through the dual-port DRAM controller. The global addresses on the VMEbus fall into the range of the one megabyte of user DRAM in the master circuit board subsystem's memory map (Table II).

3. System Controller Circuit Board

a. Bus Arbiter

The VMEbus arbitration circuitry (Fig. E.25) provides the logic to arbitrate prioritized bus requests in parallel. Each bus request is then daisy chained down to the requesting device. Each subsystem capable of VMEbus access must have the ability to provide a bus request at one of four priority levels. The highest priority signal used is DBG7*, while the lowest priority level signal used is DBG4*. The process of resolving the VMEbus requests was described in Chapter II. Since the MC68452 bus arbitration module (BAM) [Ref. 8] is an asynchronous device, the bus grant signals (DBGx*) are not guaranteed to be spike-free. Consequently, a 50 nanosecond delay circuit is used to disable the DBGx* signals during the parallel arbitration process.

b. System Reset

The system reset circuitry (Fig. E.26) provides a system-wide master reset. This signal is sent on the VMEbus to all circuit boards and it is used to reset the entire system much like the local reset discussed earlier in this chapter.

c. VMEbus Drivers

The circuitry for the system controller drivers (Fig. E.27) provides the drive capability for signals to/from the VMEbus. Since circuitry was not designed to detect an AC power failure, the ACFAIL* signal is never asserted. This signal is input to the non-maskable interrupt of the interrupt handler (Fig. E.17). The bus clear (BCLR*) signal informs the current bus master that there is

a higher pending bus request. Burden is placed upon the current bus master to either relinquish control of the bus or to continue control until its task is completed. For the sake of simplicity, the master circuit board subsystem was designed to relinquish control upon the completion of its task. Finally, an IACK* daisy chain driver is provided for VMEbus interrupts.

V. RESULTS

Once the minimal system and fully integrated system hardware was designed, the schematic drawings drafted and the pin-out list implemented, software support was required to implement the minimal system. The monitor/debugger program required a thorough check of all its software features. These software features include the capability to set and remove a breakpoint, to display and modify memory, to display and change registers, to start program execution and to down-load software from a development system.

It was discovered while debugging the down-load portion of the monitor/debugger program that the 2500AD 68010 cross assembler's linking process incorrectly resolved external references. The lirking process generates a file in the Motorola S-record format. The problem was isolated only after comparing the Motorola S-record to Motorola's instruction format. It was identified that the 2500AD cross assembler was improperly resolving external references. A corrected version of the 2500AD cross assembler was obtained from the vendor that resolved this problem. With the monitor/debugger software developed, the minimal system design was complete.

The monitor/debugger and vector table were programmed in the erasable programmable read-only memory (EPROM) with the Data I/O System 29 Universal Programmer. The Data I/O System 29 segregated

the even bytes and odd bytes into separate EPROMs as required by the Motorola MC68010 central processor unit (CPU).

Erasable programmable logic devices (EPLDs) were used to reduce the chip count in the minimal system design. The minimal system used an EPLD to perform the interrupt request (IRQ681*) and the interrupt acknowledge (IACK681*) logic. Also, EPLDs were used to implement the circuit logic required for the generation of the data transfer acknowledge (DTACK) and the bus error (BERR) signals and for address decoding. In order to program the EPLDs, Abel software was used to compile the source code representation of the logic to be implemented with the EPLD. Once all of the source code for the EPLDs had been written, compiled and software tested, the EPLDs were programmed.

On the Data I/O System 29, once the EPLD is programmed, the test vectors are again tested against the programmed EPLD. During this test run, the System 29 failed for every EPLD that was programmed, even though they passed the software tests. On the advice of an applications engineer at Data I/O Corporation, the test vectors were removed from the source code. This code was compiled, then the EPLDs were programmed. The EPLDs were bread-boarded, while determining with reasonable certainty that the devices were actually implementing the desired logic.

The ultimate goal in this thesis was to implement the master circuit board subsystem design. One of the steps to achieve this goal requires the memory management unit (MMU) to translate a virtual address to a physical address. To avoid significant wiring

modifications to the minimal system to build up to the master subsystem, the MMU was wire-wrapped into the minimal system design. However, the MMU was not programmed at the minimal system stage. The MMU translates a virtual address to the same physical address when the MMU is not programmed after being reset. The MMU was configured to accommodate an automatic, manual and programmed (CPU reset instruction) reset.

VI. SUMMARY AND CONCLUSIONS

A. SUMMARY

The goal of this thesis was two-fold: first, to explore hardware ramifications of designing a microprocessor system for a multi-processor environment; and secondly, to implement the minimal system design.

1. Design Concepts

In exploring hardware ramifications, the scope was limited to features of the VMEbus structure, in memory-management and interrupt control. The memory-management features included memory protection, dual-ported memory and virtual-memory.

a. VMEbus Structure

The VMEbus permits an exchange of data and control beyond the boundaries of a single circuit board. Other subsystems or circuit boards which may include processing elements, memory and/or input/output (I/O) devices can be integrated to the VMEbus. A strict adherence to data transfer protocols over the VMEbus ensures the reliability and integrity of the system. The ability to integrate various subsystems along the VMEbus supports a multi-processor environment.

b. Memory-Management

The Motorola MC68010 central processor unit (CPU) generates function codes which can be used by the memory management unit (MMU) to partition memory into supervisor and user portions.

An operating system would manage memory partitioning. Normally, systems are designed so that the supervisor memory portion contains the memory-mapped I/O devices and the read-only memory (ROM) and some random access memory (RAM). The ROM is mapped to the supervisor portion of memory since it provides the exception vector table and start-up program.

The function codes reflect the CPU's two modes of operation, the supervisor and user. The supervisor mode is a privileged mode which permits access to all instructions and the full range of memory (supervisor and user memory). The user mode permits access to only user instructions and the user memory. Typically, in the user mode, permission must be granted through the operating system to use system resources. The separation of supervisor memory from user memory prevents the user from tampering with the system assets or gaining supervisor privileges.

Dual-ported memory permits two separate sources to access the same memory block and provides the refresh signals for the dynamic random access memory (DRAM). Dual-ported memory permits RAM to be used as a shared asset. It is especially useful when a portion of the physical RAM is dedicated to passing parameters between microprocessor subsystems. Dedicating a portion of RAM for parameters is analogous to a mailbox delivery system. The mail courier (subsystem 1) delivers mail (parameters) to the mailbox (RAM). The addressee (subsystem 2) picks up the mail (parameters) and responds as required. If appropriate, the occupant (subsystem 2) places mail (parameters) in the mailbox

(RAM) to be delivered (to subsystem 1). These parameters can be used in managing a multi-processor operating system.

A MC68010-based system typically has memory-mapped I/O devices, RAM and ROM. DRAM is added the master circuit board subsystem to supplement the minimal system's static random access memory (SRAM). The MC68010 CPU has a virtual address range of 16 megabytes. However, the physical RAM's size is usually considerably less than the size of the virtual address space. Virtual-memory is used to extend the range of programming beyond the range of physical RAM. An MMU is used to map virtual addresses into RAM physical addresses. Also, the MMU detects an attempt by the CPU to access a virtual-memory address which is not currently present in physical memory. When such an attempt is detected, the MMU generates a page fault. This page fault causes the page fault exception routine to be invoked. The exception routine reads a page of information from secondary storage into RAM. The MMU maps the virtual addresses associated with the page into addresses in the physical RAM. After completion of the exception routine, program execution resumes with the completion of the instruction that caused the page fault.

c. Interrupt Control

Using interrupts results in more effective use of the microprocessor because the microprocessor is not kept waiting for a device to respond. The devices requesting interrupts in this thesis are programmed to provide an interrupt vector number during an interrupt acknowledge cycle for local interrupts. The interrupt

vector number causes the address of the exception routine to be obtained from the exception vector table by the CPU so that it can be executed.

2. Design Implementation

a. Hardware Configurations

The recommended wiring configurations that accompanied the product specifications for the MMU, VMEbus controller, dual universal asynchronous receiver/transmitter (DUART), dual-port DRAM controller, interrupt handler hardware and bus arbitration module (BAM) greatly assisted in the designs of the minimal system, system controller subsystem and master circuit board subsystem. However, in order to integrate these components into a system, care was taken to ensure that the control signals were interfaced properly. Since no computer-aided design (CAD) tools existed at the Naval Postgraduate School (NPS) to fully simulate even the minimal system design, prototyping the minimal system was necessary. The minimal system has a foundation of core resources. The intent was to prove the system design by building up a master circuit board subsystem from the minimal system.

The system controller subsystem provides a bus arbiter, interrupt acknowledge (IACK*) daisy chain driver and system-wide reset. The bus arbiter determines bus ownership between subsystems that make bus requests and it grants bus ownership to the subsystem with the highest priority. An IACK* daisy chain driver sends the IACK* signal on to the bus during an interrupt acknowledge cycle.

The system reset is used to reset all devices on all subsystems after a system failure.

The master circuit board subsystem accommodates the VMEbus structure, virtual-memory-mapping facilities, a protected memory scheme, dual-ported memory and interrupt handling hardware. The master circuit board subsystem design is an extension of the minimal system and should not be implemented until the minimal system is operational. In the master circuit board subsystem, the VMEbus controller provides the necessary logic to meet the VMEbus specification for setting up the baseline bus structure. Drivers and transceivers are incorporated to meet the specified signal drive capability and isolation requirements.

b. Erasable Programmable Logic Devices

The erasable programmable logic device (EPLD) used in the minimal system's address decoding must be modified to include the additional memory-mapped devices of the master circuit board subsystem. The EPLD used for interrupt handling in the minimal system is replace by the interrupt handler hardware in the master circuit board subsystem design.

The master circuit board subsystem design is an upgraded version of the minimal system. A pin-out list for all wiring connections was developed in order to reduce wire-wrap errors, but it is not included as part of this thesis. The small scale integrated circuit (SSI) logic shown for the generation of the data transfer acknowledge (DTACK), bus error (BERR), physical upper data strobe (PUDS*), physical lower data strobe (PLDS*) and

physical address strobe (PAS*) signals was actually implemented with EPLDs to reduce the chip count.

B. CONCLUSIONS

Meeting all the goals set in this thesis made this thesis an ambitious undertaking. The major integrated circuit (IC) chips included the CPU, DUART, interrupt handler hardware, dual-port DRAM controller, MMU, VMEbus controller and BAM. These IC chips required an extensive study of product specification and application notes to understand the wiring configurations and programming of the devices. Study of the specification notes invoked support ideas in the design that required further investigation. These support ideas included DRAM memory refresh accommodations, driver characteristics, noise reduction and virtual-memory. Once each device was reasonably understood, the problem of integrating the devices into a single system remained. Care was exercised to ensure that control signals were properly integrated to the devices. Consequently, a major portion of this thesis was spent in the research and design process without the assistance of CAD tools.

The design and implementation work of this thesis spanned almost two years. A major problem encountered was the inability to simulate the system designs. Hence, the system's validity could only be verified by actual design implementation.

The design phase took a considerable length of time because the inter-relationships between the devices to support a multi-processor environment, dual-port memory, virtual-memory, memory

protection, dual serial ports and interrupt control features were not trivial. Some of these features should have been eliminated so that a simpler design could have been implemented. However, using the approach of building a complex subsystem from a minimal system is an important technique. For a growing number of new application IC chips, facilities to simulate designs using these chips do not yet exist. Thus, there is a strong need for advanced design tools and engineering practices to support complex designs.

An important restriction of the master circuit board subsystem design is the lack of an operating system. The capability provided in this thesis could not be fully utilized without an operating system and a mass storage device, such as a hard disk. Managing the virtual-memory and protected memory requirements would require a tremendous amount of code which is beyond the scope of this thesis. However, while designing the master circuit board subsystem, foresight was exercised to consider the requirements of an operating system. This confirms the need for a dialogue between system designers and operating system designers to communicate the system requirements.

APPENDIX A: MC68010 16-BIT MICROPROCESSOR

Since the entire hardware system design revolves about the MC68010 microprocessor, a description of the microprocessor, its external signals and its programming is appropriate.

A. MC68010 DESCRIPTION

The MC68010 has seventeen 32-bit general purpose registers, a 16 megabyte address space, virtual-memory/machine support, 57 instructions with 14 addressing modes using five main data types and memory-mapped input/output (I/O) [Ref. 7:p. 1-1]. Motorola provides a complete signal description and timing analysis of the MC68010 microprocessor [Ref. 18].

B. MC68010 SIGNALS

The MC68010 central processing unit (CPU) comes in a 64-pin package. As shown in Figure A.1, the signals are organized into groups and the direction of the signal flow is denoted by the arrows. To avoid any confusion over logic assertion levels, the asterisk (*) at the end of a signal name is used to denote an active low assertion level.

1. Address Bus

The address bus consists of 23 address lines giving an eight megaword address range for the CPU.

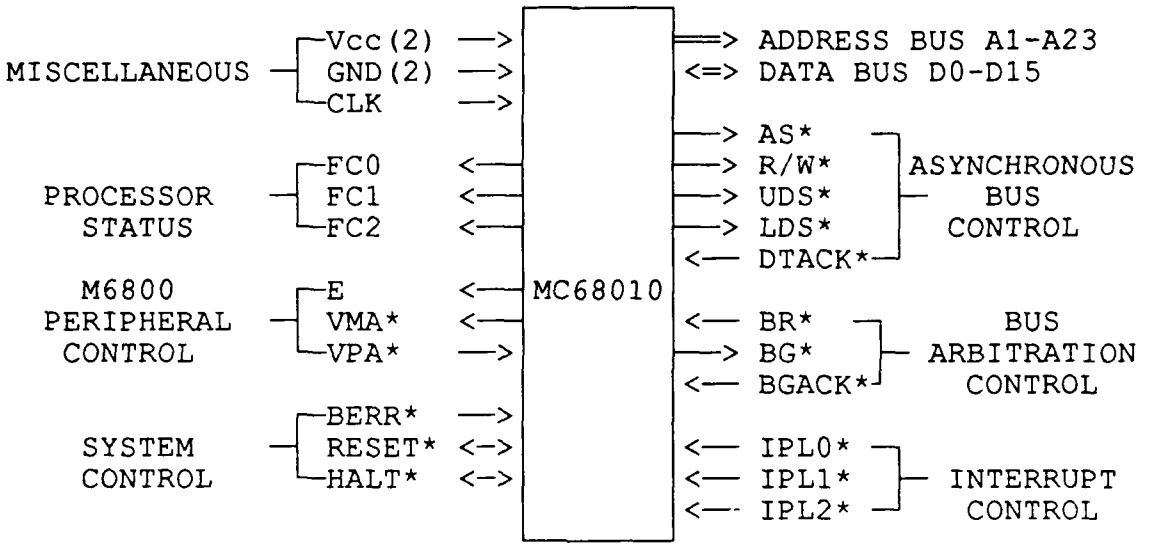


Figure A.1: MC68010 Signal Groups

2. Data Bus

The data bus is a 16-bit bi-directional bus used for transferring byte or word length data.

3. Asynchronous Bus Control

The asynchronous bus control group provides information about the data that is being transferred. The address strobe (AS*) signal signifies that valid address signals are being gated from the CPU. The read/write (R/W*) line denotes that the CPU is reading from a device (active high) or that the CPU is writing to the device (active low). The upper data strobe (UDS*) indicates that the data being transferred is on an even byte boundary. The lower data strobe (LDS*) indicates that the data being transferred is on an odd byte boundary. When UDS* and LDS* are both asserted, a word (16-bits) of data is being transferred. The UDS* and LDS*

signals together determine address bit A0, thus giving an address range of 16 megabytes for the CPU. The UDS*, LDS* and R/W* signals control the flow of the data on the data bus as illustrated in Table III [Ref. 18:p. 4-2]. Finally, the data transfer acknowledge (DTACK*) signal informs the CPU that the current data transfer has been completed by the peripheral device or memory location addressed.

TABLE III: DATA STROBE CONTROL OF THE DATA BUS

UDS*	LDS*	R/W*	D8 - D15	D0 - D7
1	1	1or0	NO VALID DATA BITS	NO VALID DATA BITS
0	0	1	VALID DATA BITS	VALID DATA BITS
1	0	1	NO VALID DATA BITS	VALID DATA BITS
0	1	1	VALID DATA BITS	NO VALID DATA BITS
0	0	0	VALID DATA BITS	VALID DATA BITS
1	0	0	#VALID DATA BITS 0-7	VALID DATA BITS
0	1	0	VALID DATA BITS	#VALID DATA BITS 8-15

These conditions are a result of current implementation and may not appear on future devices.

4. Bus Arbitration Control

As a group, the bus arbitration control signals provide a mechanism for the CPU to give up control of the bus. However, these signals do not determine (directly) which alternate bus master gets control. The bus request (BR*) signal is a signal generated by a device or devices requesting access to the bus. The bus grant (BG*) is a signal from the CPU indicating that it will release the bus at the end of the current bus cycle. The bus

grant acknowledge (BGACK*) is a signal asserted by an alternate bus master while it has control of the bus.

5. Interrupt Control

The interrupt priority levels (IPL0* through IPL2*) are signals which represent the encoded priority level for the highest priority device desiring interrupt service. The signal IPL0* is the least significant bit and the signal IPL2* is the most significant bit of the group. A level zero interrupt (all signals are asserted high) indicates there is no interrupt request pending. A level seven interrupt (all IPLx* signals are asserted low) has the highest priority and is non-maskable. This implies that level seven is not an ordinary interrupt level for requesting routine interrupt service. Rather, a level seven interrupt should be reserved for catastrophic events such as alternating current (AC) power failure where the non-maskable property is essential.

6. System Control

The system control group is used to reset the CPU and to indicate to the CPU that a bus error has occurred. It is also used to reset peripheral devices and to generate a bus error exception. The halt signal (HALT*), active low, is a bi-directional signal. As an input, it is used to stop the CPU at the completion of the current bus cycle. As an output, HALT* is asserted only when a double bus error or address error exception has caused the MC68010 to enter a halt state.

The reset signal (RESET*), active low, is also a bi-directional signal. It can be used as an input to reset the

internal microcircuitry within the CPU. When a reset instruction is executed by the CPU, it can be used to reset system devices.

Typically, a maximum time is allotted for data transfer. If the data transfer is not completed within the allotted time, bus error (BERR*) is asserted by a time out circuit called a watchdog timer. Often, the BERR* signal is used to inform the CPU that the current address on the address bus is invalid because no physical memory or peripheral device is mapped at that address. The BERR* signal can also be used to flag the condition that the CPU is making an attempt to write to read-only memory (ROM). In a virtual-memory system, BERR* is asserted by the memory management unit (MMU) when a page fault occurs.

7. M6800 Peripheral Control

The M6800 peripheral control group is a group of signals which are used to interface the MC68010's 16-bit asynchronous data bus to synchronous peripheral devices in the Motorola M6800 eight-bit family.

The enable (E) signal which acts as the 6800 phase two clock is used to synchronize data transfer between the MC68010 CPU and M6800 peripheral device. The E signal's period is ten clock periods of the MC68010's clock input. The valid peripheral address (VPA*) signal denotes to the CPU that the device selected is a M6800 peripheral device. The VPA* signal indicates to the CPU that it should initiate a data transfer synchronized with the E signal. The valid memory address (VMA*) signal from the CPU indicates to a

M6800 device that there is a valid address on the address bus and that the MC68010 is synchronized with the E signal.

8. Processor Status

The MC68010 has three function code lines (FC0 through FC2) which delineate the current processor state (user or supervisor) and the address space (program or data) being accessed as defined by Table IV [Ref. 18:p. 5-3]. The address strobe (AS*) signal from the CPU indicates that a valid address and function code are available from the CPU.

TABLE IV: STATE AND ADDRESS SPACE

FUNCTION CODE OUTPUT			ADDRESS SPACE
FC2	FC1	FC0	
0	0	0	UNDEFINED, RESERVED FOR FUTURE USE
0	0	1	USER DATA SPACE
0	1	0	USER PROGRAM SPACE
0	1	1	UNDEFINED, RESERVED FOR FUTURE USE
1	0	0	UNDEFINED, RESERVED FOR FUTURE USE
1	0	1	SUPERVISOR DATA SPACE
1	1	0	SUPERVISOR PROGRAM SPACE
1	1	1	CPU SPACE (INTERRUPT ACKNOWLEDGE)

9. Miscellaneous

Both Vcc pins and both GND pins must be connected in order to power the CPU. The clock (CLK) input signal is used to develop all the synchronizing signals required within the CPU.

C. PROGRAMMING

Motorola provides programming information in its reference manual [Ref. 7]. The MC68010's instruction set includes the following operations:

- Data Movement
- Integer Arithmetic
- Logical
- Shift and Rotate
- Bit Manipulation
- Bit Manipulation
- Binary Coded Decimal (BCD) Arithmetic
- Program Control
- System Control
- Multi-processor Communications

supporting the following data types:

- Bit
- BCD (Four-bits)
- Byte (Eight-bits)
- Word (16-bits)
- Long Word (32-bits)

Fourteen addressing modes that are available to the assembly language programmer. The addressing modes available include:

- Data Register Direct
- Address Register Direct
- Address Register Indirect
- Address Register Indirect with Postincrement
- Address Register Indirect with Predecrement
- Address Register Indirect with Offset
- Address Register Indirect with Index and Offset
- Absolute Short
- Absolute Long
- Program Counter with Offset
- Program Counter with Index and Offset
- Immediate Data
- Quick Immediate
- Implied Register

The following assets are available:

- Eight Data Registers
- Seven Address Registers
- User Stack Pointer (User Mode)
- Supervisor Stack Pointer (Supervisor Mode)
- Program Counter
- *Status Register (Supervisor mode)
- Vector Base Register (Supervisor Mode)
- Alternate Function Code Registers (Supervisor Mode)

* The condition code register is the lower byte of the status register and it is accessible in the user mode.

To support virtual-memory, the MC68010 microprocessor allows an interrupted bus cycle to be re-run after a bus error exception. The return from exception (RTE) instruction uses the format field of the exception stack to determine whether the exception was caused by bus or address error. After a bus or address error caused the exception, the CPU continues the interrupted instruction after completion of the exception routine. [Ref. 19]

APPENDIX B: MINIMAL SYSTEM EXCEPTION VECTOR TABLE AND MONITOR/DEBUGGER PROGRAM

This appendix contains the source listings of the exception vector table and monitor/debugger program. The separate file names are as follows:

- VECTABLE.ASM
- MAIN.ASM
- MESSAGE.ASM
- CONSOLE.ASM
- GETSTRIN.ASM
- GET_ADDR.ASM
- IO_UTIL.ASM
- DECODER.ASM
- BYTEOUT.ASM
- MEM_LIST.ASM
- HEXCONV.ASM
- GO.ASM
- STUB.ASM
- REG.ASM
- REGCHANG.ASM
- DOWNLOAD.ASM
- UNUSED.ASM

Using the 2500AD 68010 cross assembler and linker, a Motorola S-record format file was generated as a load module. The load module was loaded as a ASCII file into a Data I/O System 29 Universal Programmer. Once resident in the programmer, the load module was programmed to erasable programmable read-only memory (EPROM). It should be noted that the data section as contained in MAIN.ASM was not programmed on EPROM, but rather it resides in random access memory (RAM).

The first two entries in the exception vector table are used during the system boot up to provide the initial contents for the stack pointer and the program counter. The exception vector table contains the addresses of exception routines. The monitor/debugger

program initializes the MC68681 peripheral device and provides facilities for performing software debugging and the down-loading of files from an IBM XT/AT compatible computer.

```
*****************************************************************
*                EXCEPTION VECTOR TABLE                         *
*****************************************************************
* WRITTEN BY LARRY ABBOTT JUNE 5, 1987                          *
*****************************************************************
* FILENAME: VECTABLE.ASM                                        *
*****************************************************************
* VERSION 1.3                                                   *
* REV   DATE         NAME              DESCRIPTION              *
*  A  29 SEPT 87  DAVID M. SENDEK  ADDITIONAL DOCUMENTATION     *
*****************************************************************
*  DEFINING MODULES OF EXTERNALLY DECLARED VARIABLES:           *
*    BKPT        - GO.ASM                                       *
*    INIT        - MAIN.ASM                                     *
*    INIT_SP     - MAIN.ASM                                     *
*    MESSAGE     - MESSAGE.ASM                                  *
*    MONITOR     - MAIN.ASM                                     *
*    UNUSED      - UNUSED.ASM                                   *
*****************************************************************

     EXTERNAL     BKPT,INIT,INIT_SP,MESSAGE,MONITOR
     EXTERNAL     UNUSED

     ORG 0    VECTOR TABLE STARTS AT ABSOLUTE ADDRESS $000000
     LONG     INIT_SP          INITIAL STACK POINTER VECTOR
     LONG     INIT             INITIAL PROGRAM COUNTER (PC)
                               ;       VECTOR
     LONG     UNUSED           BUS ERROR VECTOR
     LONG     UNUSED           ADDRESS ERROR VECTOR
     LONG     UNUSED           ILLEGAL INSTRUCTION VECTOR
     LONG     UNUSED           ZERO DIVIDE VECTOR
     LONG     UNUSED           CHK INSTRUCTION VECTOR
     LONG     UNUSED           TRAPV INSTRUCTION VECTOR
     LONG     UNUSED           PRIVILEGE VIOLATION VECTOR
     LONG     UNUSED           TRACE VECTOR
     LONG     UNUSED           LINE 1010 EMULATION VECTOR
     LONG     UNUSED           LINE 1111 EMULATION VECTOR
     ORG $38                   NOTE: VECTOR NUMBERS 12 AND 13
                               ;     ARE UNASSIGNED,RESERVED
     LONG     UNUSED           FORMAT ERROR VECTOR
     LONG     UNUSED           UNINITIALIZED INTERRUPT VECTOR
     ORG $60                   NOTE: VECTOR NUMBERS 16-23 ARE
                               ;           UNASSIGNED,RESERVED
     LONG     UNUSED            SPURIOUS INTERRUPT VECTOR
     LONG     UNUSED            LEVEL 1 AUTOVECTOR VECTOR
     LONG     UNUSED            LEVEL 2 AUTOVECTOR VECTOR
     LONG     UNUSED            LEVEL 3 AUTOVECTOR VECTOR
     LONG     UNUSED            LEVEL 4 AUTOVECTOR VECTOR
     LONG     UNUSED            LEVEL 5 AUTOVECTOR VECTOR
     LONG     UNUSED            LEVEL 6 AUTOVECTOR VECTOR
     LONG     UNUSED            LEVEL 7 AUTOVECTOR VECTOR
```

```
LONG      BKPT            TRAP 0 VECTOR USED AS
                         ;    MONITOR BRKPT
LONG      UNUSED          TRAP 1 VECTOR
LONG      UNUSED          TRAP 2 VECTOR
LONG      UNUSED          TRAP 3 VECTOR
LONG      UNUSED          TRAP 4 VECTOR
LONG      UNUSED          TRAP 5 VECTOR
LONG      UNUSED          TRAP 6 VECTOR
LONG      UNUSED          TRAP 7 VECTOR
ORG $100                  NOTE: VECTOR NUMBERS 48-63 ARE
                          ;     UNASSIGNED,RESERVED
LONG      MONITOR         USER INTERRUPT 0    VECTOR
                          ;    DEFINED FOR MONITOR
LONG      UNUSED          USER INTERRUPT 1    VECTOR
LONG      UNUSED          USER INTERRUPT 2    VECTOR
LONG      UNUSED          USER INTERRUPT 3    VECTOR
LONG      UNUSED          USER INTERRUPT 4    VECTOR
LONG      UNUSED          USER INTERRUPT 5    VECTOR
LONG      UNUSED          USER INTERRUPT 6    VECTOR
LONG      UNUSED          USER INTERRUPT 7    VECTOR
LONG      UNUSED          USER INTERRUPT 8    VECTOR
LONG      UNUSED          USER INTERRUPT 9    VECTOR
LONG      UNUSED          USER INTERRUPT 10   VECTOR
LONG      UNUSED          USER INTERRUPT 11   VECTOR
LONG      UNUSED          USER INTERRUPT 12   VECTOR
LONG      UNUSED          USER INTERRUPT 13   VECTOR
LONG      UNUSED          USER INTERRUPT 14   VECTOR
LONG      UNUSED          USER INTERRUPT 15   VECTOR
LONG      UNUSED          USER INTERRUPT 16   VECTOR
LONG      UNUSED          USER INTERRUPT 17   VECTOR
LONG      UNUSED          USER INTERRUPT 18   VECTOR
LONG      UNUSED          USER INTERRUPT 19   VECTOR
LONG      UNUSED          USER INTERRUPT 20   VECTOR
LONG      UNUSED          USER INTERRUPT 21   VECTOR
LONG      UNUSED          USER INTERRUPT 22   VECTOR
LONG      UNUSED          USER INTERRUPT 23   VECTOR
LONG      UNUSED          USER INTERRUPT 24   VECTOR
LONG      UNUSED          USER INTERRUPT 25   VECTOR
LONG      UNUSED          USER INTERRUPT 26   VECTOR
LONG      UNUSED          USER INTERRUPT 27   VECTOR
LONG      UNUSED          USER INTERRUPT 28   VECTOR
LONG      UNUSED          USER INTERRUPT 29   VECTOR
LONG      UNUSED          USER INTERRUPT 30   VECTOR
LONG      UNUSED          USER INTERRUPT 31   VECTOR
LONG      UNUSED          USER INTERRUPT 32   VECTOR
LONG      UNUSED          USER INTERRUPT 33   VECTOR
LONG      UNUSED          USER INTERRUPT 34   VECTOR
LONG      UNUSED          USER INTERRUPT 35   VECTOR
LONG      UNUSED          USER INTERRUPT 36   VECTOR
LONG      UNUSED          USER INTERRUPT 37   VECTOR
LONG      UNUSED          USER INTERRUPT 38   VECTOR
LONG      UNUSED          USER INTERRUPT 39   VECTOR
```

```
LONG    UNUSED          USER INTERRUPT 40   VECTOR
LONG    UNUSED          USER INTERRUPT 41   VECTOR
LONG    UNUSED          USER INTERRUPT 42   VECTOR
LONG    UNUSED          USER INTERRUPT 43   VECTOR
LONG    UNUSED          USER INTERRUPT 44   VECTOR
LONG    UNUSED          USER INTERRUPT 45   VECTOR
LONG    UNUSED          USER INTERRUPT 46   VECTOR
LONG    UNUSED          USER INTERRUPT 47   VECTOR
LONG    UNUSED          USER INTERRUPT 48   VECTOR
LONG    UNUSED          USER INTERRUPT 49   VECTOR
LONG    UNUSED          USER INTERRUPT 50   VECTOR
LONG    UNUSED          USER INTERRUPT 51   VECTOR
LONG    UNUSED          USER INTERRUPT 52   VECTOR
LONG    UNUSED          USER INTERRUPT 53   VECTOR
LONG    UNUSED          USER INTERRUPT 54   VECTOR
LONG    UNUSED          USER INTERRUPT 55   VECTOR
LONG    UNUSED          USER INTERRUPT 56   VECTOR
LONG    UNUSED          USER INTERRUPT 57   VECTOR
LONG    UNUSED          USER INTERRUPT 58   VECTOR
LONG    UNUSED          USER INTERRUPT 59   VECTOR
LONG    UNUSED          USER INTERRUPT 60   VECTOR
LONG    UNUSED          USER INTERRUPT 61   VECTOR
LONG    UNUSED          USER INTERRUPT 62   VECTOR
LONG    UNUSED          USER INTERRUPT 63   VECTOR
LONG    UNUSED          USER INTERRUPT 64   VECTOR
LONG    UNUSED          USER INTERRUPT 65   VECTOR
LONG    UNUSED          USER INTERRUPT 66   VECTOR
LONG    UNUSED          USER INTERRUPT 67   VECTOR
LONG    UNUSED          USER INTERRUPT 68   VECTOR
LONG    UNUSED          USER INTERRUPT 69   VECTOR
LONG    UNUSED          USER INTERRUPT 70   VECTOR
LONG    UNUSED          USER INTERRUPT 71   VECTOR
LONG    UNUSED          USER INTERRUPT 72   VECTOR
LONG    UNUSED          USER INTERRUPT 73   VECTOR
LONG    UNUSED          USER INTERRUPT 74   VECTOR
LONG    UNUSED          USER INTERRUPT 75   VECTOR
LONG    UNUSED          USER INTERRUPT 76   VECTOR
LONG    UNUSED          USER INTERRUPT 77   VECTOR
LONG    UNUSED          USER INTERRUPT 78   VECTOR
LONG    UNUSED          USER INTERRUPT 79   VECTOR
LONG    UNUSED          USER INTERRUPT 80   VECTOR
LONG    UNUSED          USER INTERRUPT 81   VECTOR
LONG    UNUSED          USER INTERRUPT 82   VECTOR
LONG    UNUSED          USER INTERRUPT 83   VECTOR
LONG    UNUSED          USER INTERRUPT 84   VECTOR
LONG    UNUSED          USER INTERRUPT 85   VECTOR
LONG    UNUSED          USER INTERRUPT 86   VECTOR
LONG    UNUSED          USER INTERRUPT 87   VECTOR
LONG    UNUSED          USER INTERRUPT 88   VECTOR
LONG    UNUSED          USER INTERRUPT 89   VECTOR
LONG    UNUSED          USER INTERRUPT 90   VECTOR
LONG    UNUSED          USER INTERRUPT 91   VECTOR
```

```
LONG      UNUSED          USER INTERRUPT 92   VECTOR
LONG      UNUSED          USER INTERRUPT 93   VECTOR
LONG      UNUSED          USER INTERRUPT 94   VECTOR
LONG      UNUSED          USER INTERRUPT 95   VECTOR
LONG      UNUSED          USER INTERRUPT 96   VECTOR
LONG      UNUSED          USER INTERRUPT 97   VECTOR
LONG      UNUSED          USER INTERRUPT 98   VECTOR
LONG      UNUSED          USER INTERRUPT 99   VECTOR
LONG      UNUSED          USER INTERRUPT 100  VECTOR
LONG      UNUSED          USER INTERRUPT 101  VECTOR
LONG      UNUSED          USER INTERRUPT 102  VECTOR
LONG      UNUSED          USER INTERRUPT 103  VECTOR
LONG      UNUSED          USER INTERRUPT 104  VECTOR
LONG      UNUSED          USER INTERRUPT 105  VECTOR
LONG      UNUSED          USER INTERRUPT 106  VECTOR
LONG      UNUSED          USER INTERRUPT 107  VECTOR
LONG      UNUSED          USER INTERRUPT 108  VECTOR
LONG      UNUSED          USER INTERRUPT 109  VECTOR
LONG      UNUSED          USER INTERRUPT 110  VECTOR
LONG      UNUSED          USER INTERRUPT 111  VECTOR
LONG      UNUSED          USER INTERRUPT 112  VECTOR
LONG      UNUSED          USER INTERRUPT 113  VECTOR
LONG      UNUSED          USER INTERRUPT 114  VECTOR
LONG      UNUSED          USER INTERRUPT 115  VECTOR
LONG      UNUSED          USER INTERRUPT 116  VECTOR
LONG      UNUSED          USER INTERRUPT 117  VECTOR
LONG      UNUSED          USER INTERRUPT 118  VECTOR
LONG      UNUSED          USER INTERRUPT 119  VECTOR
LONG      UNUSED          USER INTERRUPT 120  VECTOR
LONG      UNUSED          USER INTERRUPT 121  VECTOR
LONG      UNUSED          USER INTERRUPT 122  VECTOR
LONG      UNUSED          USER INTERRUPT 123  VECTOR
LONG      UNUSED          USER INTERRUPT 124  VECTOR
LONG      UNUSED          USER INTERRUPT 125  VECTOR
LONG      UNUSED          USER INTERRUPT 126  VECTOR
LONG      UNUSED          USER INTERRUPT 127  VECTOR
LONG      UNUSED          USER INTERRUPT 128  VECTOR
LONG      UNUSED          USER INTERRUPT 129  VECTOR
LONG      UNUSED          USER INTERRUPT 130  VECTOR
LONG      UNUSED          USER INTERRUPT 131  VECTOR
LONG      UNUSED          USER INTERRUPT 132  VECTOR
LONG      UNUSED          USER INTERRUPT 133  VECTOR
LONG      UNUSED          USER INTERRUPT 134  VECTOR
LONG      UNUSED          USER INTERRUPT 135  VECTOR
LONG      UNUSED          USER INTERRUPT 136  VECTOR
LONG      UNUSED          USER INTERRUPT 137  VECTOR
LONG      UNUSED          USER INTERRUPT 138  VECTOR
LONG      UNUSED          USER INTERRUPT 139  VECTOR
LONG      UNUSED          USER INTERRUPT 140  VECTOR
LONG      UNUSED          USER INTERRUPT 141  VECTOR
LONG      UNUSED          USER INTERRUPT 142  VECTOR
LONG      UNUSED          USER INTERRUPT 143  VECTOR
```

```
LONG      UNUSED          USER INTERRUPT 144  VECTOR
LONG      UNUSED          USER INTERRUPT 145  VECTOR
LONG      UNUSED          USER INTERRUPT 146  VECTOR
LONG      UNUSED          USER INTERRUPT 147  VECTOR
LONG      UNUSED          USER INTERRUPT 148  VECTOR
LONG      UNUSED          USER INTERRUPT 149  VECTOR
LONG      UNUSED          USER INTERRUPT 150  VECTOR
LONG      UNUSED          USER INTERRUPT 151  VECTOR
LONG      UNUSED          USER INTERRUPT 152  VECTOR
LONG      UNUSED          USER INTERRUPT 153  VECTOR
LONG      UNUSED          USER INTERRUPT 154  VECTOR
LONG      UNUSED          USER INTERRUPT 155  VECTOR
LONG      UNUSED          USER INTERRUPT 156  VECTOR
LONG      UNUSED          USER INTERRUPT 157  VECTOR
LONG      UNUSED          USER INTERRUPT 158  VECTOR
LONG      UNUSED          USER INTERRUPT 159  VECTOR
LONG      UNUSED          USER INTERRUPT 160  VECTOR
LONG      UNUSED          USER INTERRUPT 161  VECTOR
LONG      UNUSED          USER INTERRUPT 162  VECTOR
LONG      UNUSED          USER INTERRUPT 163  VECTOR
LONG      UNUSED          USER INTERRUPT 164  VECTOR
LONG      UNUSED          USER INTERRUPT 165  VECTOR
LONG      UNUSED          USER INTERRUPT 166  VECTOR
LONG      UNUSED          USER INTERRUPT 167  VECTOR
LONG      UNUSED          USER INTERRUPT 168  VECTOR
LONG      UNUSED          USER INTERRUPT 169  VECTOR
LONG      UNUSED          USER INTERRUPT 170  VECTOR
LONG      UNUSED          USER INTERRUPT 171  VECTOR
LONG      UNUSED          USER INTERRUPT 172  VECTOR
LONG      UNUSED          USER INTERRUPT 173  VECTOR
LONG      UNUSED          USER INTERRUPT 174  VECTOR
LONG      UNUSED          USER INTERRUPT 175  VECTOR
LONG      UNUSED          USER INTERRUPT 176  VECTOR
LONG      UNUSED          USER INTERRUPT 177  VECTOR
LONG      UNUSED          USER INTERRUPT 178  VECTOR
LONG      UNUSED          USER INTERRUPT 179  VECTOR
LONG      UNUSED          USER INTERRUPT 180  VECTOR
LONG      UNUSED          USER INTERRUPT 181  VECTOR
LONG      UNUSED          USER INTERRUPT 182  VECTOR
LONG      UNUSED          USER INTERRUPT 183  VECTOR
LONG      UNUSED          USER INTERRUPT 184  VECTOR
LONG      UNUSED          USER INTERRUPT 185  VECTOR
LONG      UNUSED          USER INTERRUPT 186  VECTOR
LONG      UNUSED          USER INTERRUPT 187  VECTOR
LONG      UNUSED          USER INTERRUPT 188  VECTOR
LONG      UNUSED          USER INTERRUPT 189  VECTOR
LONG      UNUSED          USER INTERRUPT 190  VECTOR
LONG      UNUSED          USER INTERRUPT 191  VECTOR
END
```

```
*****************************************************************
*    MAIN IS THE ENTRY POINT INTO THE MONITOR.  MAIN            *
*    INITIALIZES THE RS-232 PORT BEFORE ENTERING THE            *
*    MONITOR.  ALSO, MAIN CONTAINS THE MEMORY MAPS,             *
*    EQUATES AND MEMORY ALLOCATIONS.                            *
*****************************************************************
*    68K MONITOR VERSION V1.3 - AN ACCUMULATION OF ALL          *
*                               PRIOR VERSIONS                  *
*     COPYRIGHT @ AUG. 1986 BY DR. LARRY ABBOTT                 *
*****************************************************************
*    FILENAME: MAIN.ASM                                         *
*****************************************************************
* VERSION 1.3                                                   *
* REV. MODIFIED BY        DATE         DESCRIPTION              *
*  A      LARRY ABBOTT    11/7/86                               *
*  B      LARRY ABBOTT    12/14/86 MONSTAT-ESCAPE               *
*  C      LARRY ABBOTT    6/6/87   ADAPT TO MC68681             *
*  D  DAVID M. SENDEK 29 SEPT 87   -INCLUDE VECTOR TABLE        *
*                                  -INCLUDE MONITOR PROMPT      *
*                                  -CORRECT FOR 68681           *
*****************************************************************
*       DEFINING MODULES OF EXTERNALLY DECLARED VARIABLES:      *
*       CMD_DECODE   - DECODER.ASM                              *
*       GETSTRING    - GETSTRIN.ASM                             *
*       MESSAGE      - MESSAGE.ASM                              *
*       MONMSG       - MESSAGE.ASM                              *
*       SCRLF        - IO_UTIL.ASM                              *
*****************************************************************

     GLOBAL         BKPTAB,BS,BTLEN,BUFFIN,CHECKSUM,CK_SUM
     GLOBAL         CONTINUE,CR
     GLOBAL         END_ADDRESS,EPROMRNG,EPROMWR,ESC,ESCAPE
     GLOBAL         FOUND,FWDARW,HEX_ERR,LF,MODIFY,MONSTAT,
     GLOBAL         NULL,PORT1,PORT2,RBA,RECFULL
     GLOBAL         SPACE,SRA,SRAM,SRAMSIZE,STRING,STRINGEND
     GLOBAL         SYSTAX,SRB,TBA,TBB,RBB
     GLOBAL         TBA,XEMPTY
     GLOBAL         INIT_SP,INIT,MONITOR
     EXTERNAL       CMD_DECODE,GETSTRING,MESSAGE,MONMSG,SCRLF
     EXTERNAL       PROMPT
*
          DATA      ALL R/W DATA IS STORED IN SRAM AT ADDRESS
*                    $010000
*
*         EQUATES
*
BS         EQU      $08     ASCII CODE FOR <-- (BACKSPACE)
CR         EQU      $0D     ASCII CODE FOR RETURN
EPROMRNG   EQU      $3FF    EPROM RNG 0 -> $3FF (EXCEPTION TBL)
ESC        EQU      $1B     ASCII CODE FOR ESCAPE
FWDARW     EQU      $3E     ASCII CODE FOR '>' (FORWARD ARROW)
LF         EQU      $0A     ASCII CODE FOR LINEFEED
```

```
NULL        EQU     $00     ASCII CODE FOR NUL
SPACE       EQU     $20     ASCII CODE FOR SPACE
BTLEN       EQU     $10     BREAKPOINT TABLE LENGTH IN WORDS
*
*     MEMORY ALLOCATIONS
*
BKPTAB      BLKW 3/2*BTLEN RESERVE BTLEN/2 32-BIT BKPT's
BUFFIN      BLKB $3F       RESERVE 63 BYTE INPUT BUFFER
END_ADDRESS BLKW 2         RESERVE WORD FOR END ADDRESS
MONSTAT     BLKW 1         RESERVE A WORD FOR MONITOR STATUS
STAX        BLKW 36        SAVE AREA FOR APPLICATION REG'S
SYSTAX      BLKW 2         RESERVE MEMORY FOR STACK POINTER
CK_SUM      BLKW 1         CHECK SUM STORAGE
SRAM        EQU  BKPTAB    DATA BEGINS AT LOW ADDR OF SRAM
SRAMSIZE    EQU  $3FFF     16K BYTES OF STATIC RAM
INIT_SP     EQU  $013FFE   INITIAL STACK POINTER
*
*     DEFINITION OF MONSTAT (MONITOR STATUS WORD)
*
EPROMWR     EQU     0      WRITE TO EPROM FLAG
ESCAPE      EQU     1      ESCAPE FLAG
CONTINUE    EQU     2      CONTINUATION FLAG
FOUND       EQU     3      CMD FOUND FLAG
HEX_ERR     EQU     4      HEX CONVERSION ERROR
MODIFY      EQU     5      MEMORY MODIFY FLAG
STRING      EQU     6      STRING BUILDING IN PROGRESS
STRINGEND   EQU     7      END OF STRING BUILDING
CHECKSUM    EQU     8      CHECKSUM ERROR FLAG
*
*     68681 EQUATES
*
RECFULL     EQU     $00  SRA(0)=1=>RECEIVE FIFO HAS A CHAR
XEMPTY      EQU     $02  SRA(2)=1=>XMIT HOLDING REG EMPTY
MR1RFSET    EQU     $1A  RESET MODE REG PTR & DISABLE XMIT/RECV
CLK_SRC     EQU     $30  XTAL/16 CLOCK
CONF_1AB    EQU     $13  8-BIT DATA, NO PARITY
CONF_2A     EQU     $07  1 STOP BIT
CONF_2B     EQU     $0F  2 STOP BITS
BAUD2400    EQU     $88  2400 BAUD
BAUD9600    EQU     $BB  9600 BAUD
EN_PORT     EQU     $45  RESET ERROR, ENABLE XMIT & RECV
RUPTMASK    EQU     $02  ENABLE RECV READY RUPT
RUPTVECT    EQU     $40  USER INTERRUPT 0 VECTOR
*
*     68681 REGISTERS
*
* CRT      <- PORT A:9600 BAUD,8 DATA BITS,
*                    NO PARITY,1 STOP BIT
* DOWNLOAD <- PORT B:2400 BAUD,8 DATA BITS,
*                    NO PARITY,2 STOP BITS
*
```

```
DUART     EQU  $7F7000   BASE ADDRESS FOR MC68681
PORT1     EQU  DUART     PORT A
PORT2     EQU  DUART+$10 PORT B
MR1A      EQU  1         R/W:MODE REG 1 FOR PORT A
MR2A      EQU  1         R/W:MODE REG 2 FOR PORT A
SRA       EQU  3         R  :STATUS REGISTER FOR PORT A
CSRA      EQU  3           W:CLOCK SELECT REGISTER A
CRA       EQU  5           W:COMMAND REGISTER FOR PORT A
RBA       EQU  7         R  :RECEIVER BUFFER FOR PORT A
TBA       EQU  7           W:TRANSMITTER BUFFER FOR PORT A
IPCR      EQU  9         R  :INPUT PORT CHANGE REGISTER
ACR       EQU  9           W:AUXILIARY CONTROL REGISTER
ISR       EQU  $B        R  :INTERRUPT STATUS REG
IMR       EQU  $B          W:INTERRUPT MASK REGISTER
CUR       EQU  $D        R  :COUNTER MODE: CURRENT CNTR MSB
CTUR      EQU  $D          W:COUNTER/TIMER UPPER REGISTER
CLR       EQU  $F        R  :COUNTER MODE: CURRENT CNTR LSB
CTLR      EQU  $F          W:COUNTER/TIMER LOWER REGISTER
MR1B      EQU  $11       R/W:MODE REG 1 FOR PORT B
MR2B      EQU  $11       R/W:MODE REG 2 FOR PORT B
SRB       EQU  $13       R  :STATUS REGISTER FOR PORT B
CSRB      EQU  $13         W:CLOCK SELECT REGISTER B
CRB       EQU  $15         W:COMMAND REGISTER FOR PORT B
RBB       EQU  $17       R  :RECEIVER BUFFER FOR PORT B
TBB       EQU  $17         W:TRANSMITTER BUFFER FOR PORT B
IVR       EQU  $19       R/W:INTERRUPT VECTOR REGISTER
OPCR      EQU  $1B         W:OUTPUT PORT CONFIGURATION REG
*
          CODE
*
INIT:  LEA     DUART,A4             A4 <-- PTR TO DUART
       CLR.W   MONSTAT              CLR MONITOR STATUS WORD
       MOVE.B  #MR1RESET,CRA(A4)    RESET PORT A MR1 PTR,
*                                      DISABLE XMIT & RECV
       MOVE.B  #MR1RESET,CRB(A4)    RESET PORT B MR1 PTR,
*                                      DISABLE XMIT & RECV
       MOVE.B  #CLK_SRC,ACR(A4)     CNTR/TMR CLK FROM CRYSTAL/16
       MOVE.B  #CONF_1AB,MR1A(A4)   PORT A:8 DATA BITS & NO
*                                      PARITY
       MOVE.B  #CONF_2A,MR2A(A4)    PORT A: 1 STOP BIT
       MOVE.B  #BAUD9600,CSRA(A4)   PORT A: 9600 BAUD
       MOVE.B  #CONF_1AB,MR1B(A4)   PORT B:8 DATA BITS & NO
*                                      PARITY
       MOVE.B  #CONF_2B,MR2B(A4)    PORT B: 2 STOP BITS
       MOVE.B  #BAUD2400,CSRB(A4)   PORT B: 2400 BAUD
       MOVE.B  #RUPTVECT,IVR(A4)    SET DUART INTERRUPT SERVICE
*                                      AT USER INTERRUPT 0
       MOVE.B  #EN_PORT,CRA(A4)     RESET ERRS & ENABLE          *
                                    XMIT/RCV
       MOVE.B  #EN_PORT,CRB(A4)     RESET ERRS & ENABLE          *
                                    XMIT/RCV
       MOVE.B  #RUPTMASK,IMR(A4)    RUPT WHEN PORT A RCVS CHAR
```

```
BANNER:BSR    SCRLF            MOVE CURSOR TO NEXT LINE
       LEA    MONMSG,A5        SET MESSAGE POINTER TO MONMSG
       BSR    MESSAGE          CRT<--68010 MONITOR V1.3
       BSR    SCRLF            MOVE CURSOR TO NEXT LINE
       LEA    PROMPT,A5        SET UP FOR A PROMPT TO THE CRT
       BSR    MESSAGE          SEND PROMPT TO CRT
LOOP:  BRA.S LOOP               WAIT FOR AN INTERRUPT
*
MONITOR: MOVE.L  SP,SYSTAX             SAVE PTR TO APPL REGs
         MOVEM.L A0-A7/D0-D7,-(SP)     SAVE ALL REGISTERS
         LEA     STAX,A6               SET MONITOR STATE PTR
         MOVEM.L (A6)+,A0-A5/D0-D7     GET LAST MONITOR STATE
         BSR     GETSTRING             ENTER MONITOR
         BCLR.B  #STRINGEND,MONSTAT    CHECK FOR END OF STRING
         BEQ     RESTORE               NOT THE END, SO EXIT
         BCLR.B  #STRING,MONSTAT       CLEAR NEW STRING FLAG
         BSR     CMD_DECODE            IF END THEN DECODE
         LEA     PROMPT,A5             SET MSG PNTR TO PROMPT
         BSR     MESSAGE               CRT <- '>' (CRT PROMPT)
RESTORE: MOVEM.L A0-A5/D0-D7,-(A6)     SAVE MONITOR STATE
         MOVEM.L (SP)+,A0-A7/D0-D7     RESTORE ALL REGISTERS
         RTE
         END
```

```
*****************************************************************
* THIS PROGRAM OUTPUTS MESSAGES TO THE CRT SCREEN.              *
*****************************************************************
* WRITTEN BY DR. LARRY ABBOTT                                   *
*****************************************************************
* FILENAME: MESSAGE.ASM                                         *
*****************************************************************
* VERSION 1.3                                                   *
* REV. MODIFIED BY      DATE       DESCRIPTION                  *
*  A   DAVID M. SENDEK  29 SEPT 87 -INCLUDE A MONITOR PROMPT*
*                                  -INCLUDE BUFFER FULL         *
*                                           CONDITION           *
*****************************************************************
*       DEFINING MODULES OF EXTERNALLY DECLARED VARIABLES:      *
*       ECHO1    - CONSOLE.ASM                                  *
*****************************************************************

     GLOBAL          BKPTMSG,EPROMSG,ERRMSG,HEXMSG,ILLMSG
     GLOBAL          MONMSG,REGERR,REGMSG,SREC_ERR,USEMSG
     GLOBAL          MESSAGE,PROMPT,BUFFULLMSG,SPCE
     EXTERNAL        ECHO1
*
CR          EQU     $0D          ASCII CODE FOR RETURN
LF          EQU     $0A          ASCII CODE FOR LINEFEED
NULL        EQU     $00          ASCII CODE FOR NUL
*
MESSAGE:MOVE.B (A5)+,D0    ;GET MESSAGE CHAR,
*                                INCREMENT POINTER
        BEQ.S  MSGRET      IF CHAR = NULL THEN EXIT
        BSR    ECHO1       ;OUTPUT CHAR TO CONSOLE
        BRA.S  MESSAGE     ;GET ANOTHER CHARACTER
MSGRET: RTS
;
BKPTMSG: BYTE     'BREAKPOINT TRAP AT '
         BYTE     NULL
ERRMSG:  BYTE     'ERROR RE-ENTER',CR,LF
         BYTE      NULL
EPROMSG: BYTE     'ATTEMPTED WRITE TO EPROM',CR,LF
         BYTE     NULL
HEXMSG:  BYTE     'HEX CONVERSION ERROR...RE-ENTER',CR,LF
         BYTE     NULL
ILLMSG:  BYTE     'ILLEGAL INSTRUCTION TRAP',CR,LF
         BYTE     NULL
MONMSG:  BYTE     '68010 MONITOR V1.3',CR,LF
         BYTE     'WRITTEN BY DR. LARRY ABBOTT',CR,LF
         BYTE     '@ COPYRIGHT 1986',CR,LF
         BYTE     NULL
REGERR:  BYTE     'REGISTER CONTENTS ERROR RE-ENTER',CR,LF
         BYTE     NULL
```

```
REGMSG:   BYTE     'D0=',NULL,' D1=',NULL,' D2=',NULL,' D3=',
                   NULL,CR,LF
          BYTE     'D4=',NULL,' D5=',NULL,' D6=',NULL,' D7=',
                   NULL,CR,LF
          BYTE     'A0=',NULL,' A1=',NULL,' A2=',NULL,' A3=',
                   NULL,CR,LF
          BYTE     'A4=',NULL,' A5=',NULL,' A6=',NULL,' A7=',
                   CR,LF
          BYTE     'SR=',NULL,' PC=',NULL,' (PC)=',NULL,CR,LF
          BYTE     'US=',NULL,' SS=',NULL,CR,LF
          BYTE     NULL
SREC_ERR: BYTE     'S RECORD ERROR MESSAGE',LF,CR
          BYTE     NULL
USEMSG:   BYTE     'UNUSED EXCEPTION ENCOUNTERED',LF,CR
          BYTE     'WITH FORMAT WORD = '
          BYTE     NULL
PROMPT:   BYTE     '>'
          BYTE     NULL
SPCE:     BYTE     ' '
          BYTE     NULL
BUFFULLMSG: BYTE   LF,CR,'INPUT BUFFER IS FULL, TRY AGAIN.',LF,CR
            BYTE   NULL
          END
```

```
*************************************************************
* THIS MODULE INPUTS FROM THE KEYBOARD AND DOWNLOAD PORT,   *
* AND IT OUTPUTS CHARACTERS TO THE CRT.                     *
*************************************************************
* NEW CONSOLE WRITTEN DEC. 19, 1986  BY DR. LARRY ABBOTT    *
*************************************************************
* FILENAME: CONSOLE.ASM                                     *
*************************************************************
* VERSION 1.3                                               *
* REV. MODIFIED BY          DATE         DESCRIPTION        *
*  A   LARRY ABBOTT         6/6/87    ADAPT TO 68681        *
*  B   DAVID M. SENDEK 30 SEPT 87  DOCUMENTATION UPGRADE    *
*************************************************************
*       DEFINING MODULES OF EXTERNALLY DECLARED VARIABLES:  *
*       ESCAPE   - MAIN.ASM                                 *
*       MONSTAT  - MAIN.ASM                                 *
*       PORT1    - MAIN.ASM                                 *
*       PORT2    - MAIN.ASM                                 *
*       RECFULL  - MAIN.ASM                                 *
*       RBA,RBB  - MAIN.ASM                                 *
*       SRA,SRB  - MAIN.ASM                                 *
*       TBA,TBB  - MAIN.ASM                                 *
*************************************************************

          GLOBAL          ECHO1,ECHO2
          GLOBAL          GETCHAR1,GETCHR2
          GLOBAL          SCANCHR2
          EXTERNAL        ESCAPE,MONSTAT,PORT1,PORT2
          EXTERNAL        RECFULL,RBA,SRA,TBA,TBB,SRB
          EXTERNAL        XEMPTY,RBB
*
ESC       EQU    $1B              ASCII CODE FOR ESCAPE
*
GETCHAR1: LEA    PORT1,A4         POINT TO RS_232 PORT 1
          BTST.B #RECFULL,SRA(A4) CONSOLE CHAR READY ?
          BEQ    GETCHAR1           - NO,  CHECK AGAIN
          MOVE.B RBA(A4),D0         - YES, GET CHAR
          RTS
GETCHAR2: LEA    PORT2,A4         POINT TO RS-232 PORT 2
          BTST.B #RECFULL,SRB(A4) CONSOLE CHAR READY ?
          BEQ    GETCHAR2           - NO,  CHECK AGAIN
          MOVE.B RBB(A4),D0         - YES, GET CHAR
          RTS
*
*     SCANCHAR GETS A CHARACTER FROM A PORT IF IT IS THERE
*     OTHERWISE, SCANCHAR RETURNS TO THE CALLING ROUTINE
*
SCANCHR1  LEA    PORT1,A4         POINTS TO RS-232 PORT 1
          BTST.B #RECFULL,SRA(A4) DOES PORT 1 HAVE A CHAR?
          BEQ.S  SCAN1_EX           - NO,  EXIT
          MOVE.B RBA(A4),D0         - YES, GET CHAR
SCAN1_EX  RTS
```

```
SCANCHR2  LEA     PORT2,A4          POINTS TO RS-232 PORT 2
          BTST.B  #RECFULL,SRB(A4)  DOES PORT 2 HAVE A CHAR?
          BEQ.S   SCAN2_EX           - NO,  EXIT
          MOVE.B  RBB(A4),D0         - YES, GET CHAR
SCAN2_EX  RTS
*
* WHILE DOWNLOADING CHARACTERS FROM PORT 2, THIS PROCESS CAN BE
* HALTED BY SENDING AN ESC CHARACTER FROM THE KEYBOARD TO PORT 1
*
GETCHR2   BSR     SCANCHR1          GET CHAR FROM PORT 1,IF PRESENT
          CMP.B   #ESC,D0           IS THE CHAR AN ESCAPE ?
          BEQ     GC2_EXIT           - YES, SO EXIT
          BSR     GETCHAR2           - NO,  GET DOWNLOAD CHAR
          BRA.S   EXIT_GC2
GC2_EXIT  BSET.B  #ESCAPE,MONSTAT   IF ESC CHAR, SET MONSTAT BIT
EXIT_GC2  RTS
*
ECHO2     LEA     PORT2,A4          POINTS TO RS-232 PORT 2
          BTST.B  #XEMPTY,SRB(A4)   IS CONSOLE XMIT RDY ?
          BEQ     ECHO2             - NO,  CHECK AGAIN
          MOVE.B  D0,TBA(A4)        - YES, OUTPUT CHAR TO PORT 1
          RTS

ECHO1     LEA     PORT1,A4          POINTS TO RS-232 PORT 1
          BTST.B  #XEMPTY,SRA(A4)   IS CONSOLE XMIT RDY ?
          BEQ     ECHO1             - NO,  CHECK AGAIN
          MOVE.B  D0,TBA(A4)        - YES, OUTPUT CHAR TO PORT 1
          RTS
*
          END
```

```
*****************************************************************
*THIS PROGRAM BUILDS THE CMD STRING INPUT FROM THE KEYBOARD.*
*****************************************************************
* WRITTEN BY DR. LARRY ABBOTT                                   *
*****************************************************************
* FILENAME: GETSTRIN.ASM                                        *
*****************************************************************
* VERSION 1.3                                                   *
* REV.  MODIFIED BY       DATE       DESCRIPTION                *
*  A    DAVID M. SENDEK   2 OCT 87   DOCUMENTATION UPGRADE      *
*****************************************************************
* DEFINING MODULES OF EXTERNALLY DECLARED VARIABLES:            *
*   BS          - MAIN.ASM          MESSAGE  - MESSAGE.ASM      *
*   BUFFIN      - MAIN.ASM          SPCE     - MESSAGE.ASM      *
*   CR          - MAIN.ASM                                      *
*   CMD_DECODE  - DECODER.ASM                                   *
*   ECHO1       - CONSOLE.ASM                                   *
*   GETCHAR1    - CONSOLE.ASM                                   *
*   MONSTAT     - MAIN.ASM                                      *
*   STRING      - MAIN.ASM                                      *
*   STRINGEND   - MAIN.ASM                                      *
*   BUFFULLMSG - MESSAGE.ASM                                    *
*****************************************************************


          GLOBAL    GETSTRING
          EXTERNAL  BS,BUFFIN,CR,CMD_DECODE,ECHO1,GETCHAR1
          EXTERNAL  MONSTAT,STRING,STRINGEND
          EXTERNAL  BUFFULLMSG,SPCE
          EXTERNAL  MESSAGE
*
GETSTRING:BSET.B #STRING,MONSTAT    IS THIS A NEW STRING ?
          BNE    BUILD              - NO,SKIP PTR INIT
          BCLR.B #STRINGEND,MONSTAT - YES,CLR STRG END BIT
          LEA    BUFFIN+1,A0        - YES, INIT STRING PTR
BUILD:    BSR    GETCHAR1                 D0 <- CHR FROM CRT
          CMP.B  #CR,D0             IS CHAR A CR ?
          BNE    ADD_STRING         - NO, ADD CHAR TO STRG
          BSET.B #STRINGEND,MONSTAT - YES,SET STRG END BIT
          MOVE.W A0,D0              - YES, D0 <-- CURRENT
*                                                BUFFIN PTR
          SUB.W  #BUFFIN+1,D0       - YES, CALC BUFFIN LEN
          MOVE.B D0,BUFFIN          - YES, BUFFIN(0)<-
*                                          BUFFIN LENGTH
          BRA    STRING_EXIT        - YES, EXIT
ADD_STRING: BSR   ECHO1             ECHO CHAR TO CRT
            BSR   CONCAT            ADD CHAR TO END OF STRG
STRING_EXIT:RTS
```

```
*      CONCAT CONCATENATES THE CHAR ONTO THE END OF THE STRING
*
CONCAT:      CMP.B   #BS,D0          IS INPUT CHAR A BACKSPACE?
             BEQ     BKSPACE           - YES, GOT BACKSPACE
             CMPA.L  BUFFIN+63,A0    IS BUFFIN FULL ?
             BNE     ADD_TO_STRING    - NO,  ADD BYTE TO STRING
             LEA     BUFFULLMSG,A5    - YES, SET UP POINTER
*                                             FOR MESSAGE
             BSR     MESSAGE          - YES, SEND MSG TO CRT
             BRA     CONCAT_EXIT      - YES, NOW EXIT
ADD_TO_STRING:MOVE.B D0,(A0)+         ADD BYTE TO STRING

             BRA     CONCAT_EXIT
BKSPACE:     CMPA.L  BUFFIN,A0       IS BUFFIN PTR POINTING TO
*                                                1st BYTE ?
             BEQ     CONCAT_EXIT      - YES, EXIT
             SUBQ.W  #1,A0            - NO,  BACKUP BUFFIN PNTR
             LEA     SPCE,A5
             BSR     MESSAGE
CONCAT_EXIT:RTS
             END
```

```
*****************************************************************
* GET_ADDRESS CONVERTS THE START AND END ADDRESS TO HEX.        *
*****************************************************************
* WRITTEN BY DR. LARRY ABBOTT                                   *
*****************************************************************
* FILENAME: GET_ADDR.ASM                                        *
*****************************************************************
* VERSION 1.3                                                   *
* REV.   MODIFIED BY      DATE          DESCRIPTION             *
*  A     DAVID M. SENDEK  30 SEPT 87    DOCUMENTATION UPGRADE*
*****************************************************************
* DEFINING MODULES OF EXTERNALLY DECLARED VARIABLES:            *
*   BUFFIN       - MAIN.ASM                                     *
*   END_ADDRESS  - MAIN.ASM                                     *
*   HEX_CONV     - HEXCONV.ASM                                  *
*   HEX_ERR      - MAIN.ASM                                     *
*   MONSTAT      - MAIN.ASM                                     *
*   HEXMSG       - MESSAGE.ASM                                  *
*   MESSAGE      - MESSAGE.ASM                                  *
*****************************************************************

          GLOBAL     GET_ADDR
          EXTERNAL   BUFFIN,END_ADDRESS,HEX_CONV,HEX_ERR
          EXTERNAL   MONSTAT,HEXMSG,MESSAGE
*
GET_ADDR:  CLR.L  D2                CLEAR HEX BUFFER
           LEA    0,A2              CLEAR START ADDRESS
           LEA    0,A3              CLEAR END ADDRESS
           CLR    D3
           MOVE.B BUFFIN,D3         D3 <-- BUFFIN LENGTH
           BLE    EXIT              EXIT IF NULL CMD STRING
           SUBQ.W #1,D3             ADJUST FOR DBCC INST
START_ADDR:MOVE.B (A0)+,D0          D0 <-- BUFFIN(I) &
*                                           I <- I + 1
           CMP.B  #',',D0           IS CHAR IN D0 A COMMA ?
           BEQ    STORE_START       - YES, INDICATE END OF
*                                           START ADDRESS
           BSR    HEX_CONV          CONVERT 1 CHAR OF START
*                                          ADDR TO HEX
           BTST.B #HEX_ERR,MONSTAT  WAS THERE AN HEX
*                                       CONVERSION ERROR ?
           BNE    ADDR_ERR           - YES, EXIT ROUTINE
           DBF    D3,START_ADDR     IF MORE CHARACTERS CONT
STORE_START:SUBQ.W #1,D3            ADJUST LENGTH FOR COMMA
            MOVE.L D2,A2            STORE START ADDRESS IN A2
            CLR.L  D2               CLEAR HEX BUFFER
*
*     D3 CONTAINS THE LENGTH OF THE REMAINING COMMAND LINE
*
            TST.W  D3             IS BUFFIN LENGTH < 0 ?
            BMI    ADDREXIT        - YES,EXIT WITH END.ADDR=0
```

```
END_ADDR:MOVE.B  (A0)+,D0          D0 <-- BUFFIN(I, & I <- I+1
         BSR     HEX_CONV          CONVERT 1 CHAR OF END
*                                          ADDR TO HEX
         BTST.B  #HEX_ERR,MONSTAT  WAS THERE AN HEX
*                                        CONVERSION ERROR ?
         BNE     ADDR_ERR           - YES, EXIT ROUTINE
         DBF     D3,END_ADDR       IF MORE CHARS CONTINUE
         MOVE.L  D2,A3                ELSE STR END ADDR IN A3
ADDREXIT MOVE.L  A3,END_ADDRESS    SAV END ADR IN MEM
         BRA     EXIT
ADDR_ERR LEA     HEXMSG,A5
         BSR     MESSAGE
EXIT     RTS
         END
```

```
*************************************************************
* THIS PROGRAM CONTAINS A GROUP OF CONSOLE UTILITIES.       *
*************************************************************
* WRITTEN BY LARRY ABBOTT JAN. 1986                         *
*************************************************************
* FILENAME: IO_UTIL.ASM                                     *
*************************************************************
* VERSION 1.3                                               *
* REV.  MODIFIED BY     DATE        DESCRIPTION             *
*  A    DAVID M. SENDEK  30 SEPT 87  -DOCUMENTATION UPGRADE *
*                                    -CORRECT FOR 68681     *
*************************************************************
* DEFINING MODULES OF EXTERNALLY DECLARED VARIABLES:        *
*    BS        - MAIN.ASM          PORT1   -  MAIN.ASM      *
*    CR        - MAIN.ASM                                   *
*    ECHO1     - CONSOLE.ASM                                *
*    ESC       - CONSOLE.ASM                                *
*    FWDARW    - MAIN.ASM                                   *
*    GETCHAR1  - CONSOLE.ASM                                *
*    LF        - MAIN.ASM                                   *
*    RECFULL   - MAIN.ASM                                   *
*    SRA       - MAIN.ASM                                   *
*    SPACE     - MAIN.ASM                                   *
*************************************************************

          GLOBAL    BACKSPACES,SCROLL,SCRLF,SPACES
          EXTERNAL  BS,CR,ECHO1,ESC,FWDARW,GETCHAR1,LF
          EXTERNAL  RECFULL,SPACE
          EXTERNAL  SRA,PORT1

*     BACKSPACES MOVES THE CURSOR ON THE CRT TO THE LEFT
*               N TIMES
BACKSPACES:SUBQ.W #1,D2       ADJ INDEX FOR THE # OF BK_SP
BK_SPACE:  MOVE.B #BS,D0      D0 <- ASCII CODE FOR BACKSPACE
           BSR    ECHO1       OUTPUT BACKSPACE TO CONSOLE
           DBF    D2,BK_SPACE IF MORE BCKSP LOOP TO BK_SPACE
           RTS
*
*     SCRLF SEND A CARRIAGE RETURN AND A LINEFEED
*          TO THE CONSOLE
*
SCRLF: MOVE.B  #CR,D0     D0 <-- ASCII CODE FOR CR
       BSR     ECHO1      OUTPUT CR TO CONSOLE
       MOVE.B  #LF,D0     D0 <-- ASCII CODE FOR LF
       BSR     ECHO1      OUTPUT LF TO CONSOLE
       RTS
```

```
*      SPACES MOVE THE CURSOR ON THE CRT TO THE RIGHT N TIMES
*
SPACES:     SUBQ.W #1,D2          ADJUST INDEX FOR THE # OF SP
SPACE_LOOP:MOVE.B #SPACE,D0       ASCII CODE FOR ' '
           BSR    ECHO1           OUTPUT SPACE TO CONSOLE
           DBF    D2,SPACE_LOOP  IF MORE SPACES LOOP TO SPACE
           RTS
*      SCROLL ALLOWS THE SCREEN SCROLL TO BE ABORTED BY AN ESC
*      OR STOPPED AND STARTED BY ANY OTHER KEY
*
SCROLL:    LEA      PORT1,A4
           BTST.B   #RECFULL,SRA(A4) GET CONSOLE STATUS
           BEQ.S    SCROLL_EXIT      IF NO CHAR FROM
*                                        CONSOLE,EXIT
           BSR      GETCHAR1         ELSE GET CHAR
           CMP.B    #ESC,D0          IS THE CHAR AN ESC?
           BEQ.S    SCROLL_EXI       - YES, ABORT

PAUSE_CHK:LEA       PORT1,A4
           BTST.B   #RECFULL,SRA(A4) GET CONSOLE STATUS
           BEQ.S    PAUSE_CHK        IF NO NEW KEY STROKE, WAIT
           BSR      GETCHAR1         ELSE GET CHAR
SCROLL_EXIT:RTS
            END
```

```
*****************************************************************
* THIS PROGRAM DECODES COMMANDS FROM THE COMMAND LINE.          *
*****************************************************************
* 68K MONITOR VERSION 1.3                                       *
* WRITTEN BY DR. LARRY ABBOTT NOV. 7, 1986                      *
*****************************************************************
* FILENAME: DECODER.ASM                                         *
*****************************************************************
* VERSION 1.3                                                   *
* REV.  MODIFIED BY      DATE      DESCRIPTION                  *
*  A    DAVID M. SENDEK  1 OCT 87  DOCUMENTATION UPGRADE        *
*****************************************************************
* DEFINING MODULES OF EXTERNALLY DECLARED VARIABLES:            *
*  BUFFIN   - MAIN.ASM          BKPT_LIST   - STUB.ASM          *
*  ERRMSG   - MESSAGE.ASM       DOWNLOAD    - DOWNLOAD.ASM      *
*  FOUND    - MAIN.ASM          GO          - GO.ASM            *
*  MESSAGE  - MESSAGE.ASM       MEM_DISPLAY - MEM_LIST.ASM      *
*  MONSTAT  - MAIN.ASM          MEM_MODIFY  - MEM_LIST.ASM      *
*  NULL     - MAIN.ASM          NO_BKPT     - STUB.ASM          *
*  SPACE    - MAIN.ASM          REG         - REG.ASM           *
*  SCRLF    - IO_UTIL.ASM       REGCHANG    - REGCHANG.ASM      *
*  BKPT     - GO.ASM                                           *
*****************************************************************
*  COMMAND FORMATS:                                             *
*          LEGEND :  < .. >  - OPTIONAL                         *
*                    { .. }  - SELECT ONE ITEM                  *
*                       xx   - NUMBER 0 -> 15                   *
*           NOTE   : ALL ADDRESSES AND VALUES IN HEX            *
*                                                               *
*  BREAK POINT       -   BR          (NOT IMPLEMENTED)          *
*  NO BREAKPOINT     - NOBR          (NOT IMPLEMENTED)          *
*  DOWNLOAD          - LOAD                                     *
*  GO                - GO address <,break point address>        *
*  MEMORY MODIFY     - MM start address <,end address>          *
*  MEMORY DISPLAY    - MD start address <,end address>          *
*  REGISTER CHANGE   - RCH { Axx,Dxx,PC,US,SP,SR} value         *
*  DISPLAY REGISTERS- REG                                       *
*****************************************************************


     GLOBAL      CMD_DECODE
     EXTERNAL    BUFFIN,ERRMSG,FOUND,MESSAGE,MONSTAT,NULL
     EXTERNAL    SPACE,SCRLF
     EXTERNAL    BKPT,BKPT_LIST,DOWNLOAD,GO
     EXTERNAL    MEM_DISPLAY,MEM_MODIFY,NO_BKPT,REG,REGCHANG
*
*
CMD_DECODE: LEA     COMMANDS,A1     INITIALIZE COMMAND POINTER
            BCLR    #FOUND,MONSTAT
DECODE_INIT:LEA     BUFFIN+1,A0     INITIALIZE BUFFIN POINTER
            MOVE.L  #3,D1           INIT INDEX FOR 4 CHARS
```

```
SCAN:        MOVE.B  (A1)+,D0        GET COMMAND.TABLE(I)
*                                        & I<--I+1
             CMP.B   #SPACE,D0       IS CHARACTER A SPACE ?
             BEQ     FOUND_CMD        - YES, FOUND COMMAND
             CMP.B   #NULL,D0        IS CHARACTER A NULL ?
             BEQ     NO_CMD           - YES, EXHAUSTED COM TABLE
             CMP.B   (A0)+,D0        IS BUFFIN = COMMAND.TABLE ?
             DBNE    D1,SCAN          - YES & MORE CHAR, CONT
             BNE     ADDR_FIELD       - NO, ADJUST ADDR FOR NEXT
*                                         COMMAND
FOUND_CMD:  BSET    #FOUND,MONSTAT   SET COMMAND FND STATUS BIT
            CMPI.W  #0,D1            IS COMMAND A 4 CHAR COM?
            BMI     CMD_FOUND         - YES,SKIP "JUMP ADDRESS"
*                                         ADJUST
ADDR_FIELD: ADDQ.L  #2,D1            ADJUST INDEX FOR NEXT COM
            ADD.L   D1,A1            ADD INDEX TO COMMAND PNTR
            BCLR    #FOUND,MONSTAT   CLEAR COM FOUND STATUS BIT
            BEQ     DECODE_INIT      CHECK NEXT CMD
            SUB.L   #5,D1
            ADD.B   D1,BUFFIN        ADJUST BUFFIN LENGTH
            SUBQ.L  #2,A1            ADJUST ADDRESS FOR JUMP
CMD_FOUND:  MOVE.W  (A1),A1          GET JUMP ADDRESS
            JSR     (A1)             JUMP TO COMMAND
            BRA     DECODEXT         EXIT DECODER
NO_CMD:     BSR     SCRLF
            MOVE.W  #ERRMSG,A5       SET MESSAGE POINTER
            BSR     MESSAGE          PRINT ERROR MESSAGE TO CRT
DECODEXT:   RTS
*
          EVEN ON
COMMANDS:       BYTE     'BR  '
                WORD     BKPT_LIST
                BYTE     'LOAD'
                WORD     DOWNLOAD
                BYTE     'GO  '
                WORD     GO
                BYTE     'MD  '
                WORD     MEM_DISPLAY
                BYTE     'MM  '
                WORD     MEM_MODIFY
                BYTE     'NOBR'
                WORD     NO_BKPT
                BYTE     'RCH '
                WORD     REGCHANG
                BYTE     'REG '
                WORD     REG
                BYTE     NULL,NULL,NULL,NULL
          EVEN    OFF
          END
```

```
*****************************************************************
* THIS PROGRAM CONVERTS A BYTE INTO 2 ASCII CHARACTERS AND      *
* IT SENDS THE CHARACTERS TO THE CRT DISPLAY.                   *
*****************************************************************
* WRITTEN BY DR. LARRY ABBOTT                                   *
*****************************************************************
* FILENAME: BYTEOUT.ASM                                         *
*****************************************************************
* VERSION 1.3                                                   *
* REV. MODIFIED BY     DATE        DESCRIPTION                  *
*  A   DAVID M. SENDEK 1 OCT 87    DOCUMENTATION UPGRADE        *
*****************************************************************
* DEFINING MODULES OF EXTERNALLY DECLARED VARIABLES             *
*    ECHO1 - CONSOLE.ASM                                        *
*****************************************************************

          GLOBAL     OUTPUT_BYTE
          EXTERNAL   ECHO1
*
OUTPUT_BYTE:MOVE.B D0,D2    MAKE A TEMPORARY COPY OF BYTE
            LSR.B  #4,D0    SHIFT M.S. NIBBLE TO L.S. NIBBLE
            BSR    ASCONV   CONVERT M.S. NIBBLE TO ASCII
            MOVE.B D2,D0    D0  <-- TEMPORARY COPY OF BYTE
            ANDI.B #$0F,D0 MASK OFF M.S. NIBBLE
            BSR    ASCONV   CONVERT L.S. NIBBLE TO ASCII
            RTS
ASCONV:     ADDI.B #$30,D0 ADD ASCII BASE
            CMP.B  #$3A,D0 IS NUMBER 0-9 ?
            BLT    ASCOUT    - YES, OUTPUT TO CONSOLE
            ADDQ.B #7,D0    ADJUST FOR A - F (HEX)
ASCOUT:     BSR    ECHO1    OUTPUT TO CONSOLE
            RTS
            END
```

```
*****************************************************************
* THIS PROGRAM MODIFIES OR LISTS THE CONTENTS OF THE            *
* SPECIFIED MEMORY LOCATIONS.                                   *
*****************************************************************
* WRITTEN BY DR. LARRY ABBOTT                                   *
*****************************************************************
* FILENAME: MEM_LIST.ASM                                        *
*****************************************************************
* VERSION 1.3                                                   *
* REV.  MODIFIED BY      DATE     DESCRIPTION                   *
*  A    DAVID M. SENDEK  1 OCT 87 DOCUMENTATION UPGRADE         *
*****************************************************************
* DEFINING MODULES OF EXTERNALLY DECLARED VARIABLES:            *
*  BUFFIN       -  MAIN.ASM        MONSTAT      - MAIN.ASM      *
*  BACKSPACES   -  IO_UTIL.ASM     OUTPUT_BYTE  - BYTEOUT.ASM   *
*  END_ADDRESS  -  MAIN.ASM        SCRLF        - IO_UTIL.ASM   *
*  ESC          -  MAIN.ASM        SCROLL       - IO_UTIL/ASM   *
*  GET_ADDR     -  GET_ADDR.ASM    SPACE        - MAIN.ASM      *
*  GETSTRING    -  GETSTRIN.ASM    SPACES       - IO_UTIL.ASM   *
*  HEX_CONV     -  HEXCONV.ASM     STRINGEND    - MAIN.ASM      *
*  HEX_ERR      -  MAIN.ASM        STRING       - MAIN.ASM      *
*  MODIFY       -  MAIN.ASM                                     *
*****************************************************************

     GLOBAL   MEM_DISPLAY,MEM_MODIFY
     EXTERNAL BUFFIN,BACKSPACES,END_ADDRESS,ESC
     EXTERNAL GET_ADDR,GETSTRING,HEX_CONV,HEX_ERR,MODIFY
     EXTERNAL MONSTAT,OUTPUT_BYTE,SCRLF,SCROLL,SPACE,SPACES
     EXTERNAL STRINGEND,STRING
*
MEM_MODIFY:BSET.B  #MODIFY,MONSTAT     SET MODIFY FLAG
           BSR     MEM_DISPLAY         DISPLAY MEMORY
           BCLR.B  #MODIFY,MONSTAT     CLEAR MODIFY FLAG
           RTS
*
*     THIS PROGRAM LIST THE CONTENTS OF THE SPECIFIED
*
MEM_DISPLAY:CMPI.B #SPACE,(A0)        DOES BUFFIN(I)
*                                       CHAR = SPACE?
            BNE    START_ADDR           - NO,  GET START & END
*                                             ADDRESS
            ADDQ.W #1,A0                - YES, SO I <-- I+1
            SUBQ.B #1,BUFFIN          DECREMENT BUFFIN LENGTH
            BRA    MEM_DISPLAY        CONT SCANNING BUFFIN
START_ADDR: BSR    GET_ADDR           CONVERT ADDRS TO HEX
            BCLR.B #HEX_ERR,MONSTAT   WAS THERE AN HEX ERROR ?
            BNE    MD_EXIT              - YES, SO EXIT
NEWLINE:    BSR    SCRLF              MOVE CURSOR TO NEXT LINE
            BSR    LINE_NUMBER        DISPLAY LINE ADDRESS
```

```
GETABYTE:    MOVE.B (A2)+,D0          D0 <-- (START ADDRESS)
             BSR    OUTPUT_BYTE       OUTPUT BYTE TO CRT
             BTST   #MODIFY,MONSTAT   IS MEMORY MODIFY STATUS
*                                         BIT SET ?
             BEQ    WORD_SPACE            - NO,  SKIP CHANGE
             BSR    CHANGE                - YES, MODIFY MEMORY
             BCLR.B #HEX_ERR,MONSTAT  CLR HEX STATUS BIT ERROR
             BNE    MD_EXIT           IF ERROR EXIT
WORD_SPACE:  MOVE.W #2,D2             SETUP FOR 2 SPACES
             BSR    SPACES            OUTPUT 2 SPACES TO CRT
             MOVE.L END_ADDRESS,D1    GET END ADDRESS
             MOVE.L A2,D0              D0 <- START ADDRESS
             SUB.L  D0,D1              D1<--END ADDR-START ADDR
             BLT    MD_EXIT           IF START > END THEN EXIT
             ANDI.B #$0F,D0           DOES L.S. NIBBLE = 0 ?
             BNE    GETABYTE          - NO, GET ANOTHER BYTE
             BSR    SCROLL            SCROLL PAUSE CHECK
             CMP.B  #ESC,D0           ABORT SCROLL ?
             BEQ    MD_EXIT           - YES, SO EXIT
             BRA    NEWLINE           - NO,  START A NEW LINE
MD_EXIT:     BSR    SCRLF             MOVE CURSOR TO NEXT LINE
             RTS
*
LINE_NUMBER:MOVE.L A2,D0          GET CURRENT ADDRESS
             ROR.L  #8,D0         MOVE M.S. BYTE TO L.S. BYTE
             BSR    OUTPUT_BYTE   DISPLAY BYTE ON CRT
             ROR.L  #8,D0         MOVE M.S. BYTE TO L.S. BYTE
             BSR    OUTPUT_BYTE   DISPLAY BYTE ON CRT
             ROR.L  #8,D0         MOVE M.S. BYTE TO L.S. BYTE
             BSR    OUTPUT_BYTE   DISPLAY BYTE ON CRT
             ROR.L  #8,D0         MOVE M.S. BYTE TO L.S. BYTE
             BSR    OUTPUT_BYTE   DISPLAY BYTE ON CRT
             MOVE.W #4,D2         SETUP FOR 4 SPACES
             BSR    SPACES        OUTPUT 4 SPACES TO CRT
             RTS
*
CHANGE:    MOVE.W #2,D2               SETUP FOR 2 BCKSPCES
           BCLR.B #STRING,MONSTAT     SET FOR NEW STRING
CHGAGIN:   BSR    BACKSPACES          MOVE 2 SP TO THE LEFT
MORE_CHAR:BSR     GETSTRING           GET ANY NEW CHARACTERS
           BCLR.B #STRINGEND,MONSTAT CHECK FOR END OF STR
           BEQ    MORE_CHAR          IF MORE STRING, BRANCH
           MOVE.B BUFFIN,D3          GET STRING LENGTH
           BEQ    NO_ENTRY           IF STR LEN=0
*                                        THEN NO ENTRY
           CMPI.B #2,D3              DOES STRING LEN = 2 ?
           BNE    CHGAGIN             - NO, THEN RE-ENTER
           BSR    GET_DATA           CONVERT BYTE TO HEX
           BTST.B #HEX_ERR,MONSTAT   IS THERE A HEX ERROR ?
           BNE    CHG_EXIT            - YES, EXIT
           MOVE.B D2,-(A2)           BUFFIN(I) <-- HEX
           ADDQ.W #1,A2
```

```
NO_ENTRY: CLR.W  D2
          MOVE.B D3,D2              GET STRING LENGTH
          NEG.W  D2                 D1 <- -(STRING_LENGTH)
          ADDQ.W #4,D2              ADJUST SPACE COUNT
          BSR    SPACES             SPACE TO END OF BYTE
CHG_EXIT  RTS
*
GET_DATA  CLR.L  D2                 CLEAR HEXBUF
          CLR    D4                 CLR WORD FOR DBCC INDEX
          MOVE.B BUFFIN,D4          GET BUFFIN LENGTH
          SUBQ   #1,D4              ADJUST FOR DBCC INST
          LEA    BUFFIN+1,A0        INITIALIZE BUFFING PNTR
DATALOOP  MOVE.B (A0)+,D0           GET CHAR FROM BUFFIN
          BSR    HEX_CONV           CONV ASCII CHAR TO HEX
          BTST.B #HEX_ERR,MONSTAT   IS THERE A HEX ERR ?
          DBNE   D4,DATALOOP        IF MORE CHARS,
*                                      THEN LOOP AGAIN
DATAEXIT  RTS
          END
```

```
*****************************************************************
* THIS PROGRAM CONVERTS THE CONTENTS OF D0<7..0> FROM           *
* ASCII TO HEX AND STORES THE RESULT IN REG D2.                 *
*****************************************************************
* WRITTEN BY DR. LARRY ABBOTT                                   *
*****************************************************************
* FILENAME: HEXCONV.ASM                                         *
*****************************************************************
* VERSION 1.3                                                   *
* REV.  MODIFIED BY       DATE       DESCRIPTION                *
*  A    DAVID M. SENDEK   1 OCT 87   DOCUMENTATION UPGRADE      *
*****************************************************************
* DEFINING MODULES OF EXTERNALLY DECLARED VARIABLES:            *
*   HEX_ERR - MAIN.ASM                                          *
*   MONSTAT - MAIN.ASM                                          *
*****************************************************************

     GLOBAL     HEX_CONV
     EXTERNAL   HEX_ERR,MONSTAT
*
HEX_CONV:  SUB.B  #$30,D0           ADJUST ASCII TO
*                                         HEX BASE
           CMPI.B #9,D0            IS CHARACTER <= 9 ?
           BLS.S  ZERO_CHECK        - YES, CHECK >= 0
           SUB.B  #7,D0            ADJUST FOR A-F
           CMPI.B #$A,D0           IS CHARACTER >= A ?
           BCS.S  HEXERR            - NO, HEX ERROR
           CMPI.B #$F,D0           IS CHARACTER <= F ?
           BHI.S  HEXERR            - NO, HEX ERROR
ZERO_CHECK:CMPI.B #0,D0            IS CHARACTER >= 0 ?
           BMI.S  HEXERR            - NO, HEX ERROR
           BSR    HEX_SHIFT        HEX # INTO
*                                        HEX BUFFER
           BCLR.B #HEX_ERR,MONSTAT CLR HEX CONVERSION
*                                        ERROR STATUS BIT
           BRA.S  HEX_EXIT         EXIT HEX CONVERSION
HEXERR:    BSET.B #HEX_ERR,MONSTAT SET HEX CONVERSION ERROR
*                                        STATUS BIT
HEX_EXIT:  RTS
*
HEX_SHIFT: LSL.B  #4,D0      SHIFT L.S. NIBBLE TO M.S. NIBBLE
           MOVE.W #3,D1          SET FOR INDEX TO 4 SHIFTS
NIBBLE_SHF:LSL.B  #1,D0          SHIFT HEX CHARACTER OUT
           ROXL.L #1,D2          SHIFT INTO HEX BUFFER
           DBF    D1,NIBBLE_SHF BRANCH IF MORE BITS
           RTS
           END
```

```
*****************************************************************
* THE GO ROUTINE EXECUTES A PROGRAM FROM THE MONITOR.           *
* THE FORMAT IS:                                                *
*     GO <start address>,[optional breakpoint]                  *
*****************************************************************
*       WRITTEN BY DR. LARRY ABBOTT                             *
*****************************************************************
*       FILENAME: GO.ASM                                        *
*****************************************************************
* VERSION 1.3                                                   *
* REV.  MODIFIED BY      DATE      DESCRIPTION                  *
*  A    DAVID M. SENDEK  1 OCT 87  DOCUMENTATION UPGRADE        *
*  B    DAVID M. SENDEK  5 OCT 87  BSET,BCLR ASSEMBLY           *
*                                  LANGUAGE CORRECTION          *
*****************************************************************
*  DEFINING MODULES OF EXTERNALLY DECLARED VARIABLES:           *
*  BKPTAB      - MAIN.ASM       ILLMSG      - MESSAGE.ASM       *
*  BKPTMSG     - MESSAGE.ASM    MESSAGE     - MESSAGE.ASM       *
*  BTLEN       - MAIN.ASM       MONSTAT     - MAIN.ASM          *
*  BUFFIN      - MAIN.ASM       OUTPUT_BYTE - BYTEOUT.ASM       *
*  CONTINUE    - MAIN.ASM       SCRLF       - IO_UTIL.ASM       *
*  CMD_DECODE  - DECODER.ASM    SPACE       - MAIN.ASM          *
*  GET_ADDR    - GET_ADDR.ASM   STRING      - MAIN.ASM          *
*  GETSTRING   - GETSTRIN.ASM   STRINGEND   - MAIN.ASM          *
*  HEX_ERR     - MAIN.ASM       SYSTAX      - MAIN.ASM          *
*****************************************************************

     GLOBAL    BKPT,GO
     EXTERNAL  BKPTAB,BKPTMSG,BTLEN,BUFFIN,CONTINUE
     EXTERNAL  GET_ADDR,GETSTRING,HEX_ERR,ILLMSG,MESSAGE
     EXTERNAL  OUTPUT_BYTE,SCRLF,SPACE,STRING,STRINGEND
     EXTERNAL  MONSTAT,SYSTAX,CMD_DECODE
*
TRAP0   EQU     $4E40           OP CODE FOR TRAP #0
*
GO      CMPI.B  #SPACE,(A0)+    IS BUFFIN(X) A SPACE ?
        BNE     GO_ADDR          - NO,  GET GO ADDRESS
        SUBQ.B  #1,BUFFIN        - YES, ADJUST BUFFIN LENGTH
        BRA     GO               - YES, SCAN FOR NEXT SPACE
GO_ADDR SUBQ    #1,A0           ADJUST FOR POST INCREMENT
        BSR     GET_ADDR        A2<-GO ADDR, A3<-BREAKPOINT
        CMPA    #0,A2           IS THERE A START ADDRESS ?
        BEQ     CONTINU          - NO,THIS IS A CONTINUATION
        BCLR.B  #4,MONSTAT      CHECK FOR HEXCONV ERROR
        BNE     GO_EXIT         IF HEX ERROR THEN EXIT
        MOVEA.L SYSTAX,A0          ELSE GET SYSTAX POINTER
        MOVE.L  A2,(A0)         SYSTAX(PC) <-- GO ADDRESS
        CMPA    #0,A3           IS THERE A BREAKPOINT ?
        BEQ     GO_EXIT          - NO, SO EXIT
        LEA     BKPTAB,A0       SET BREAK TAB POINTER
        MOVEA.L A3,(A0)         STORE BREAKPOINT IN TABLE
        MOVE.W  (A3),BTLEN(A0)  STORE INSTRUCTION AT BKPT
```

```
        MOVE.W  #TRAP0,(A3)      STORE ILL INSTRUCT AT BKPT
CONTINU:BSET.B  #CONTINUE,MONSTAT     SET CONTINUE FLAG
GO_EXIT RTS
*
*     THE BREAKPOINT (BKPT) ROUTINE RESTORES THE INSTRUCTION
*     AT THE BREAKPOINT
*
BKPT:   BCLR.B #CONTINUE,MONSTAT  INIT CONTINUATION FLAG
        LEA    BKPTAB,A0          SET BREAK TABLE POINTER
        MOVE.L (A0),A3            GET BKPT ADDRESS
        MOVE.W 16(A0),(A3)        RESTORE INSTRUCTION
        LEA    BKPTMSG,A5         SET BREAKPOINT MESSAGE
BADINST BSR    MESSAGE            PRINT MESSAGE
        MOVE.L A3,D0              GET BKPT ADDRESS
        MOVE.W #3,D3              SET BYTE INDEX
ADDROUT ROL.L  #8,D0              ROTATE D0 BY 1 BYTE
        BSR    OUTPUT_BYTE        CRT <-- D0<0..7>
        DBF    D3,ADDROUT         MORE ADDRESS THE LOOP
        BSR    SCRLF              MOV CURSOR TO STRT OF LINE
        SUBQ.L #2,2(SP)           ADJUST RETURN ADDRESS
        MOVE.L SP,SYSTAX          SAVE POINTER TO RETURN ADDR
EXAMINE BSR    GETSTRING          ALLOWS EXAM AT BKPT
        BCLR.B #STRINGEND,MONSTAT END OF STRING ?
        BEQ    EXAMINE             - NO, SO LOOP
        BCLR.B #STRING,MONSTAT    CLEAR NEW STRING FLAG
        BSR    CMD_DECODE         IF END THEN DECODE
        BCLR.B #CONTINUE,MONSTAT  IS THIS A CONTINUATION ?
        BEQ    EXAMINE             - YES, LOOP AGAIN
        RTE
        END
```

```
*****************************************************************
* THIS FILE CONTAINS PROGRAMMING STUBS TO COMPLETE THE          *
* LINKING PROCESS WHILE BUILDING AND TESTING HIGHER             *
* LEVEL MODULES.                                                *
*****************************************************************
* WRITTEN BY DR. LARRY ABBOTT                                   *
*****************************************************************
* FILENAME: STUB.ASM                                            *
*****************************************************************
* VERSION 1.3                                                   *
* REV. MODIFIED BY      DATE      DESCRIPTION                   *
*  A   DAVID M. SENDEK  1 OCT 87  -DOCUMENTATION UPGRADE        *
*                                 -INCORPORATE PROMPT MSG       *
*****************************************************************
* DEFINING MODULES OF EXTERNALLY DECLARED VARIABLES:            *
*   PROMPT  - MESSAGE.ASM                                       *
*   MESSAGE - MESSAGE.ASM                                       *
*****************************************************************

     GLOBAL    BKPT_LIST,NO_BKPT
     EXTERNAL  PROMPT,MESSAGE

BKPT_LIST:LEA     PROMPT,A5
          BSR     MESSAGE
          RTS
NO_BKPT:  LEA     PROMPT,A5
          BSR     MESSAGE
          RTS
          END
```

```
*****************************************************************
* THIS ROUTINE PRINTS OUT THE CONTENTS OF THE REGISTERS.        *
*****************************************************************
* WRITTEN BY DR. LARRY ABBOTT                                   *
*****************************************************************
* FILENAME: REG.ASM                                             *
*****************************************************************
* VERSION 1.3                                                   *
* REV.  MODIFIED BY      DATE       DESCRIPTION                 *
*  A    DAVID M. SENDEK  1 OCT 87   DOCUMENTATION UPGRADE       *
*****************************************************************
* DEFINING MODULES OF EXTERNALLY DECLARED VARIABLES:            *
* MESSAGE     - MESSAGE.ASM          SCRLF   - IO_UTIL.ASM*
* OUTPUT_BYTE - BYTEOUT.ASM          SPACES  - IO_UTIL.ASM*
* REGMSG      - MESSAGE.ASM          SYSTAX  - MAIN.ASM   *
*****************************************************************


     GLOBAL     REG
     EXTERNAL   MESSAGE,OUTPUT_BYTE,REGMSG
     EXTERNAL   SCRLF,SPACES,SYSTAX


REG      BSR      SCRLF
         LEA      REGMSG,A5   GET POINTER TO MESSAGE
         MOVEA.L  SYSTAX,A2   GET STACK POINTER AT MONITOR
*                                 ENTRY
         SUB.L    #$40,A2     OFFSET OF THE STACK
         MOVE.W   #15,D3      SET REGS CNTR FOR 16 REGS
REGLIST  BSR      MESSAGE     PRINT PART OF REGISTER MESSAGE
         MOVE.W   #3,D4       SET FOR 32-BIT REGISTER
         BSR      REG_DUMP    PRINT CONTENTS OF A REGISTER
         DBF      D3,REGLIST  IF MORE REGS, THEN GO TO
*                                          REGLIST
         BSR      MESSAGE     PRINT "SR ="
         MOVE.W   #1,D4       SET FOR 16-BIT REGISTER
         BSR      REG_DUMP    PR CONTENTS OF STAT REG (SR)
         MOVE.W   #4,D2       SET FOR 4 SPACES
         BSR      SPACES      PRINT 4 SPACES
         BSR      MESSAGE     PRINT "PC ="
         MOVE.W   #3,D4       SET FOR 32-BIT PC REGISTER
         BSR      REG_DUMP    PRINT CONTENTS OF PC REGISTER
         MOVE.W   #1,D2       SET FOR 1 SPACES
         BSR      SPACES      PRINT 1 SPACES
         BSR      MESSAGE     PRINT "(PC) ="
         SUBQ.L   #4,A2
         MOVE.L   (A2),A2
         MOVE.W   #1,D4       SET FOR WORD POINTED TO BY PC
         BSR      REG_DUMP    PRINT CONTENTS OF WD PNTD BY PC
         BSR      SCRLF       FORMAT DISPLAY
         RTS
```

```
*
REG_DUMP MOVE.B (A2)+,D0      GET A BYTE OF THE REG
*                                 FROM APPLICATION PSW
         BSR    OUTPUT_BYTE   OUTPUT BYTE TO CONSOLE
         DBF    D4,REG_DUMP   IF MORE BYTES THEN REG_DUMP
         RTS                     ELSE EXIT
         END
```

```
*****************************************************************
* THIS ROUTINE CHANGES THE CONTENTS OF DESIRED REGISTERS.       *
*****************************************************************
* WRITTEN BY DR. LARRY ABBOTT                                   *
*****************************************************************
* FILENAME: REGCHANG.ASM                                        *
*****************************************************************
* VERSION 1.3                                                   *
* REV.  MODIFIED BY      DATE      DESCRIPTION                  *
*  A    DAVID M. SENDEK  1 OCT 87  DOCUMENTATION UPGRADE        *
*****************************************************************
* DEFINING MODULES OF EXTERNALLY DECLARED VARIABLES:            *
* BUFFIN   - MAIN.ASM              MONSTAT   - MAIN.ASM         *
* GET_ADDR - GET_ADDR.ASM          REG       - REG.ASM          *
*                                  REGERR    - MESSAGE.ASM      *
* HEX_CONV - HEXCONV.ASM           SPACE     - MAIN.ASM         *
* HEX_ERR  - MAIN.ASM              SYSTAX    - MAIN.ASM         *
* MESSAGE  - MESSAGE.ASM           SCRLF     - IO_UTIL.ASM      *
*****************************************************************

     GLOBAL   REGCHANG
     EXTERNAL BUFFIN,GET_ADDR,HEX_CONV,HEX_ERR
     EXTERNAL MESSAGE,MONSTAT,REG,REGERR,SPACE,SYSTAX,SCRLF
*
ESC     EQU     $1B
*
*
REGCHANG: BSR    REG          DISPLAY REGISTERS ON CRT
BLANKSCAN:MOVE.B (A0)+,D0
          SUBQ.B #1,BUFFIN    DECREMENT BUFFIN LENGTH
          CMPI.B #SPACE,D0    DOES BUFFIN(I) CHAR = SPACE ?
          BNE    START_REG      - NO, GET START AND END ADDR
          BRA    BLANKSCAN    CONTINUE SCANNING BUFFIN
START_REG:CMPI.B #ESC,D0      DOES D0 = ESC (ASCII) ?
          BEQ    REG_DONE       - YES, RTS
          CMPI.B #'A',D0           DOES D0 = 'A' ?
          BEQ    REGA           - YES, ADJUST  POINTER
          CMPI.B #'D',D0           DOES D0 = 'D' ?
          BEQ    REGD           - YES, ADJUST POINTER
          CMPI.B #'P',D0           DOES D0 = 'P' ?
          BEQ    REGP           - YES,CK FOR 'C' & ADJUST PNTR
          CMPI.B #'U',D0           DOES D0 = 'U' ?
          BEQ    REGU           - YES, CHECK FOR 'S'
          CMPI.B #'S',D0           DOES D0 = 'S' ?
          BNE    PRINTERR       - NO,PRINT ERR D0 <> A,D,P,U,S
          MOVE.B (A0)+,D0      GET SECOND CHAR OF COMMAND LINE
          SUBQ.B #1,BUFFIN     SUBTRACT 1 FROM BUFFIN
          CMPI.B #'P',D0
          BEQ    REGSP
          CMPI.B #'R',D0
          BEQ    REGREP
          CMPI.B #'S',D0
```

```
            BNE     PRINTERR
            MOVE.L  #-4,D3
            BRA     REGREP
REGA:       MOVE.L  #-32,D3
            BRA     REGFIN
REGD:       MOVE.L  #-64,D3
            BRA     REGFIN
REGP:       MOVE.B  (A0)+,D0
            SUBQ.B  #1,BUFFIN
            CMPI.B  #'C',D0
            BNE     PRINTERR
            MOVE.L  #2,D3
            BRA     REGREP
REGU:       MOVE.B  (A0)+,D0
            SUBQ.B  #1,BUFFIN
            CMPI.B  #'S',D0
            BNE     PRINTERR
            MOVE.L  #-4,D3
            BRA     REGREP
REGSP:      MOVE.L  #-4,D0
            BRA     REGREP
PRINTERR:   LEA     REGERR,A5
            BSR     MESSAGE
            BRA     REG_DONE
REGFIN:     MOVE.B  (A0)+,D0
            SUBQ.B  #1,BUFFIN
            CLR.L   D2
            BSR     HEX_CONV
            BTST.B  #HEX_ERR,MONSTAT
            BNE     PRINTERR
            LSL.L   #2,D2
            ADD.L   D2,D3
REGREP:     LEA     SYSTAX,A1
            MOVE.L  (A1),A1
            ADD.L   D3,A1
RCA:        CMPI.B  #SPACE,(A0)
            BNE     FFF
            ADDQ.W  #1,A0
            SUBQ.B  #1,BUFFIN
            BRA     RCA
FFF:        BSR     GET_ADDR
            MOVE.L  A2,(A1)
            BSR     REG
REG_DONE:   RTS
```

```
*****************************************************************
* DOWNLOAD ALLOWS THE MONITOR TO DOWNLOAD Sxx RECORDS TO ITS*
* RESIDENT 680XX MICROCOMPUTER OVER A SECOND RS-232 PORT.   *
*****************************************************************
* WRITTEN BY DR. LARRY ABBOTT      APRIL 24, 1986           *
*****************************************************************
* FILENAME: DOWNLOAD.ASM                                     *
*****************************************************************
* VERSION 1.3                                                *
* REV. MODIFIED BY        DATE        DESCRIPTION            *
*  A   LARRY ABBOTT     12/18/86  INIT DEBUG PROCESS         *
*  B   DAVID M. SENDEK 1 OCT 87   -DOCUMENTATION UPGRADE     *
*                                 -CORRECT FOR MC68681       *
*  C   DAVID M. SENDEK 5 OCT 87   BCLR,BSET ASSEMBLY         *
*                                      LANGUAGE CORRECTION   *
*  D   DAVID M. SENDEK 4 JAN 88   -CORRECT DOWNLOADING OF    *
*                                   S1,S9 FORMAT RECORDS.    *
*                                   NOTE:FINAL S9 RECORD WILL*
*                                   HAVE A '*' AFTER LAST    *
*                                   CHARACTER IN THE RECORD  *
*****************************************************************
*  DEFINING MODULES OF EXTERNALLY DECLARED VARIABLES:        *
*  CHECKSUM  - MAIN.ASM      CK_SUM    - MAIN.ASM            *
*  ECHO1     - CONSOLE.ASM   ECHO2     - CONSOLE.ASM         *
*  EPROMRNG  - MAIN.ASM      EPROMSG   - MESSAGE.ASM         *
*  EPROMWR   - MAIN.ASM                                      *
*  HEX_CONV  - HEXCONV.ASM   SCRLF     - IO_UTIL.ASM         *
*  HEX_ERR   - MAIN.ASM      SREC_ERR  - MESSAGE.ASM         *
*  HEXMSG    - MESSAGE.ASM   SCANCHR2  - CONSOLE.ASM         *
*  MESSAGE   - MESSAGE.ASM   SPACES    - IO_UTIL.ASM         *
*  MONSTAT   - MAIN.ASM      SRB       - MAIN.ASM            *
*  RECFULL   - MAIN.ASM      GETCHR2   - CONSOLE.ASM         *
*****************************************************************

     GLOBAL    DOWNLOAD
     EXTERNAL  CHECKSUM,CK_SUM,ECHO1,ECHO2,EPROMRNG,EPROMSG
     EXTERNAL  EPROMWR,ESCAPE
     EXTERNAL  HEX_CONV,HEX_ERR,HEXMSG,MESSAGE,MONSTAT
     EXTERNAL  RECFULL,SCRLF,SREC_ERR
     EXTERNAL  SCANCHR2,SPACES
     EXTERNAL  SRB,GETCHR2
*
*
*
DOWNLOAD:BSR     SCANCHR2          DO DUMMY RD TO CLR CHAN B
         BTST.B #RECFULL,SRB       ANY THING ELSE IN CHAN B ?
         BNE.S  DOWNLOAD           - YES,SCAN CHANNEL B AGAIN
         BSR    SCRLF              ECHO CR & LF TO CRT
DOWNLOOP:BCLR.B #HEX_ERR,MONSTAT  CLEAR HEX ERROR FLAG
S_LOOP   BSR    GETCHR2           GET A CHAR FROM DWNLNK PORT
         BTST.B #ESCAPE,MONSTAT    ESC THE DOWNLOAD PROCESS ?
         BNE    DOWNEXIT           - YES, EXIT
```

```
          CMPI.B #'S',D0          IS CHARACTER = 'S' ?
          BNE    S_LOOP           - NO, SEARCH FOR A 'S'
          BSR    ECHO2            ECHO 'S' TO CONSOLE
          MOVE.W #1,D3            SET FOR 16-BIT  ADDR
          BSR    GETCHR2          GET A CHAR FROM DWNLNK PORT
          BSR    ECHO2            ECHO DWNLNK CHAR TO CONSOLE
          CMPI.B #'0',D0          IS THIS A S0 RECORD ?
          BEQ    S_RECORD         - YES, GO TO S RECORD
          CMPI.B #'1',D0          IS THIS A S1 RECORD ?
          BEQ    S_RECORD         - YES, GO TO S RECORD
          CMPI.B #'9',D0          IS THIS A S9 RECORD ?
          BEQ    S9_RECORD        - YES, GO TO S9 RECORD
          ADDQ.W #1,D3            SET FOR 24-BIT ADDR
          CMPI.B #'2',D0          IS THIS A S2 RECORD ?
          BEQ    S_RECORD         - YES, GO TO S RECORD
          ADDQ.W #1,D3            SET FOR A 32-BIT ADDRESS
          CMPI.B #'3',D0          IS THIS A S3 RECORD ?
          BEQ    S_RECORD         - YES, GO TO S RECORD
LOADERR:  LEA    SREC_ERR,A5      IF NO Sxx RECORD
                                    ;THEN 'S RECORD ERROR' MSG
          BSR    ERRMSG

DOWNEXIT:BSR     SCRLF            ECHO CR & LF
          LEA    EPROMSG,A5       SET UP
*                                 "ATTEMPTED WRITE TO EPROM"
          BCLR.B #EPROMWR,MONSTAT WAS THERE A WRITE TO EPROM?
          BEQ.S  ERRMSG           - YES, PRINT ERROR MESSAGE
          RTS

ERRMSG:   BSR    MESSAGE          PRINT ERROR MESSAGE
          RTS

S_RECORD:BSR     SN_RECORD        PROCESS S RECORD
          BCLR.B #HEX_ERR,MONSTAT IF NOT HEX CONVERSION ERR
          BEQ    DOWNLOOP            THEN GET NEXT RECORD
          LEA    HEXMSG,A5           ELSE HEX CONV ERROR MSG
          BRA    ERRMSG                   PRINT ERROR MSG
*
S9_RECORD:BSR    SN_RECORD        PROCESS S RECORD
          BTST.B #HEX_ERR,MONSTAT IF HEX CONVERSION ERROR
          BEQ    DOWNEXIT            THEN TERMINATE XMISSION
          RTS
*
SN_RECORD:CLR.W  D6               SET FOR 1 BYTE
          CLR.B  CK_SUM           CLEAR CHECK SUM
          BSR    GETFIELD         GET DOWNLOAD FIELD
          BTST.B #HEX_ERR,MONSTAT IF HEX CONVERSION ERROR
          BNE.S  SN_EXIT             THEN EXIT SN_RECORD
          MOVE.W D2,D4            D4<-HEXBUFFER (S REC LEN)
          SUB.W  D3,D4            LEN = (S REC LEN) - ADDR
          SUBQ.W #2,D4            ADJST FOR DBF INST & ADDR
          MOVE.W D3,D6            SET ADDRESS SIZE
```

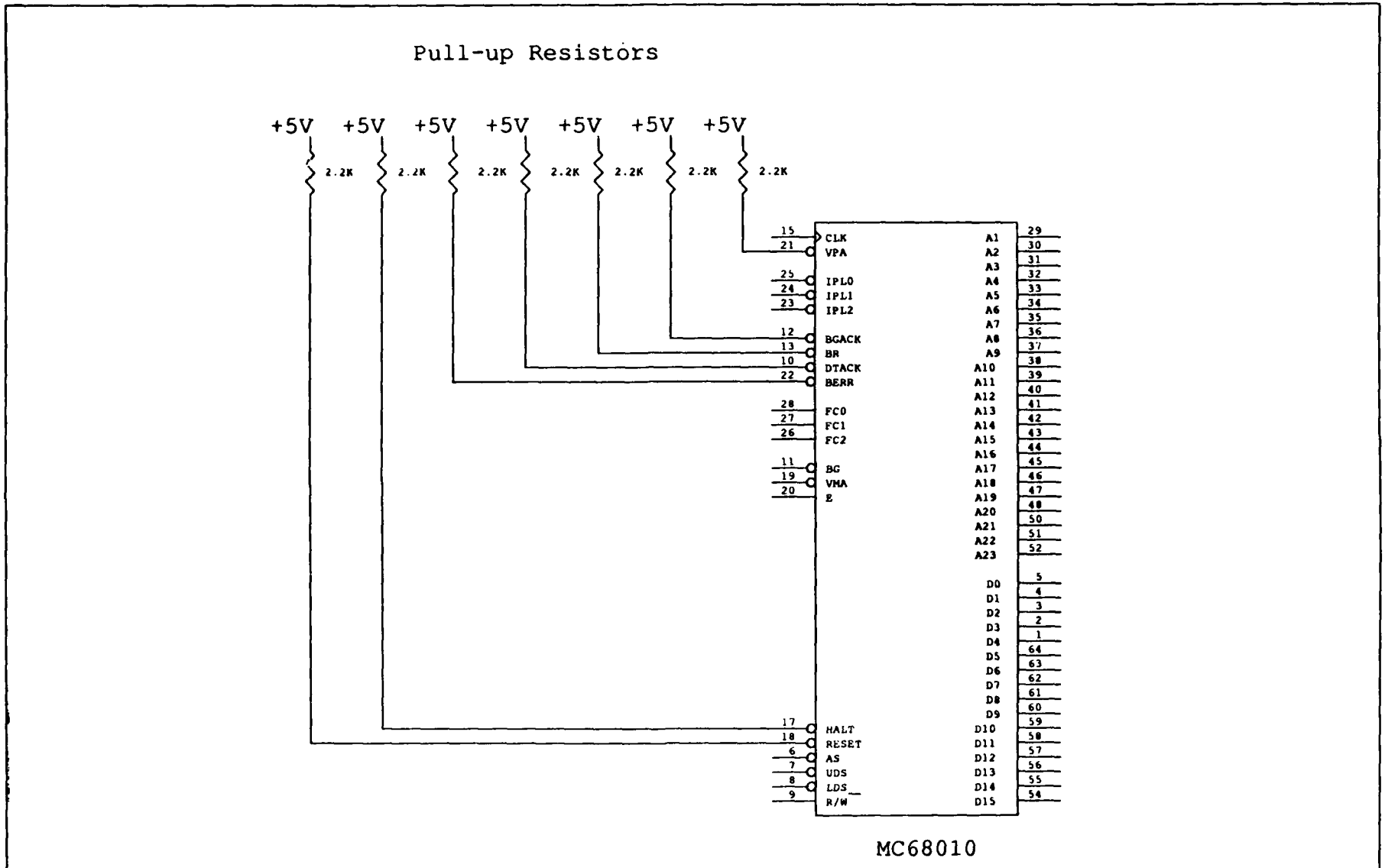
```
            BSR     GETFIELD            GET ADDRESS FIELD
            BTST.B  #HEX_ERR,MONSTAT IF HEX CONVERSION ERROR
            BNE.S   SN_EXIT                THEN EXIT SN_RECORD
            MOVE.L  D2,A0               A0 <-- LOAD ADDRESS
            BSR     DOWN_DATA           GET DOWN LOAD DATA
SN_EXIT     RTS
*
DOWN_DATA   BSR     GETCHR2             GET FIRST CHARACTER
            BSR     ECHO2               ECHO DWNLD CHARACTER
*                                            TO CONSOLE
            CLR.L   D2
            BSR     HEX_CONV            CONVERT CHAR TO HEX
            BTST.B  #HEX_ERR,MONSTAT IF HEX CONVERSION ERROR
            BNE.S   DD_EXIT                THEN EXIT DOWN_DATA
            BSR     GETCHR2             GET SECOND CHARACTER
            BSR     ECHO2               ECHO DWNLD CHAR TO CONSOLE
            CMPA.L  #EPROMRNG,A0        IS THIS A WRITE TO EPROM?
            BLS.S   EPROMERR             - YES, GO TO EPROMERR
            BSR     HEX_CONV            CONVERT CHARACTER TO HEX
            MOVE.B  D2,(A0)             LOAD BYTE INTO MEMORY
            BRA.S   CHK_SUM
EPROMERR    BSET.B  #EPROMWR,MONSTAT    FLAG EPROM WRITE
CHK_SUM     ADDQ.L  #1,A0               INCREMENT MEM LOAD ADDR
            TST.W   D4                  ARE NXT CHARS CHECK SUM ?
            BEQ.S   LOOP_END             - YES,DONT ADD TO CHK SUM
            ADD.B   D2,CK_SUM           ADD THIS BYTE TC CHK SUM
LOOP_END    DBF     D4,DOWN_DATA        IF MORE DATA THEN LOOP
            NOT.B   CK_SUM              COMPLEMENT CHECK SUM
            MOVE.B  -(A0),D2            GET COMPUTED CHECK SUM
            CMP.B   CK_SUM,D2           COMP CALC'S AND
*                                       XMIT CHK SUMS
            BEQ.S   ERRCHECK            IF CHECK SUMS AGREE
*                                          THEN EXIT DOWNLOAD
            MOVE.L  MONSTAT,D3
            BSET.L  #CHECKSUM,D3        SET FLAG IF CHECK SUM ERR
            MOVE.L  D3,MONSTAT
            BRA.S   ERR_MARK
ERRCHECK    BTST.B  #EPROMWR,MONSTAT    A WRITE TO EPROM ?
            BEQ.S   DD_EXIT              - NO,  EXIT
ERR_MARK    MOVE.W  #'*',D0              - YES, MARK ERROR WITH *
            BSR     ECHO1
DD_EXIT     BSR     SCRLF               ECHO CR & LF
            RTS
*
GETFIELD    CLR.L   D2                  CLEAR HEX BUFFER
LOOPINIT    MOVE.W  #1,D5               SET COUNT TO
*                                           PACK 2 NIBBLES
GF_LOOP     BSR     GETCHR2             GET DOWNLOAD CHARACTER
            BSR     ECHO2               ECHO DOWNLOAD CHARACTER
*                                            TO CONSOLE
            BSR     HEX_CONV            CONVERT ASCII CHAR TO HEX
            BTST.B  #HEX_ERR,MONSTAT IF HEX CONVERSION ERROR
```

```
            BNE.S   GF_EXIT             THEN EXIT GET FIELD
            DBF     D5,GF_LOOP        GET SECOND NIBBLE
            ADD.B   D2,CK_SUM         COMPUTE CHECK SUM
            DBF     D6,LOOPINIT       IF MORE CHARS THEN LOOP
GF_EXIT     RTS                          ELSE EXIT
            END
```

```
*****************************************************************
* THIS ROUTINE IS VECTORED TO BY ALL EXCEPTIONS THAT            *
* LACK A DEFINITE EXCEPTION SERVICE ROUTINE.                    *
*****************************************************************
* WRITTEN BY DR. LARRY ABBOTT                                   *
*****************************************************************
* FILENAME: UNUSED.ASM                                          *
*****************************************************************
* VERSION 1.3                                                   *
* REV.  MODIFIED BY       DATE       DESCRIPTION                *
*  A    DAVID M. SENDEK   1 OCT 87   -DOCUMENTATION UPGRADE     *
*                                    -INCORPORATE A PROMPT      *
*****************************************************************
* DEFINING MODULES OF EXTERNALLY DECLARED VARIABLES:            *
*    OUTPUT_BYTE - BYTEOUT.ASM     MESSAGE - MESSAGE.ASM        *
*    REG         - REG.ASM         SCRLF   - IO_UTIL.ASM        *
*    SYSTAX      - MAIN.ASM        USEMSG  - MESSAGE.ASM        *
*    PROMPT      - MESSAGE.ASM                                  *
*****************************************************************

     GLOBAL    UNUSED
     EXTERNAL  OUTPUT_BYTE,MESSAGE,REG,SCRLF,SYSTAX,USEMSG
     EXTERNAL  PROMPT

UNUSED:MOVEM.L SP,SYSTAX          SAVE POINTER TO APPLICATION
*                                      REGISTERS
       MOVEM.L A0-A7/D0-D7,-(SP) SAVE ALL REGISTERS
       BSR     SCRLF             MOVE CURSOR TO NEXT LINE
       LEA     USEMSG,A5         SET MSG POINTER TO MONMSG
       BSR     MESSAGE           CRT<-UNUSED EXCEPTION MSG
       MOVE.L  SYSTAX,A5         GET TOP OF STACK AT ENTRY
       ADDQ.L  #6,A5             POINT TO STACK FORMAT WORD
       MOVE.B  (A5)+,D0          GET FORMAT.HIGH
       BSR     OUTPUT_BYTE       OUTPUT FORMAT.HIGH
       MOVE.B  (A5),D0           GET FORMAT.LOW
       BSR     OUTPUT_BYTE       OUTPUT FORMAT.LOW
       BSR     SCRLF             MOVE CURSOR TO NEXT LINE
       BSR     REG               DISPLAY REGISTERS
       MOVEM.L (SP)+,A0-A7/D0-D7 RESTORE ALL REGISTERS
       RTE
       END
```

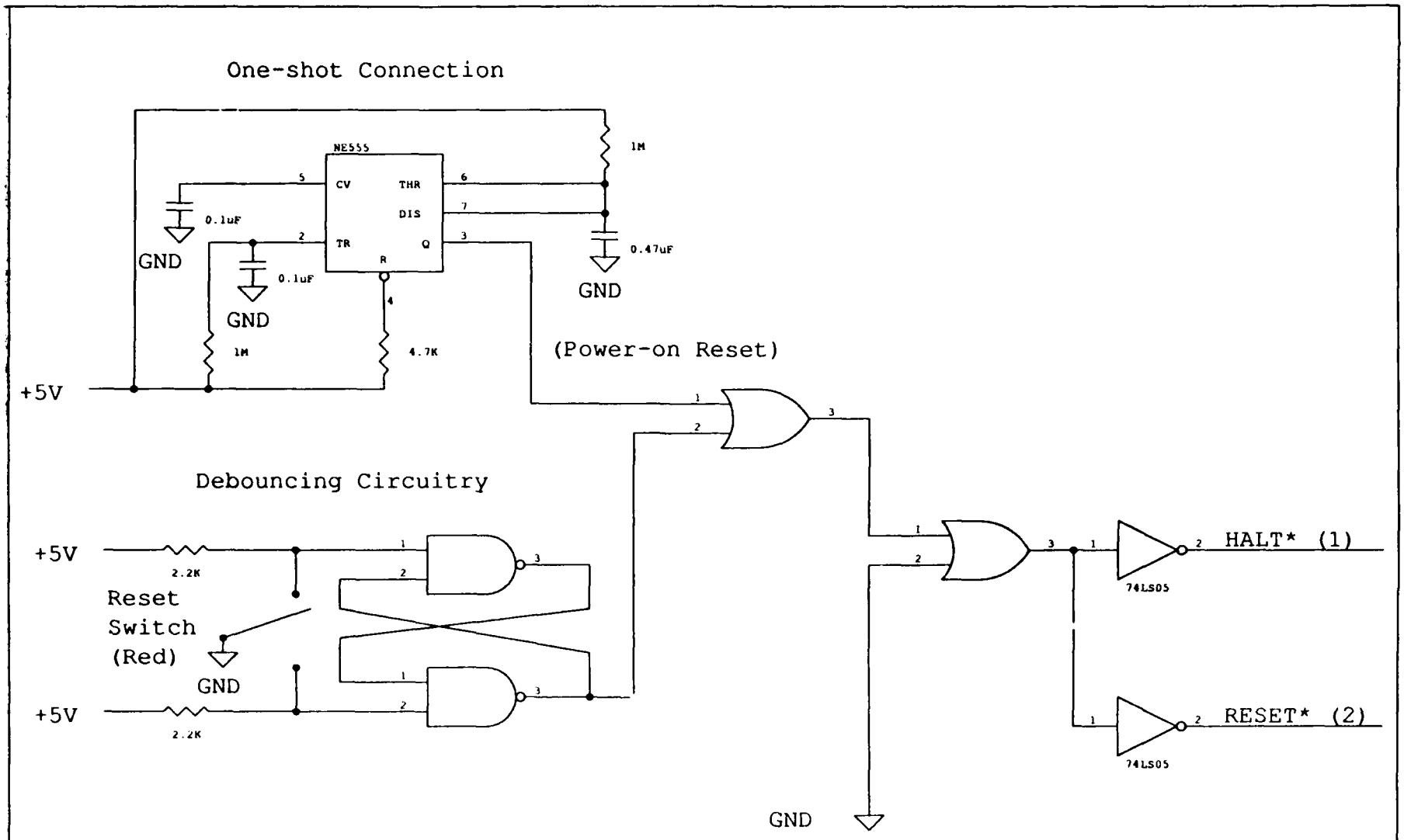
APPENDIX C: MINIMAL SYSTEM DIAGRAMS

The figures (Figs. C.1 through C.8) contained in this appendix are discussed in Chapter IV. These figures were created using the OrCAD/SDT III computer-aided design (CAD) tool. Each signal's source(s) and/or destination(s) are noted on the diagrams. It is, however, the integration of these various components into a minimal system that comprises the work that is original to this thesis.



TITLE: Minimal System MC68010 Microprocessor Circuitry

Figure: C.1



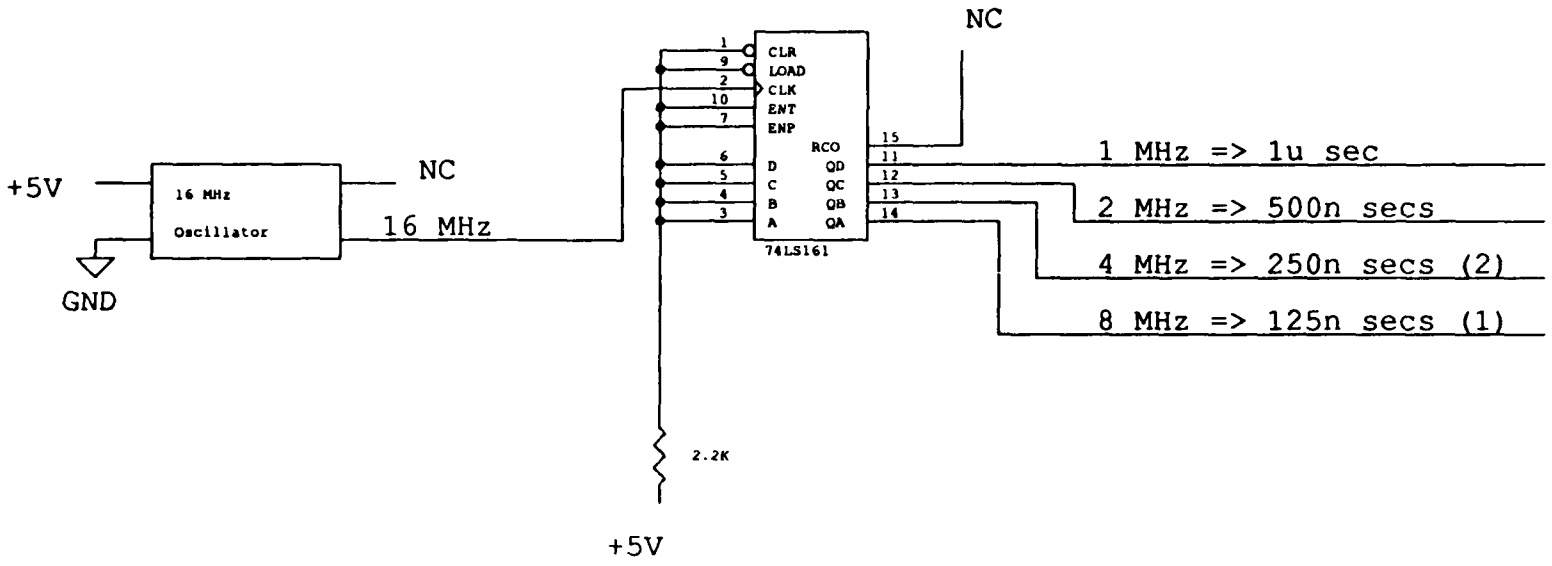
COMMENT:

(1) - To CPU

(2) - To CPU and DUART Circuitry

TITLE: Minimal System HALT* and RESET* Generation Circuitry

Figure: C.2



COMMENTS:

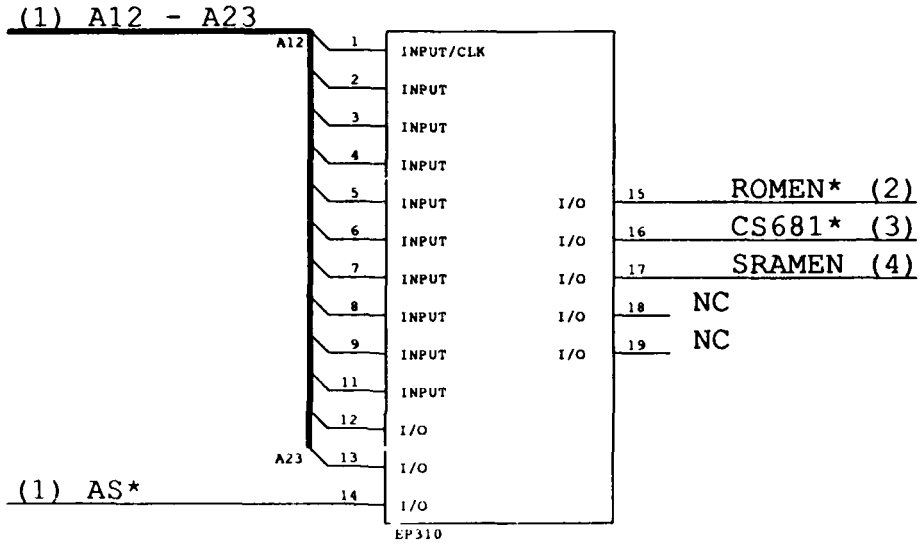
(1) - To CPU

(2) - To DTACK* and BERR* Circuitry

NC = Not Connected

TITLE: Minimal System Clock Generation Circuitry

Figure: C.3

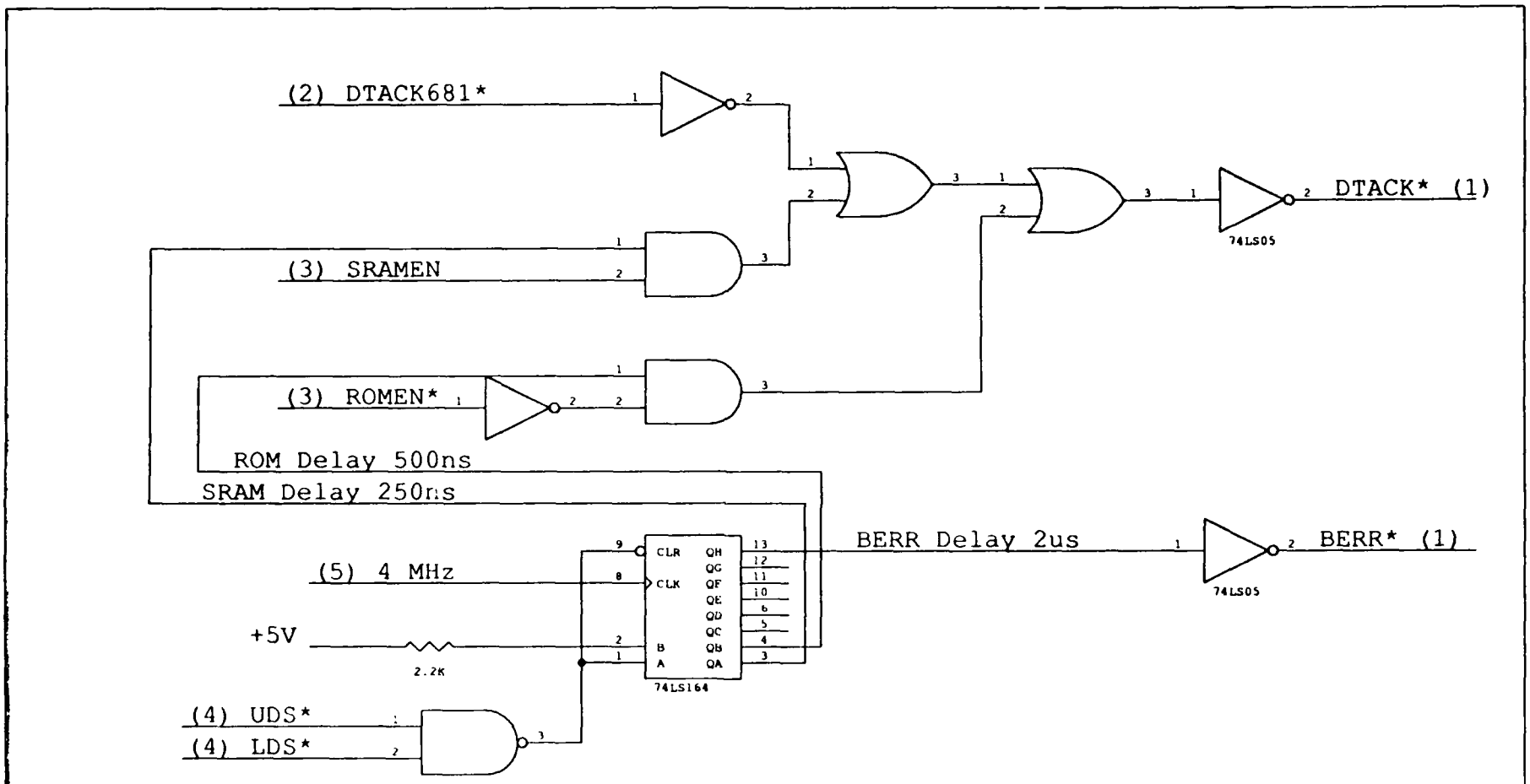


COMMENTS:

(1) - From CPU
(2) - ROM Enable; To ROM Circuitry and DTACK* Circuitry
(3) - Chip Select MC68681 DUART; To DUART Circuitry
(4) - SRAM Enable; To SRAM Circuitry and DTACK* Circuitry
NC = Not Connected

TITLE: Minimal System Address Decode Circuitry

Figure: C.4

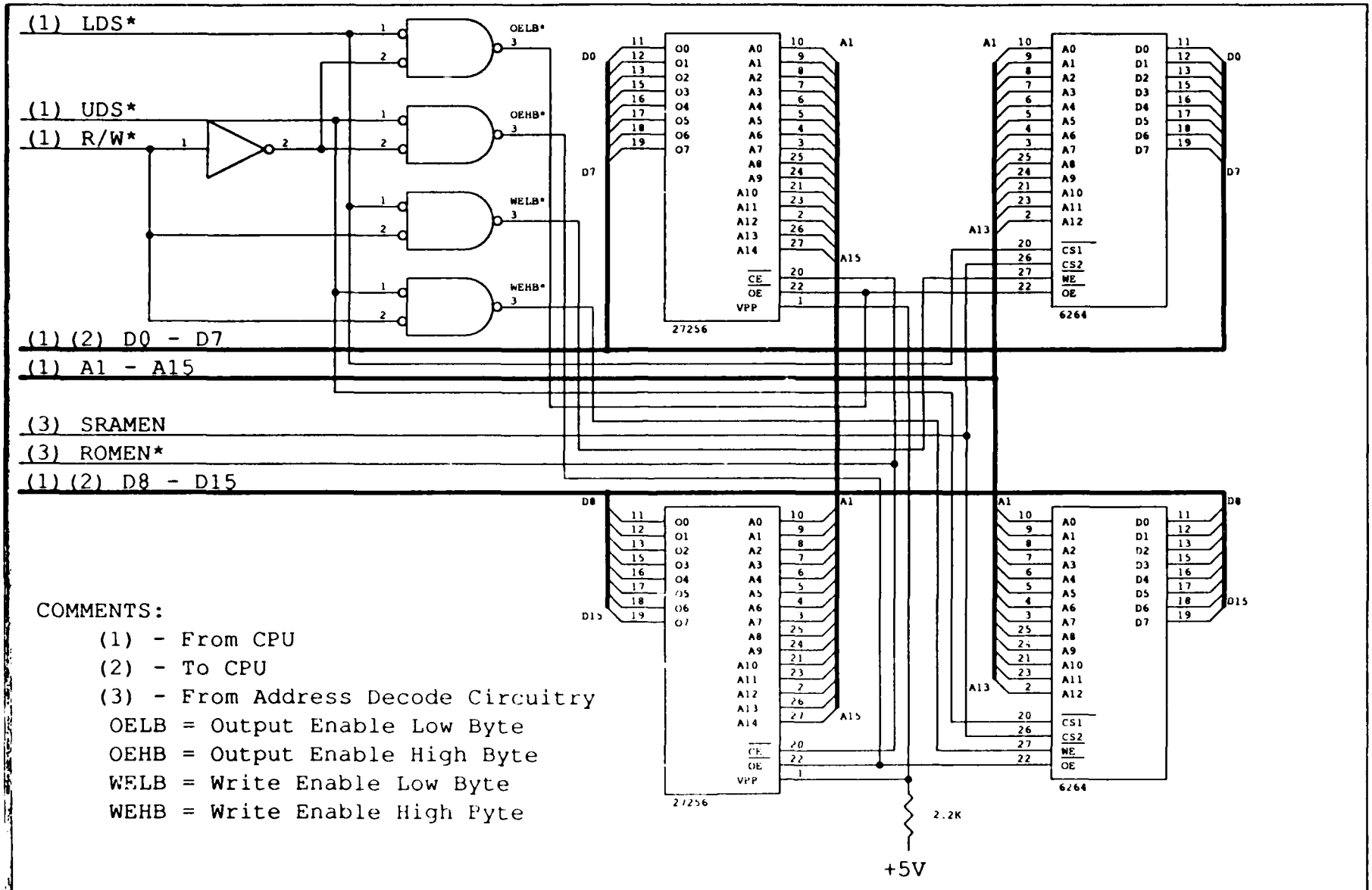


COMMENTS:

(1) - To CPU
(2) - From DUART Circuitry
(3) - From Address Decode Circuitry
(4) - From CPU
(5) - From Clock Circuitry

TITLE: Minimal System DTACK* and BERR* Generation Circuitry

Figure: C.5

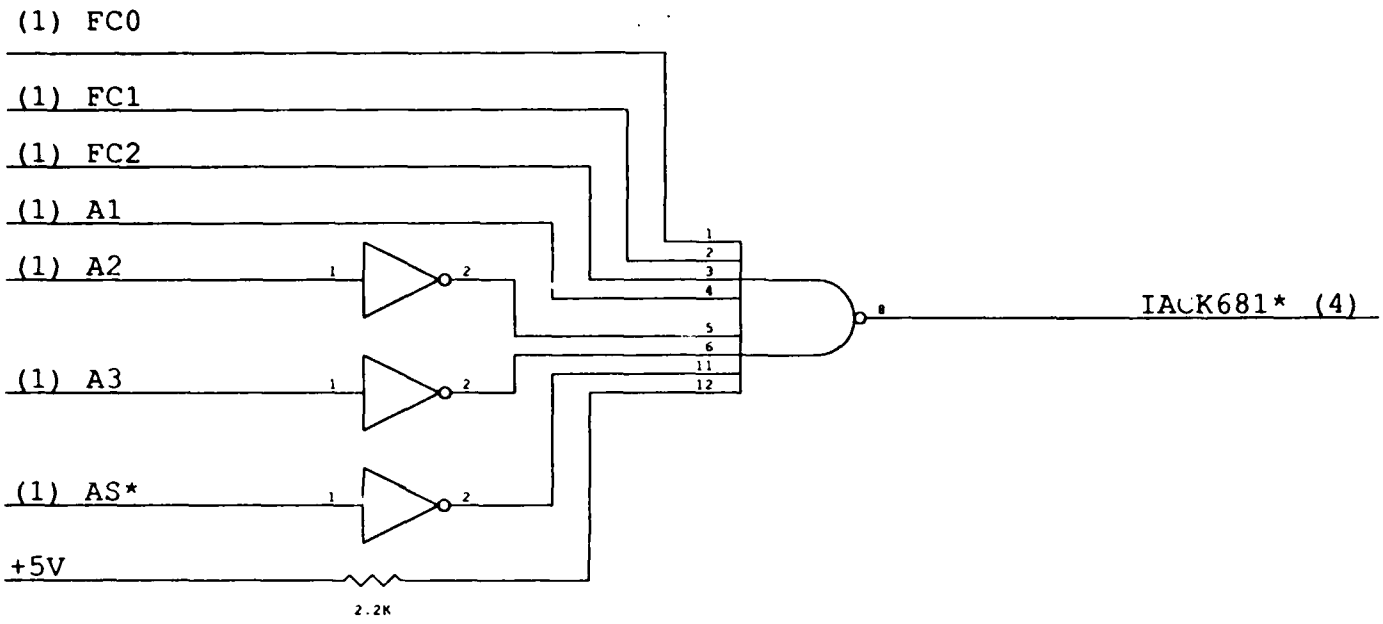


COMMENTS:

(1) - From CPU
(2) - To CPU
(3) - From Address Decode Circuitry
OELB = Output Enable Low Byte
OEHB = Output Enable High Byte
WELB = Write Enable Low Byte
WEHB = Write Enable High Pyte

TITLE: Minimal System EPROM and SRAM Circuitry

Figure: C.6

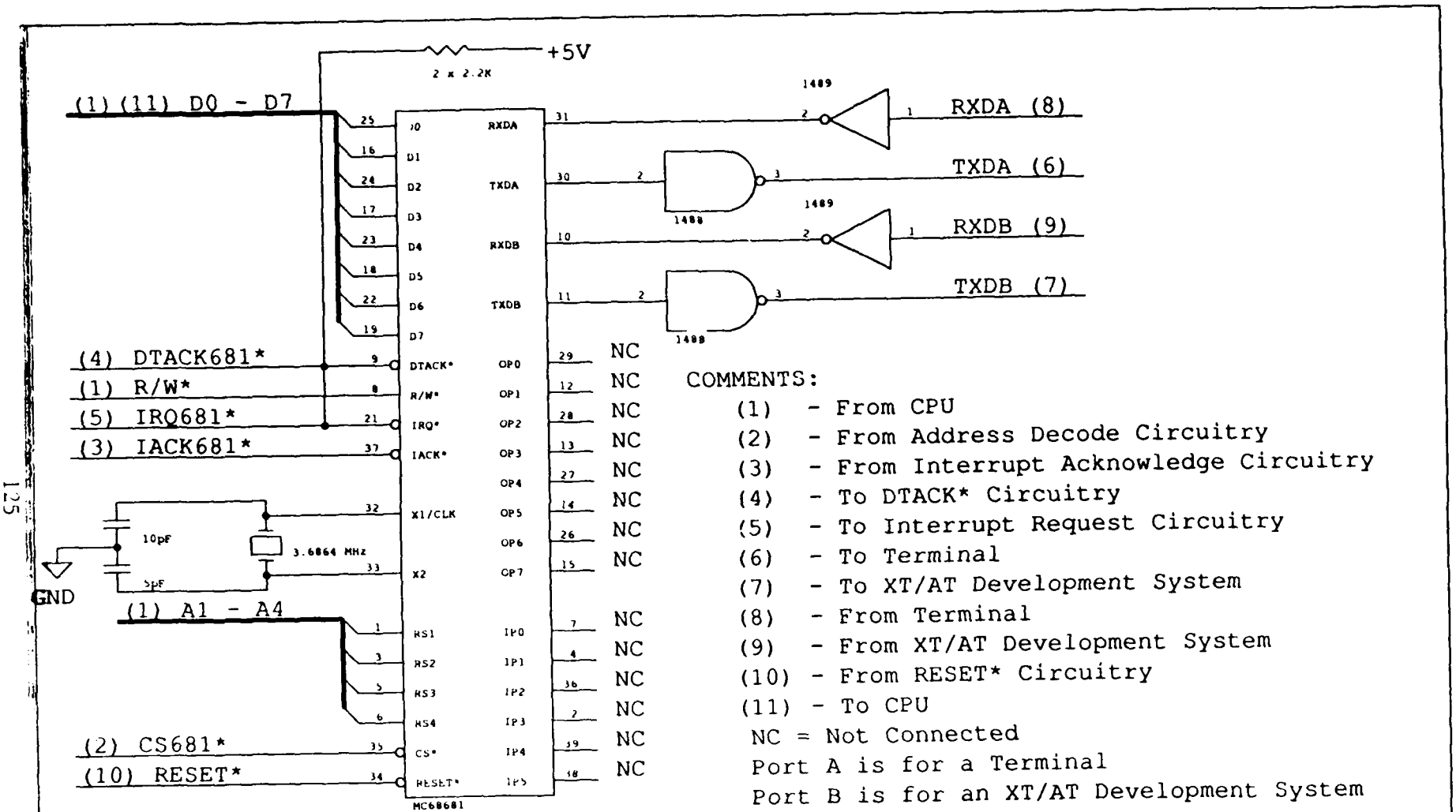


COMMENTS:

(1) - From CPU
(2) - To CPU
(3) - From DUART Circuitry
(4) - To DUART Circuitry
MSB = Most Significant Bit
LSB = Least Significant Bit

TITLE: Minimal System Interrupt Request and Interrupt Acknowledge Circuitry

Figure: C.7



COMMENTS:

(1) - From CPU
(2) - From Address Decode Circuitry
(3) - From Interrupt Acknowledge Circuitry
(4) - To DTACK* Circuitry
(5) - To Interrupt Request Circuitry
(6) - To Terminal
(7) - To XT/AT Development System
(8) - From Terminal
(9) - From XT/AT Development System
(10) - From RESET* Circuitry
(11) - To CPU

NC = Not Connected

Port A is for a Terminal

Port B is for an XT/AT Development System

TITLE: Minimal System Dual-port Receiver Transmitter Serial Port Circuitry

Figure: C.8

APPENDIX D: MINIMAL SYSTEM'S PROGRAMMABLE LOGIC DEVICE SOURCE CODE

In order to reduce the chip count, Altera EP310 erasable programmable logic devices (EPLDs) were used within the minimal system. Abel, a logic software design tool by Data I/O Corporation, was used to program Altera EP310 EPLDs [Ref. 16:pp. 2-57 - 2-62]. Abel files provides a high-level representation of the logic to be implemented on the EP310s. The EP310 comes in a 20-pin package. Nine pins are used strictly for input logic; one pin can be used for input logic or as a clocked input; eight pins can be used for input logic or output logic; the remaining two pins are used for Vcc input and ground input.

The following Abel modules were implemented:

- minimal_system_address_decoder
- dtack_and_bus_error_generation
- output_enable_write_enable
- interrupt_controller

```
"
"         THIS FILE USES DATA I/O'S ABEL DESIGN LANGUAGE
"         TO GENERATE A JEDEC FILE TO PROGRAM AN ALTERA
"         EP310 ERASABLE PROGRAMMABLE LOGIC DEVICE (EPLD).
"

MODULE minimal_system_address_decoder  FLAG '-X0'
TITLE  '68010 ADDRESS DECODER FOR THE MINIMAL SYSTEM'

u61 DEVICE 'E0310'; "Abel V2 must be used for this device.

"DEFINE LABELS ASSOCIATED WITH INPUT AND OUTPUT PINS
"          FOR THE EP310
"       -  INPUT PINS
           a12,a13,a14,a15,a16,a17,a18,a19,a20,a21,a22,a23,
               as PIN 1,2,3,4,5,6,7,8,9,11,12,13,15;

"       -  OUTPUT PINS
           cs681,romen,sramen PIN 16,18,19;

"ASSIGNMENT STATEMENTS
           h = 1;      "HIGH
           l = 0;      "LOW
           x = .X.;    "DONT CARE

    ramaddr = [a23,a22,a21,a20,a19,a18,a17,a16,a15,a14,
                    x,x,x,x,x,x,x,x,x,x,x,x,x,x];
    romaddr = [a23,a22,a21,a20,a19,a18,a17,a16,x,x,x,x,
                    x,x,x,x,x,x,x,x,x,x,x,x,x,x];
  duartaddr = [a23,a22,a21,a20,a19,a18,a17,a16,a15,a14,
                    a13,a12,x,x,x,x,x,x,x,x,x,x,x,x];

"DEFINE EQUATIONS AS PER MEMORY MAP
"       ! = INVERSION
"       & = AND
"       # = OR
EQUATIONS
sramen= (ramaddr >= ^h010000)&(ramaddr <= ^h013FFF)&!as;
!cs681= (duartaddr >= ^h7F7000)&(duartaddr <= ^h7F7FFF)&!as;
!romen = (romaddr <=^h00FFFF)&!as;

END minimal_system_address_decoder
```

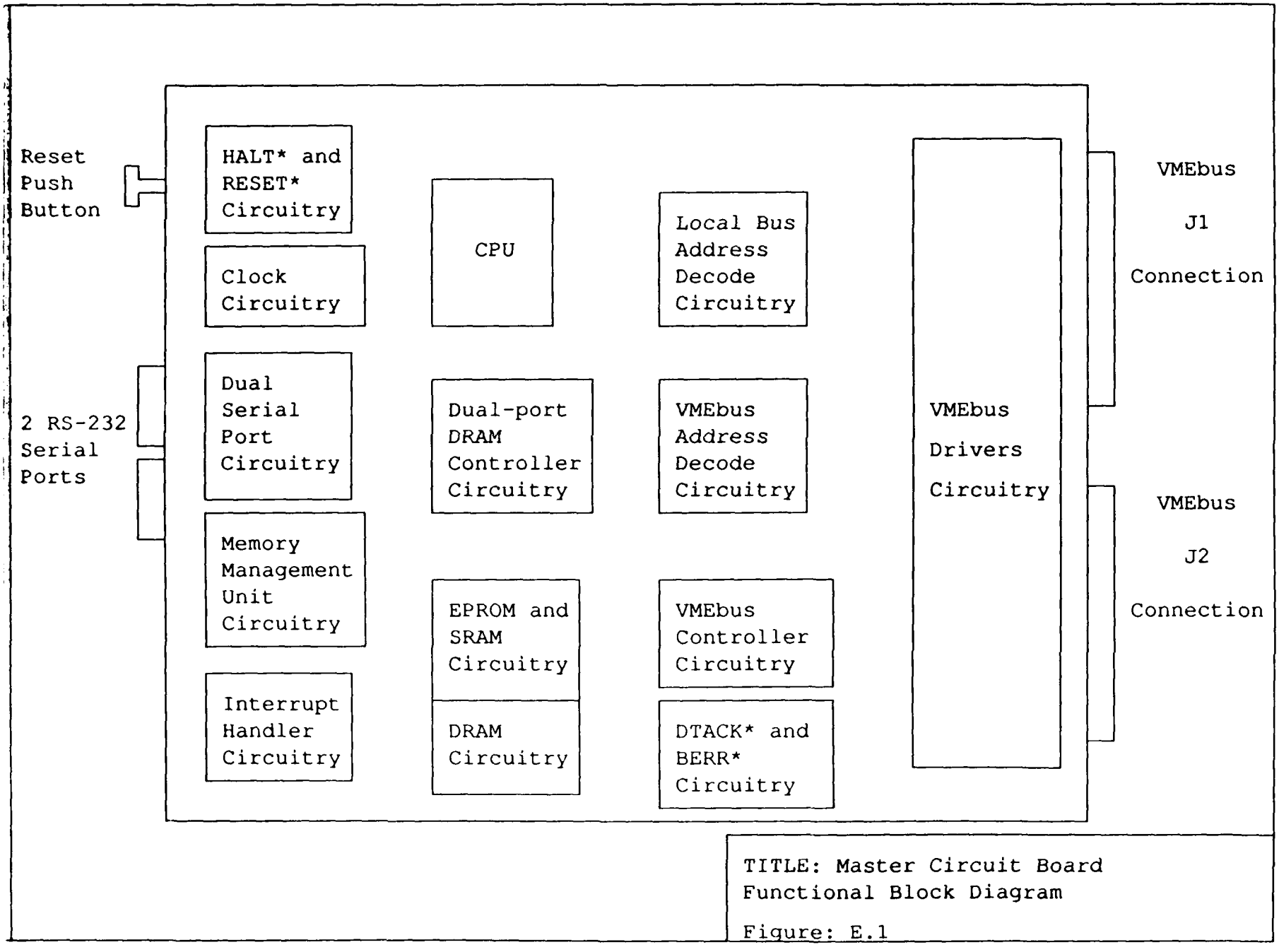
```
"
"         THIS FILE USES DATA I/O'S ABEL DESIGN LANGUAGE
"         TO GENERATE A JEDEC FILE TO PROGRAM AN ALTERA
"         EP310 ERASABLE PROGRAMMABLE LOGIC DEVICE (EPLD).
"

MODULE dtack_and_bus_error_generation  FLAG '-X0'
TITLE  'DTACK AND BUS ERROR GENERATION FOR THE MINIMAL SYSTEM'

u64 DEVICE 'E0310'; "Abel V2 must be used for this device.

"DEFINE LABELS ASSOCIATED WITH INPUT AND OUTPUT PINS
"          FOR THE EP310
"       -  INPUT PINS
           berr_delay,rom_delay,sram_delay,romen,
               dtack681,sramen PIN 1,8,9,11,13,16;

"       -  OUTPUT PINS
           dtack,berr PIN 18,19;

"ASSIGNMENT STATEMENTS
           h = 1;      "HIGH
           l = 0;      "LOW
           x = .X.;    "DONT CARE


"DEFINE EQUATIONS
"  NOTE:   ! = INVERSION
"          & = AND
"          # = OR
EQUATIONS
dtack=(!dtack681)#(sramen&sram_delay)#(!romen&rom_delay);
berr = berr_delay;


END dtack_and_bus_error_generation
```

```
"
"         THIS FILE USES DATA I/O'S ABEL DESIGN LANGUAGE
"         TO GENERATE A JEDEC FILE TO PROGRAM AN ALTERA
"         EP310 ERASABLE PROGRAMMABLE LOGIC DEVICE (EPLD).
"

MODULE output_enable_write_enable  FLAG '-X1'
TITLE  'SRAM WRITE ENABLE AND SRAM AND ROM OUTPUT ENABLES FOR
            THE MINIMAL SYSTEM'

u63 DEVICE 'E0310';"Abel V2 must be used for this device.

"DEFINE LABELS ASSOCIATED WITH INPUT AND OUTPUT PINS
"          FOR THE EP310
"       -  INPUT PINS
           rw,win,uds,lds,mas,as,pudsi,pldsi PIN 1,2,3,4,5,6,8,9;

"       -  OUTPUT PINS
           oelb,weu34,weu35,oehb,pudso,pldso,pas
                PIN 13,14,15,16,17,18,19;

"ASSIGNMENT STATEMENTS
          h = 1;      "HIGH
          l = 0;      "LOW
          x = .X.;    "DONT CARE


"DEFINE EQUATIONS
" NOTE:     ! = INVERSION
"           & = AND
"           # = OR
"          rw = read
"         !rw = write
EQUATIONS
!weu34 = !rw & !pldsi;
!weu35 = !rw & !pudsi;
!oehb  = rw & !pudsi;
!oelb  = rw & !pldsi;

END output_enable_write_enable
```

```
"
"         THIS FILE USES DATA I/O'S ABEL DESIGN LANGUAGE
"         TO GENERATE A JEDEC FILE TO PROGRAM AN ALTERA
"         EP310 ERASABLE PROGRAMMABLE LOGIC DEVICE (EPLD).
"

MODULE interrupt_controller  FLAG '-X1'
TITLE  'INTERRUPT CONTROLLER FOR THE MINIMAL SYSTEM'

"THIS IS NOT UPWARDS COMPATIBLE FOR THE FULLY INTEGRATED SYSTEM

u00 DEVICE 'E0310'; "Abel V2 must be used for this device.

"DEFINE LABELS ASSOCIATED WITH INPUT AND OUTPUT PINS
"        FOR THE EP310
"     -  INPUT PINS
        a1,a2,a3,as,irq681,fc1,fc2,fc3 PIN 1,2,3,4,5,6,7,8;

"     -  OUTPUT PINS
        ipl0,ipl1,ipl2,irack681 PIN 16,17,18,19;

"ASSIGNMENT STATEMENTS
          h = 1;      "HIGH
          l = 0;      "LOW
          x = .X.;    "DONT CARE


"DEFINE EQUATIONS
" NOTE:     ! = INVERSION
"           & = AND
"           # = OR
EQUATIONS
!irack681 = a1 & !a2 & !a3 & !as & fc1 & fc2 & fc3;
ipl0      = h;
ipl1      = h;
ipl2      = irq681;

END interrupt_controller
```

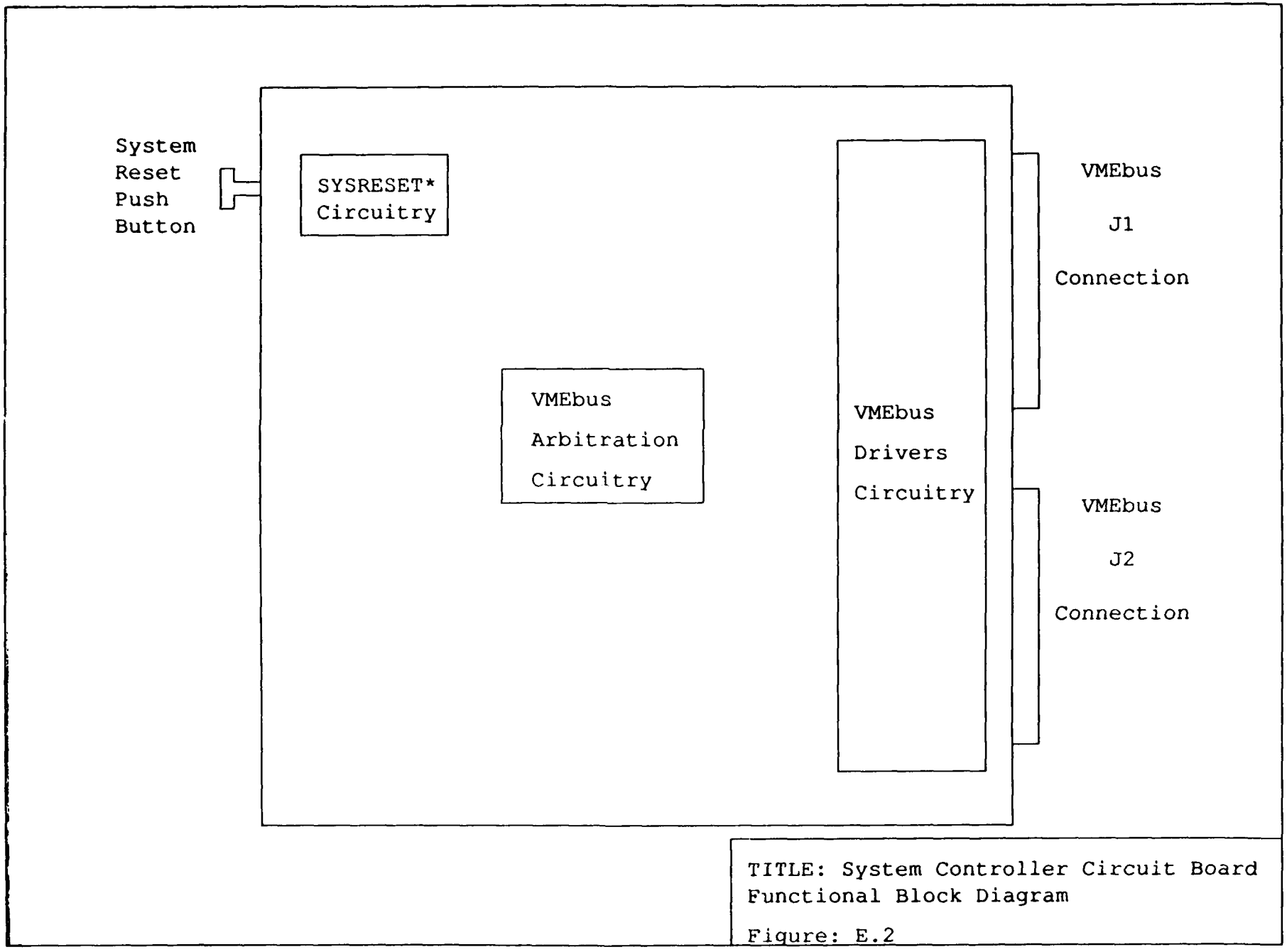
APPENDIX E: SYSTEM DIAGRAMS

In this appendix are the wiring diagrams which implement the master circuit board subsystem and system controller subsystem which are discussed in Chapter IV. These diagrams were produced by the OrCAD/SDT III computer-aided design (CAD) tool. It is, however, the integration of these various components into a multi-processor system that comprises the work that is original to this thesis.



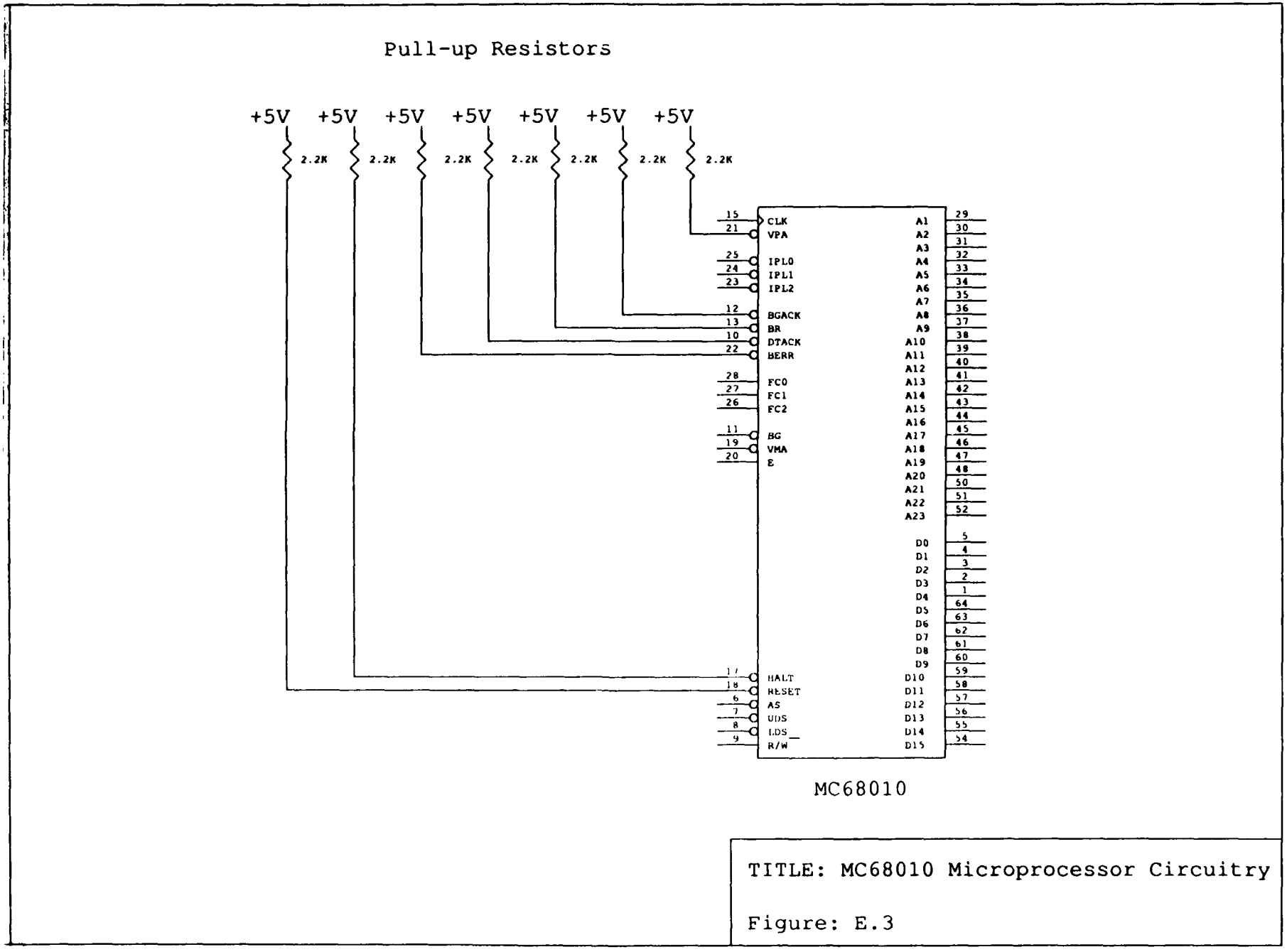
TITLE: Master Circuit Board Functional Block Diagram

Figure: E.1



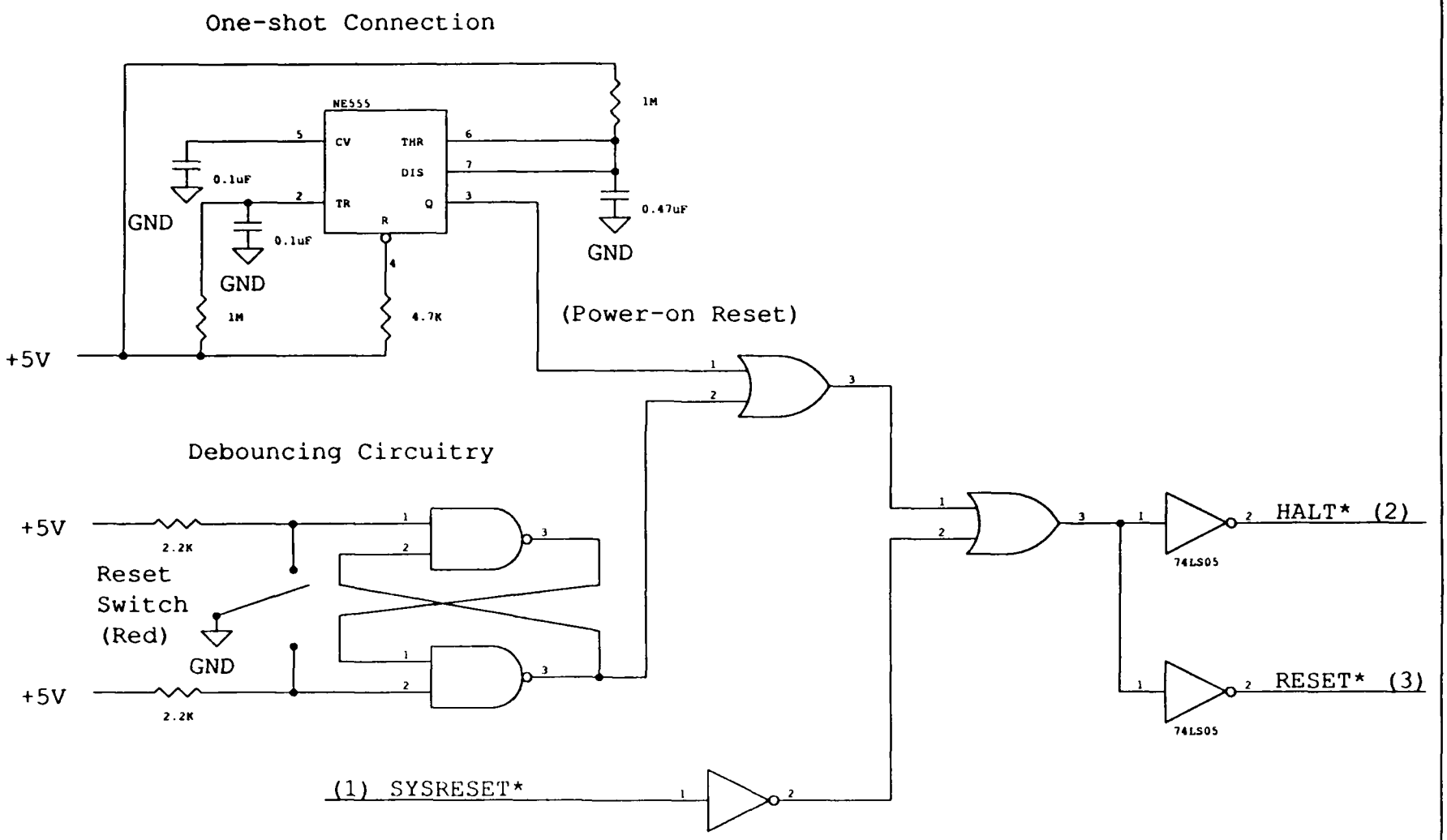
TITLE: System Controller Circuit Board
Functional Block Diagram

Figure: E.2



TITLE: MC68010 Microprocessor Circuitry

Figure: E.3

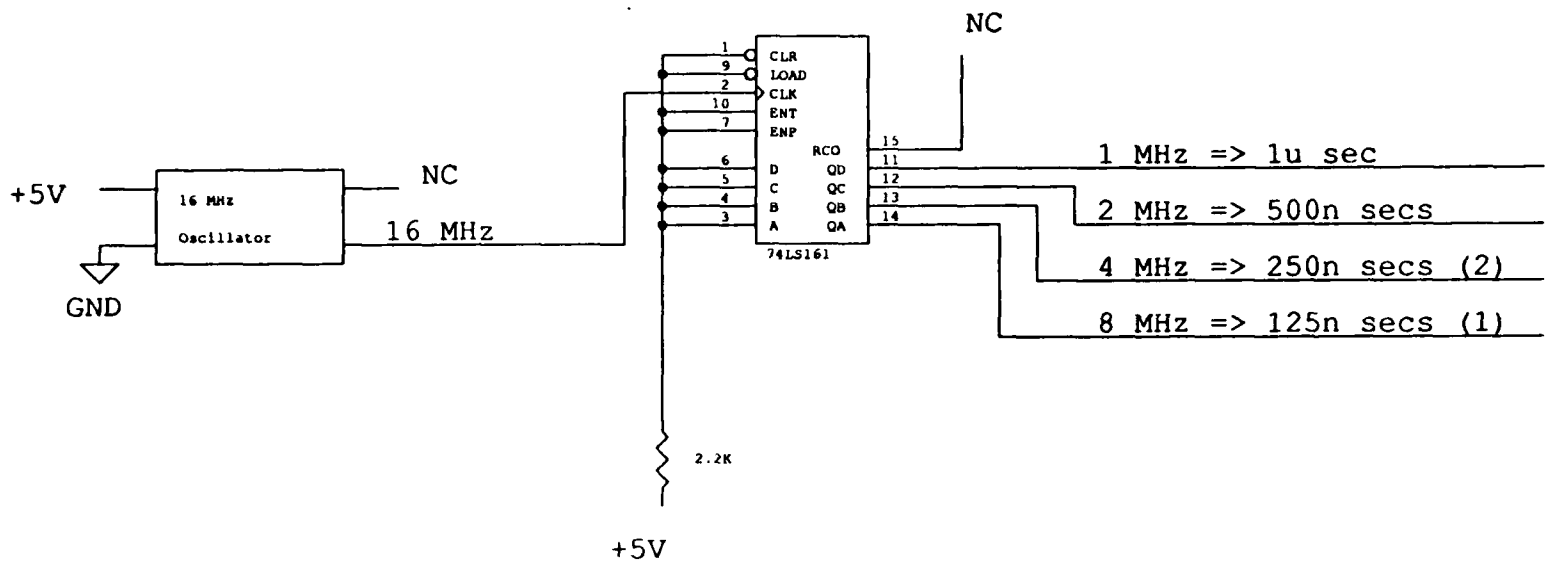


COMMENTS:

(1) - From VMEbus
(2) - To CPU
(3) - To CPU, MMU Circuitry, DUART Circuitry, Interrupt Handler Circuitry and VMEbus Controller Circuitry

TITLE: HALT* and RESET* Generation Circuitry

Figure: E.4



COMMENTS:

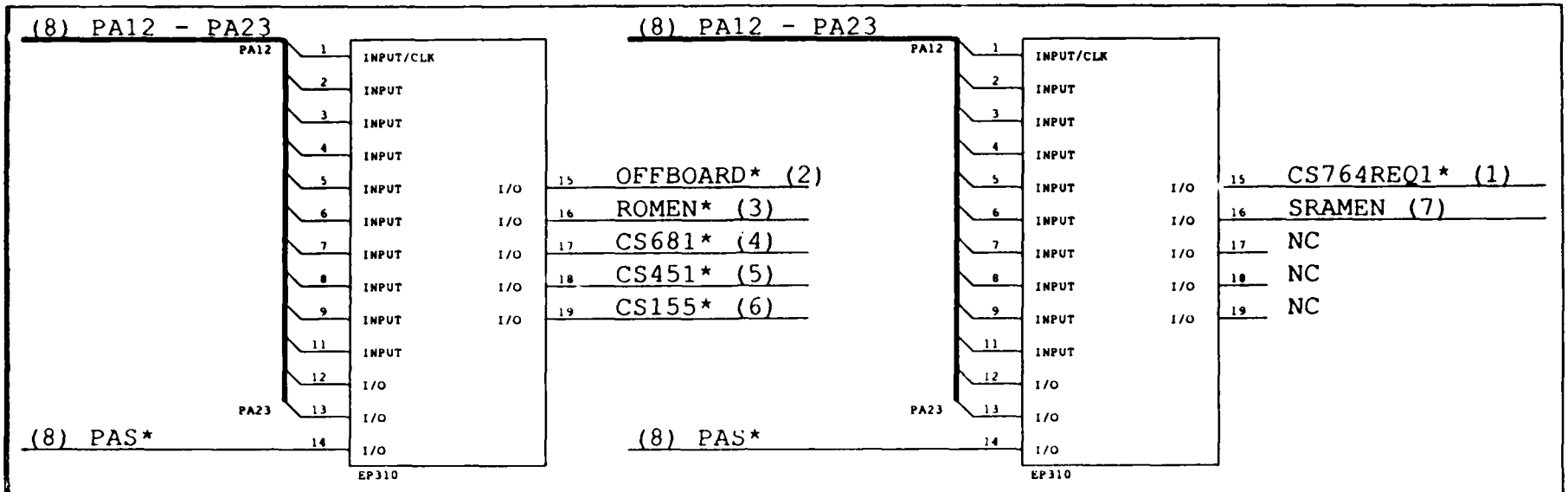
(1) - To CPU, MMU Circuitry, Interrupt Handler Circuitry, Dual-port DRAM Controller Circuitry and VMEbus Controller Circuitry

(2) - To DTACK* and BERR* Circuitry

NC = Not Connected

TITLE: Clock Generation Circuitry

Figure: E.5

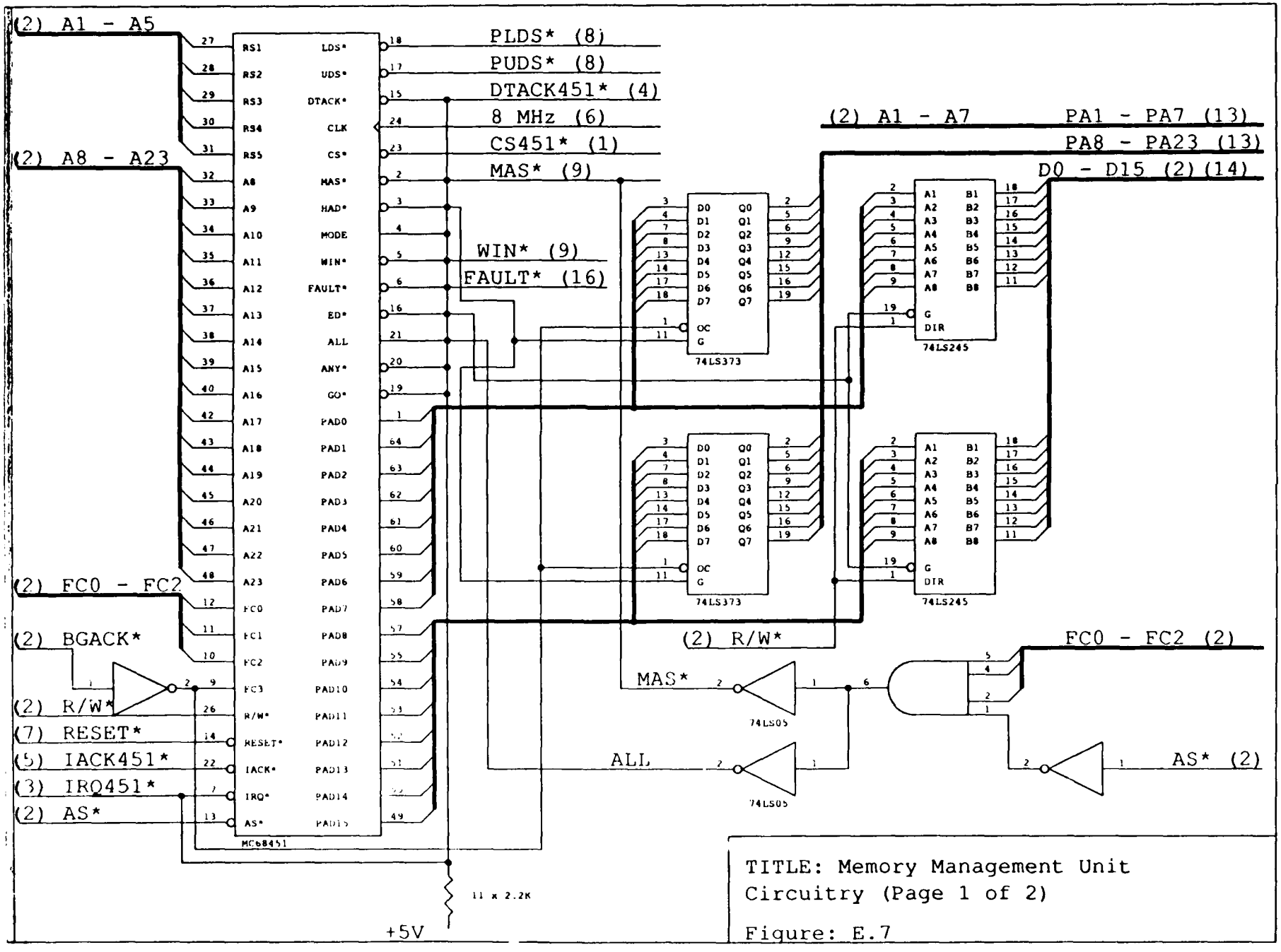


COMMENTS:

(1) - Chip Select Request Line 1 of 74F764; To Dual-port DRAM Controller Circuitry
(2) - Off Board Resource; To VMEbus Controller Circuitry and BERR* Circuitry
(3) - ROM Enable; To EPROM Circuitry and DTACK* Circuitry
(4) - Chip Select MC68681 DUART; To DUART Circuitry
(5) - Chip Select MC68451 MMU; To MMU Circuitry
(6) - Chip Select SCN68153 Interrupt Handler Hardware; To Interrupt Handler Circuitry
(7) - SRAM Enable; To SRAM Circuitry and DTACK* Circuitry
(8) - From MMU Circuitry
NC = Not Connected

TITLE: Local Bus Address Decode Circuitry

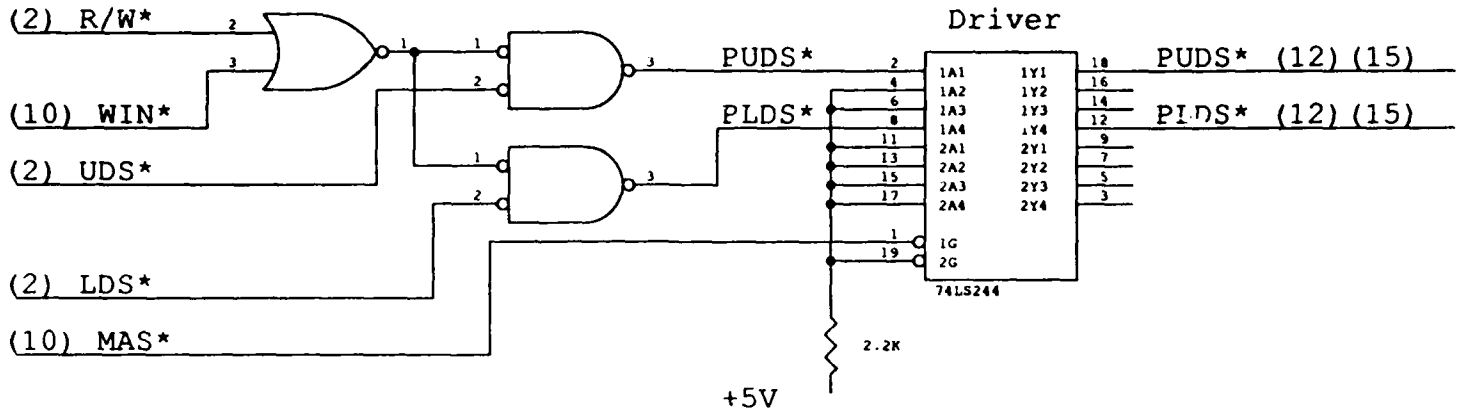
Figure: E.6



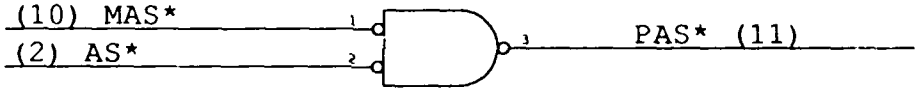
TITLE: Memory Management Unit Circuitry (Page 1 of 2)

Figure: E.7

Physical Data Strobe Generator Circuitry



Physical Address Strobe Generator Circuitry

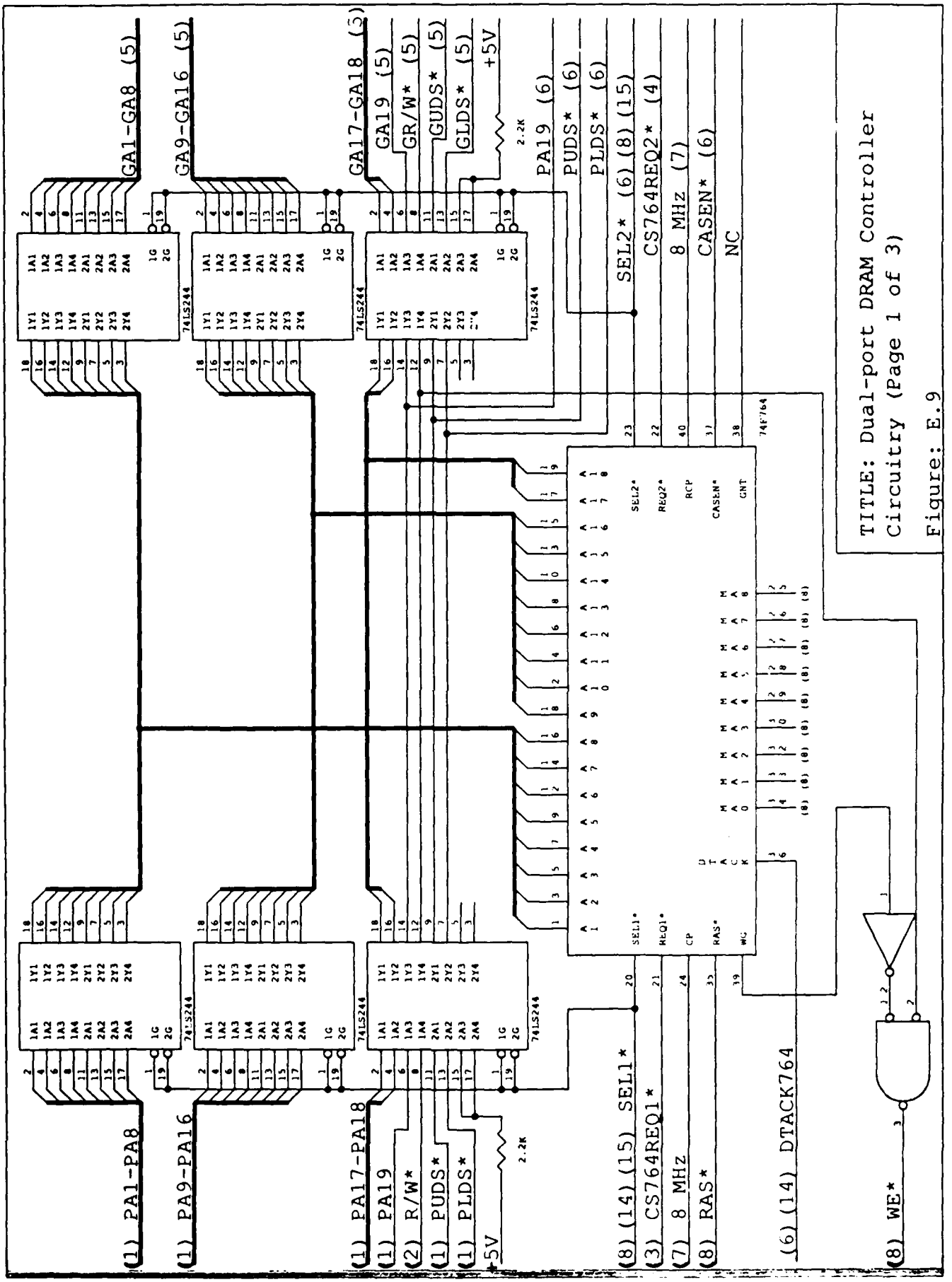


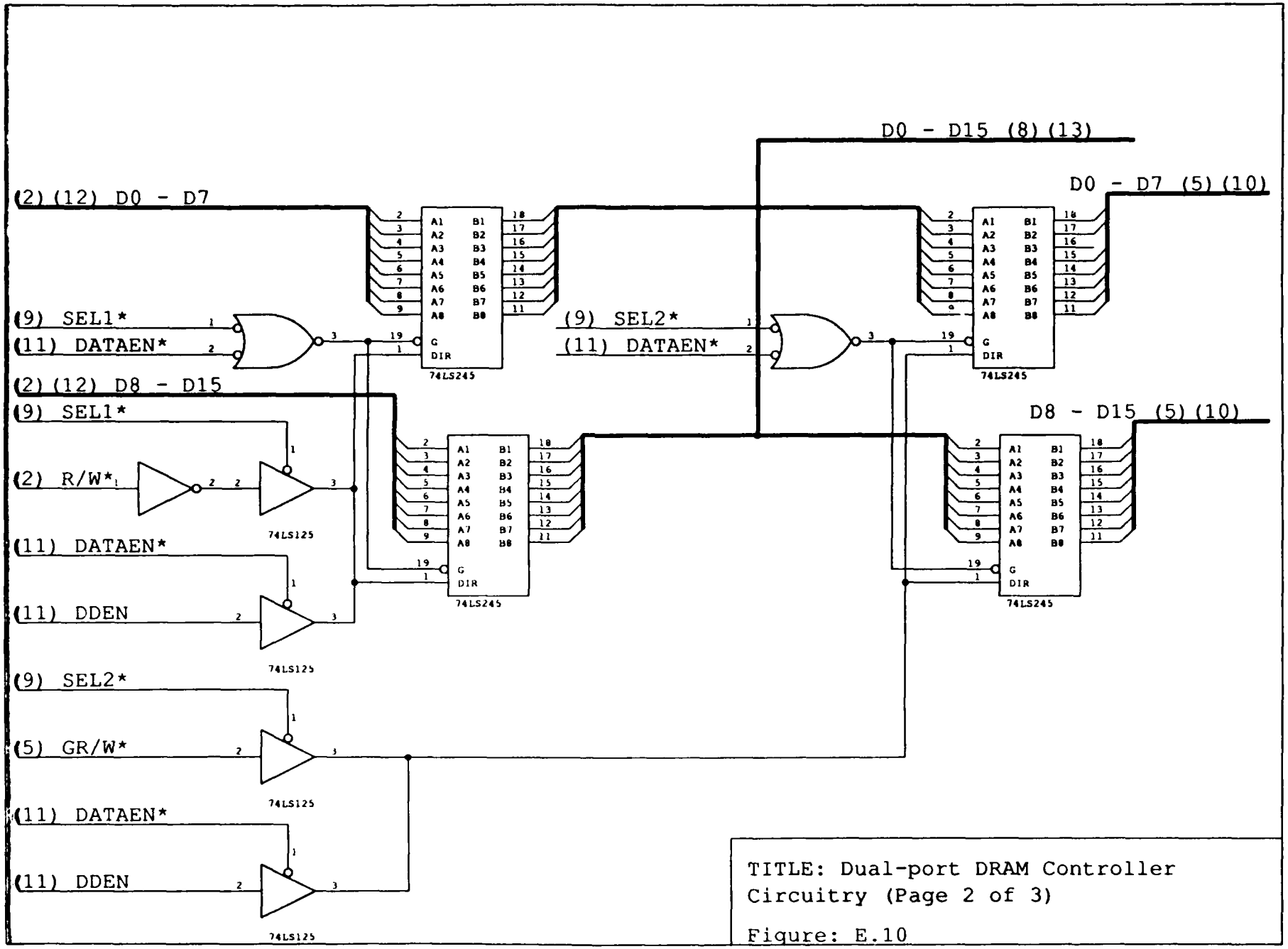
COMMENTS:

(1) - From Local Bus Address Decode Circuitry
(2) - From CPU
(3) - To Interrupt Handler Circuitry
(4) - To DTACK* Circuitry
(5) - From Interrupt Handler Circuitry
(6) - From Clock Circuitry
(7) - From RESET* Circuitry
(8) - From Physical Data Strobe Circuitry, Page 2 of 2
(9) - To Page 2 of 2
(10) - From Page 1 of 2
(11) - To Devices Requiring an Address Strobe
(12) - To Devices Requiring Data Strobes
(13) - To Devices Requiring Address Lines
(14) - To CPU
(15) - To Page 1 of 2
(16) - To BERR* Circuitry

TITLE: Memory Management Unit Circuitry (Page 2 of 2)

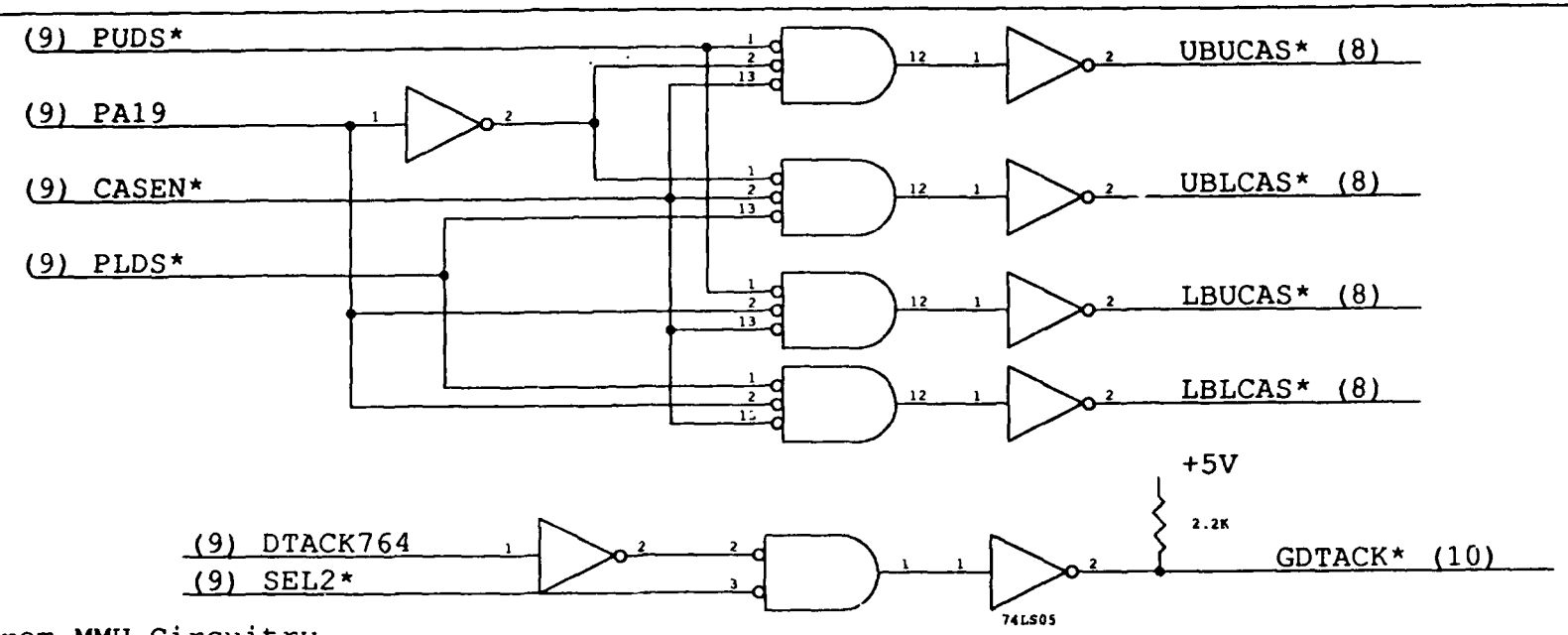
Figure: E.8


GA1-GA8 (5)
GA9-GA16 (5)
GA17-GA18 (5)
GA19 (5)
GR/W* (5)
GUDS* (5)
GLDS* (5)
+5V
PA19 (6)
PUDS* (6)
PLDS* (6)
SEL2* (6) (8) (15)
CS764REQ2* (4)
8 MHz (7)
CASEN* (6)
NC
TITLE: Dual-port DRAM Controller
Circuitry (Page 1 of 3)
Figure: E.9
(1) PA1-PA8
(1) PA9-PA16
(1) PA17-PA18
(1) PA19
(2) R/W*
(1) PUDS*
(1) PLDS*
+5V
(8) (14) (15) SEL1*
(3) CS764REQ1*
(7) 8 MHz
(8) RAS*
(6) (14) DTACK764
(8) WE*
74LS244
74F764
2.2K



TITLE: Dual-port DRAM Controller Circuitry (Page 2 of 3)

Figure: E.10



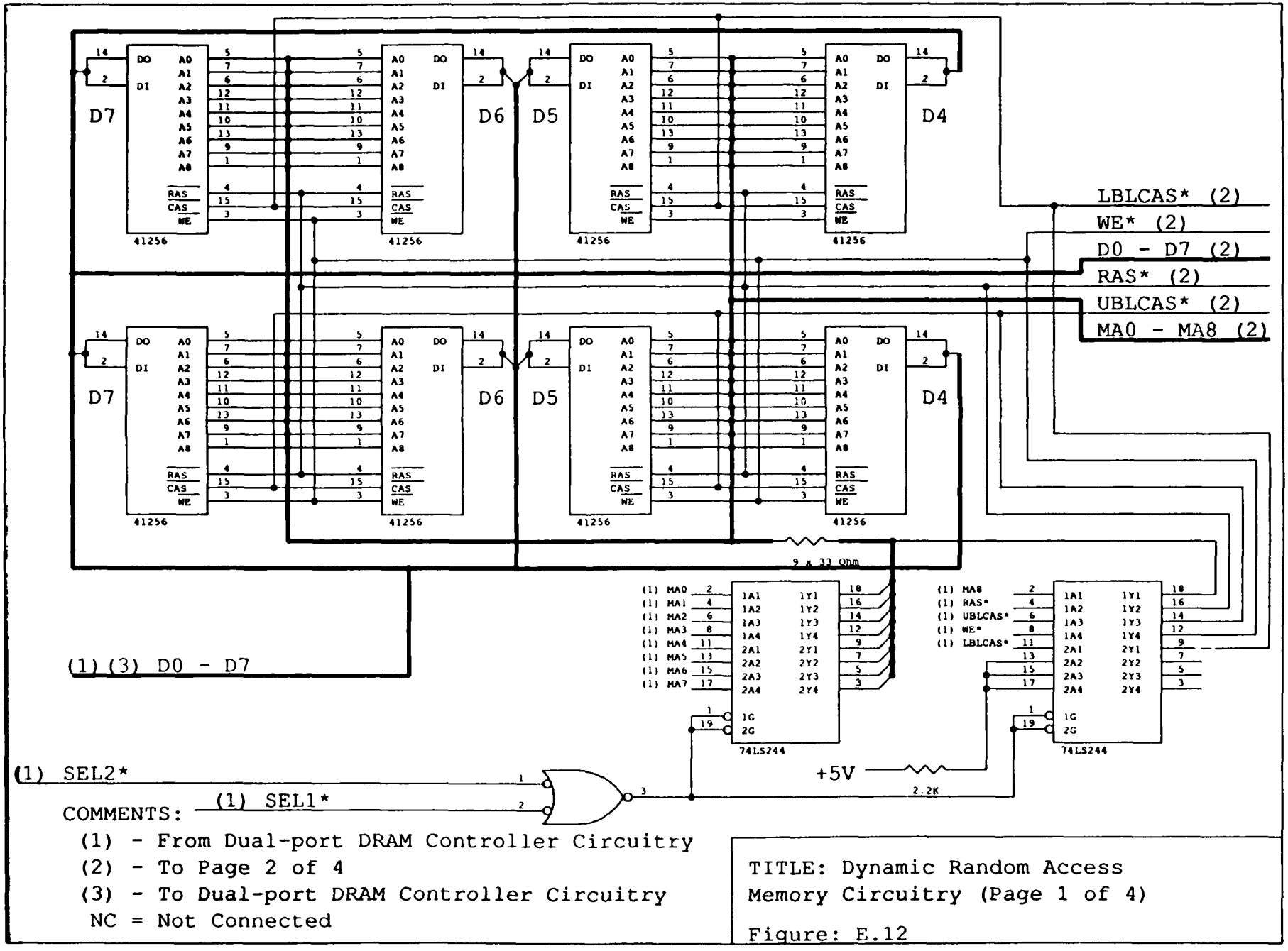
COMMENTS:

(1) - From MMU Circuitry
(2) - From CPU
(3) - From Local Bus Address Decode Circuitry
(4) - From VMEbus Address Decode Circuitry
(5) - From VMEbus
(6) - To Page 3 of 3
(7) - From Clock Circuitry
(8) - To DRAM Circuitry
(9) - From Page 1 of 3
(10) - To VMEbus
(11) - From VMEbus Controller Circuitry
(12) - To CPU
(13) - From DRAM Circuitry
(14) - To DTACK* Circuitry
(15) - To Page 2 of 3

UBLCAS* = Upper Bank Lower Byte CAS* Enable
UBUCAS* = Upper Bank Upper Byte CAS* Enable
LBLCAS* = Lower Bank Lower Byte CAS* Enable
LBUCAS* = Lower Bank Upper Byte CAS* Enable
NC = Not Connected

TITLE: Dual-port DRAM Controller Circuitry (Page 3 of 3)

Figure: E.11

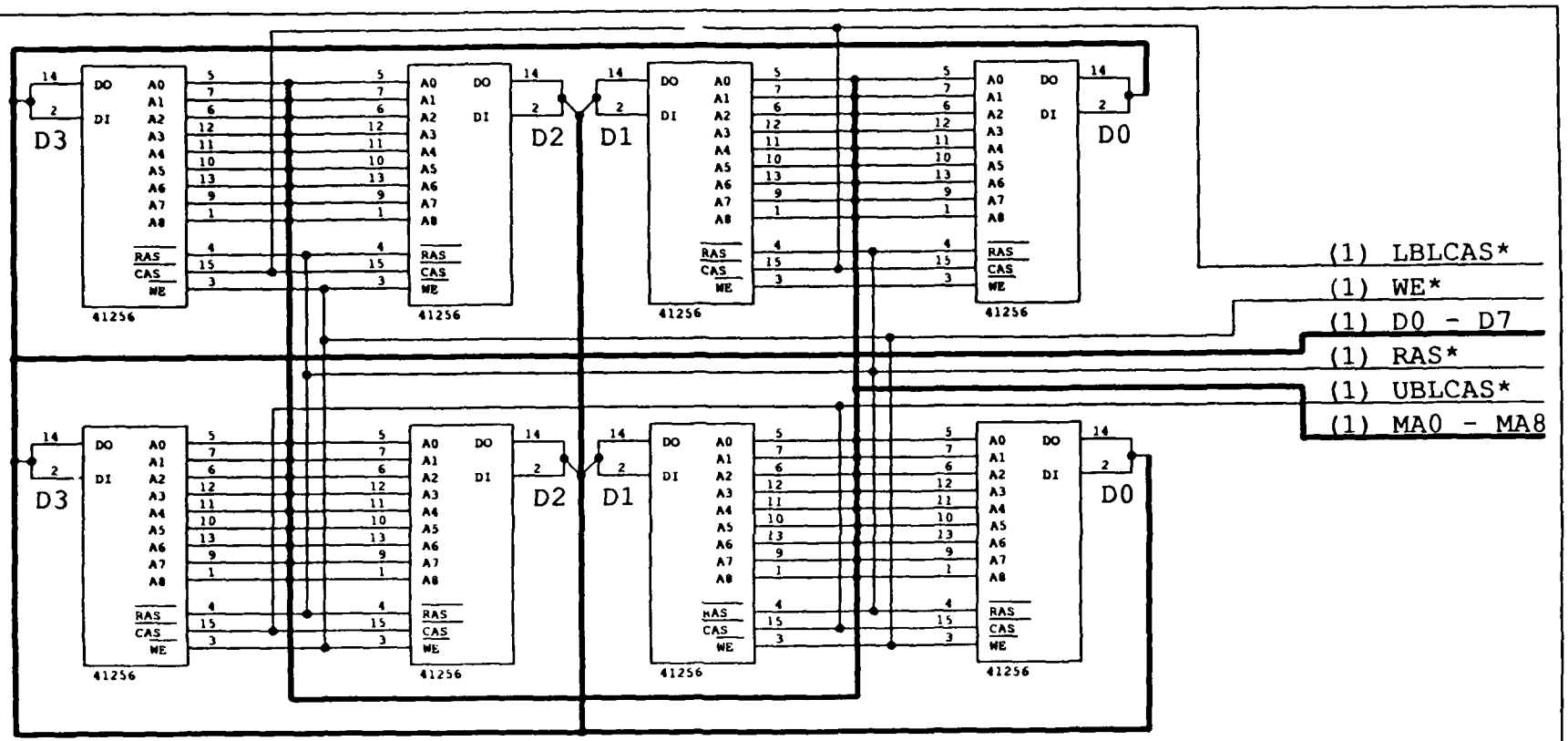


COMMENTS:

(1) - From Dual-port DRAM Controller Circuitry
(2) - To Page 2 of 4
(3) - To Dual-port DRAM Controller Circuitry
NC = Not Connected

TITLE: Dynamic Random Access Memory Circuitry (Page 1 of 4)

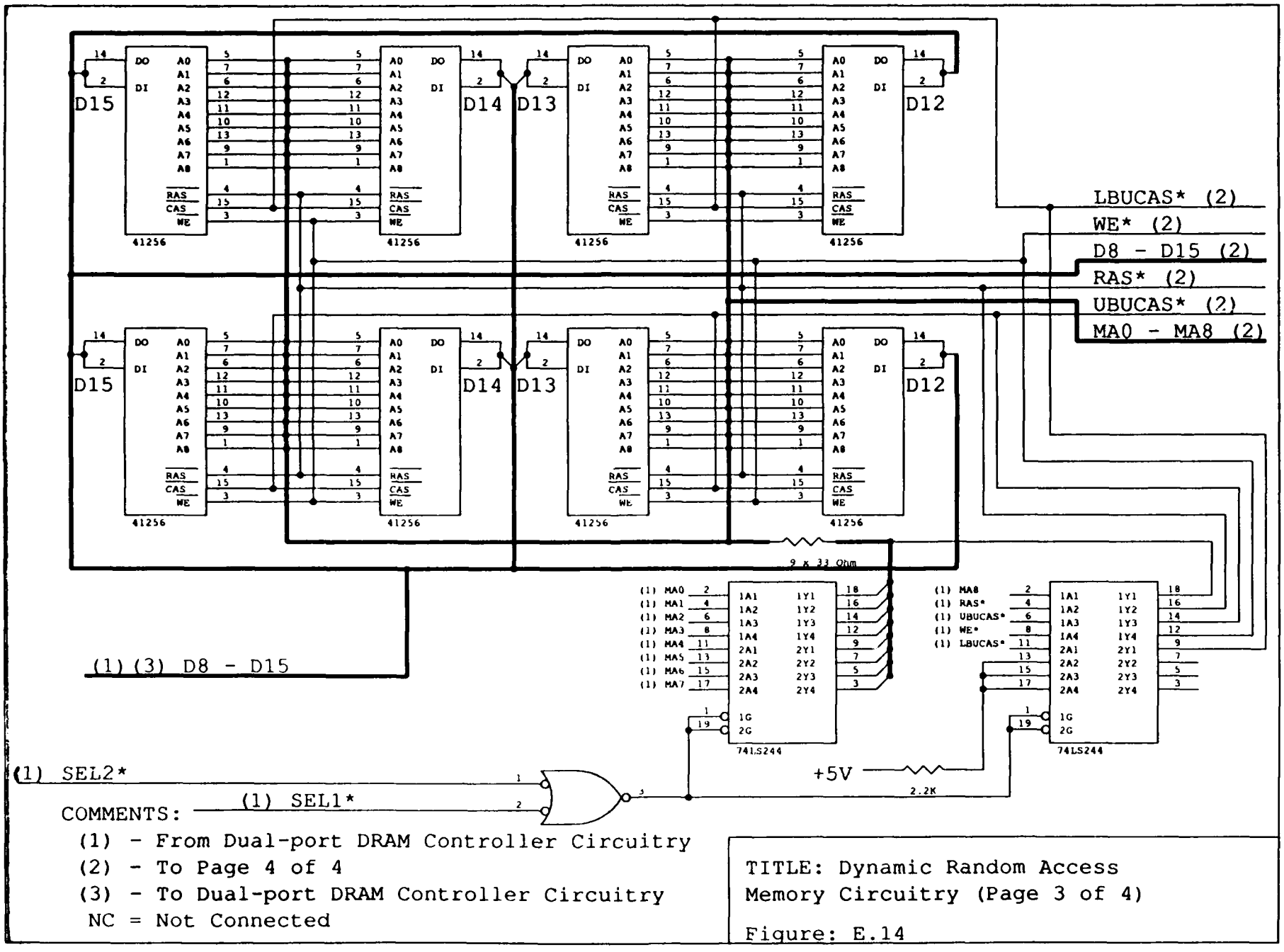
Figure: E.12



COMMENT:
(1) - From Page 1 of 4

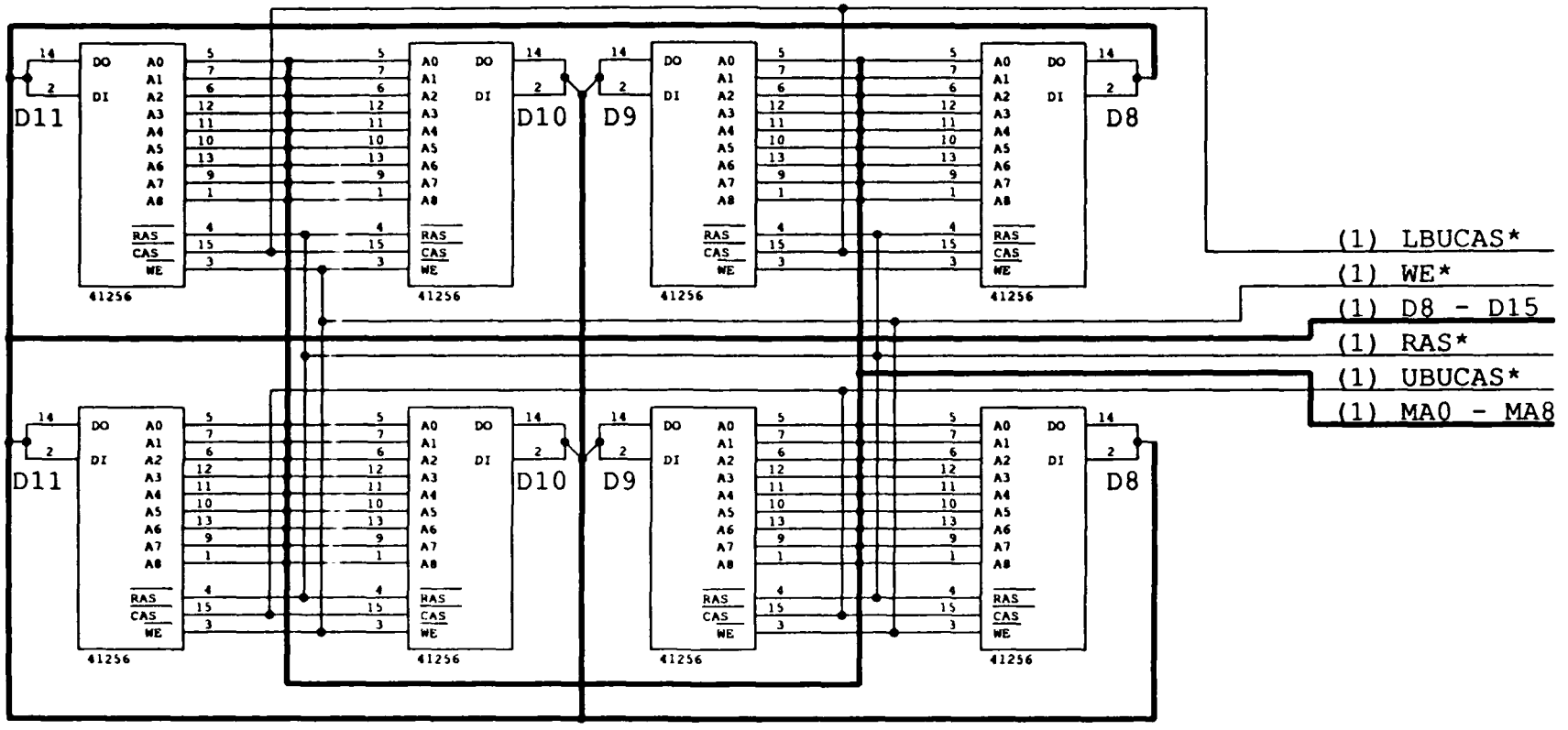
TITLE: Dynamic Random Access Memory Circuitry (Page 2 of 4)

Figure: E.13



TITLE: Dynamic Random Access Memory Circuitry (Page 3 of 4)

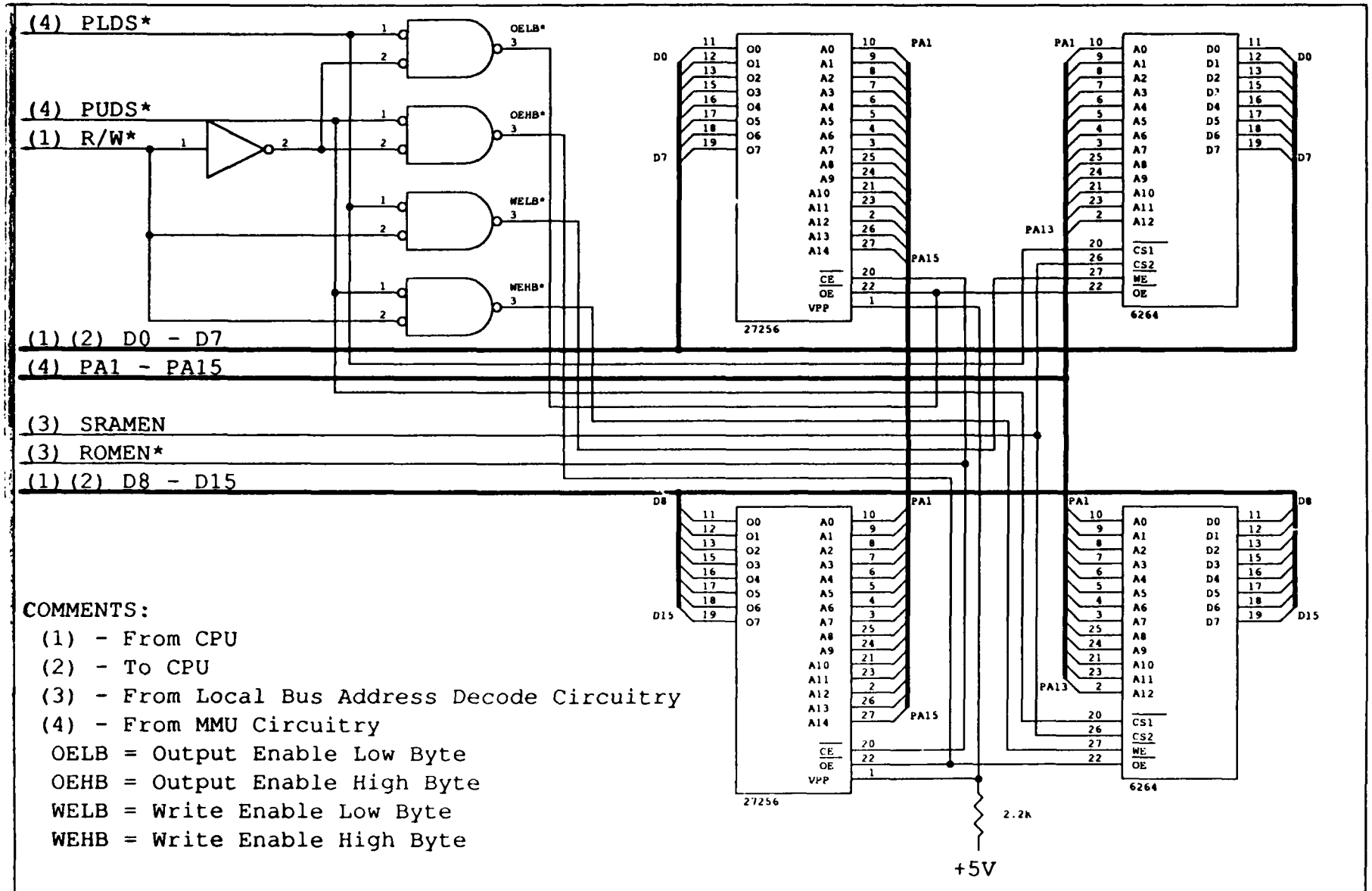
Figure: E.14



COMMENT:
(1) - From Page 3 of 4

TITLE: Dynamic Random Access Memory Circuitry (Page 4 of 4)

Figure: E.15

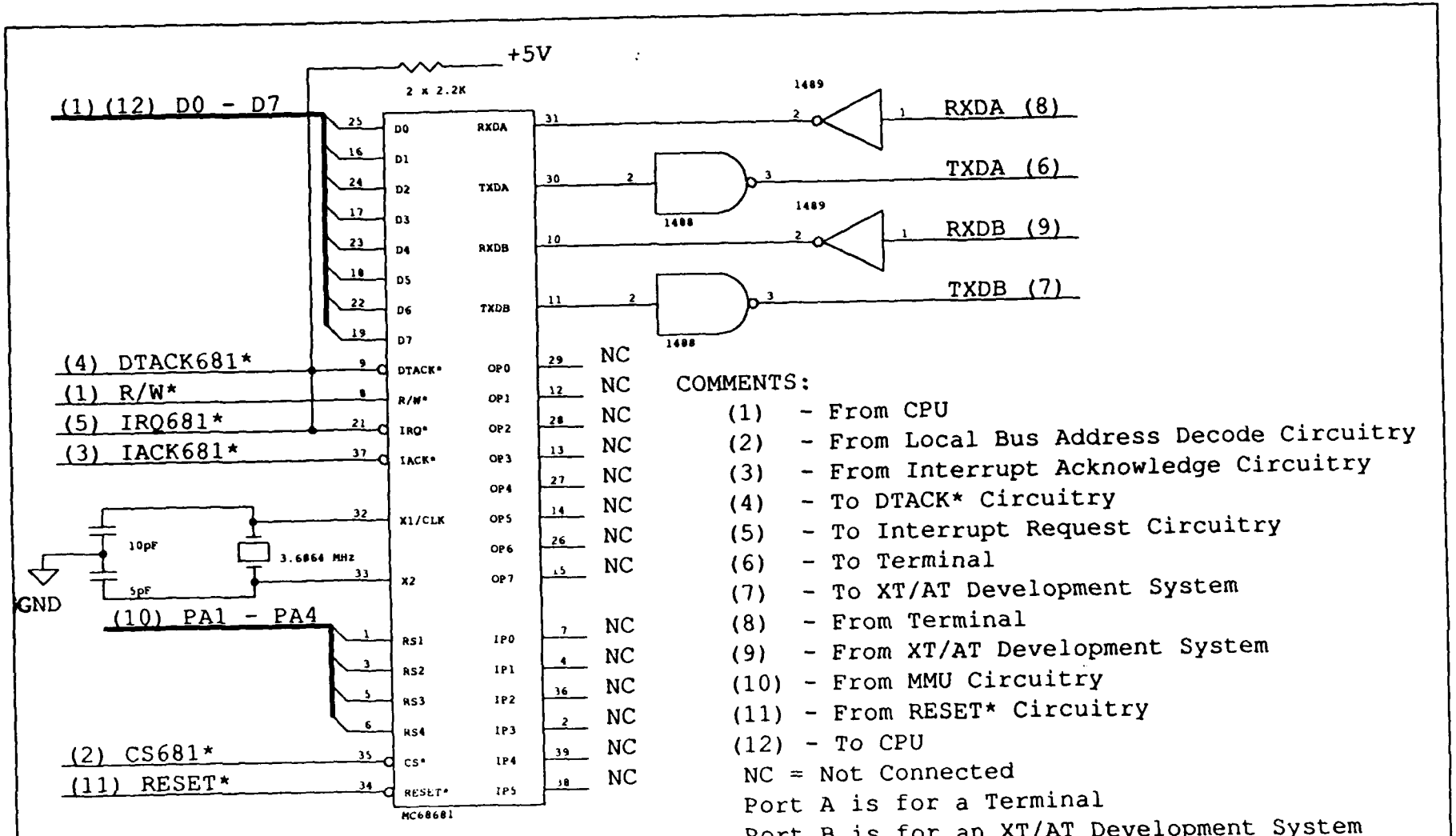


COMMENTS:

(1) - From CPU
(2) - To CPU
(3) - From Local Bus Address Decode Circuitry
(4) - From MMU Circuitry
OELB = Output Enable Low Byte
OEHB = Output Enable High Byte
WELB = Write Enable Low Byte
WEHB = Write Enable High Byte

TITLE: EPROM and SRAM Circuitry

Figure: E.16

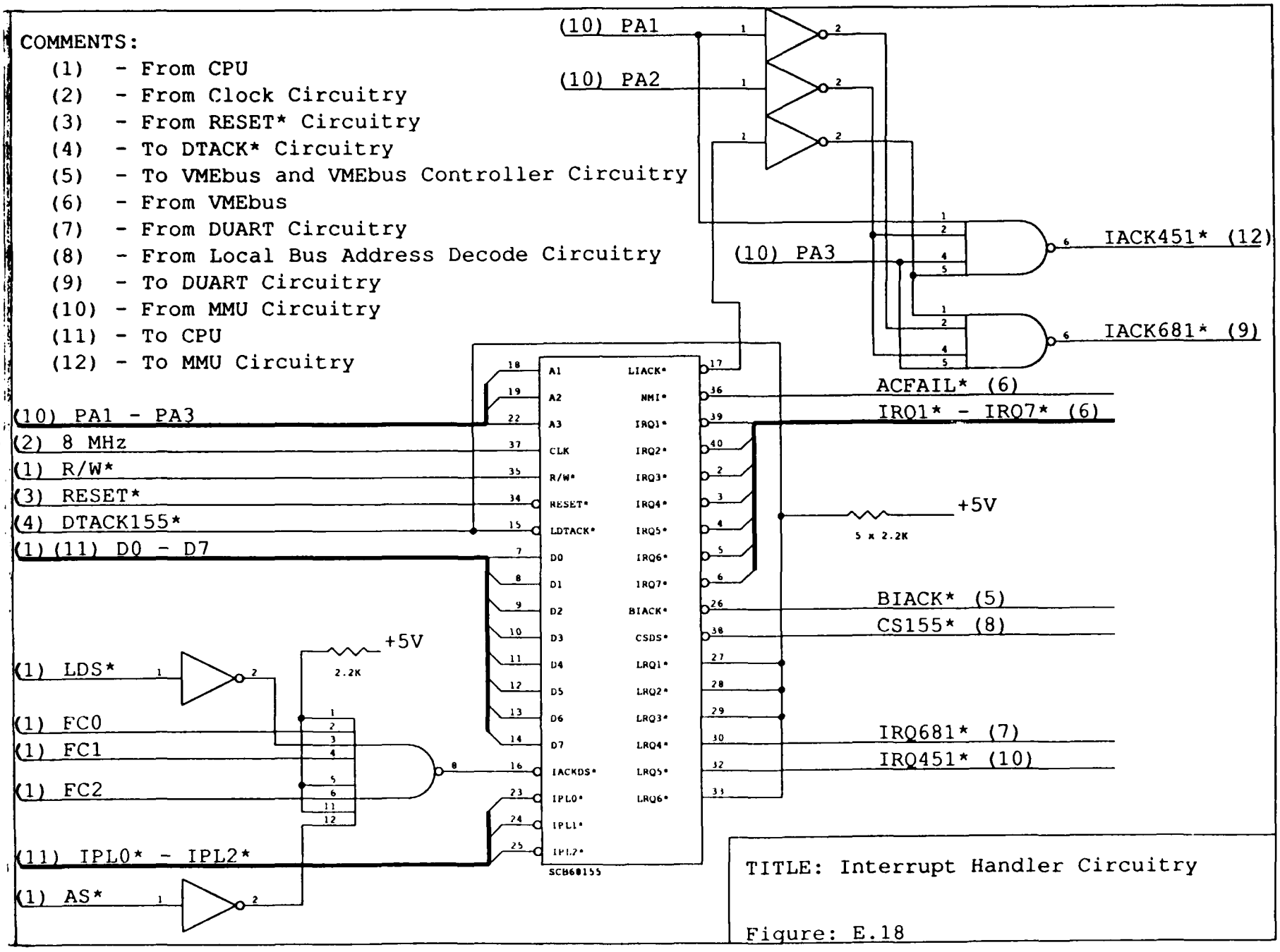


COMMENTS:

(1) - From CPU
(2) - From Local Bus Address Decode Circuitry
(3) - From Interrupt Acknowledge Circuitry
(4) - To DTACK* Circuitry
(5) - To Interrupt Request Circuitry
(6) - To Terminal
(7) - To XT/AT Development System
(8) - From Terminal
(9) - From XT/AT Development System
(10) - From MMU Circuitry
(11) - From RESET* Circuitry
(12) - To CPU
NC = Not Connected
Port A is for a Terminal
Port B is for an XT/AT Development System

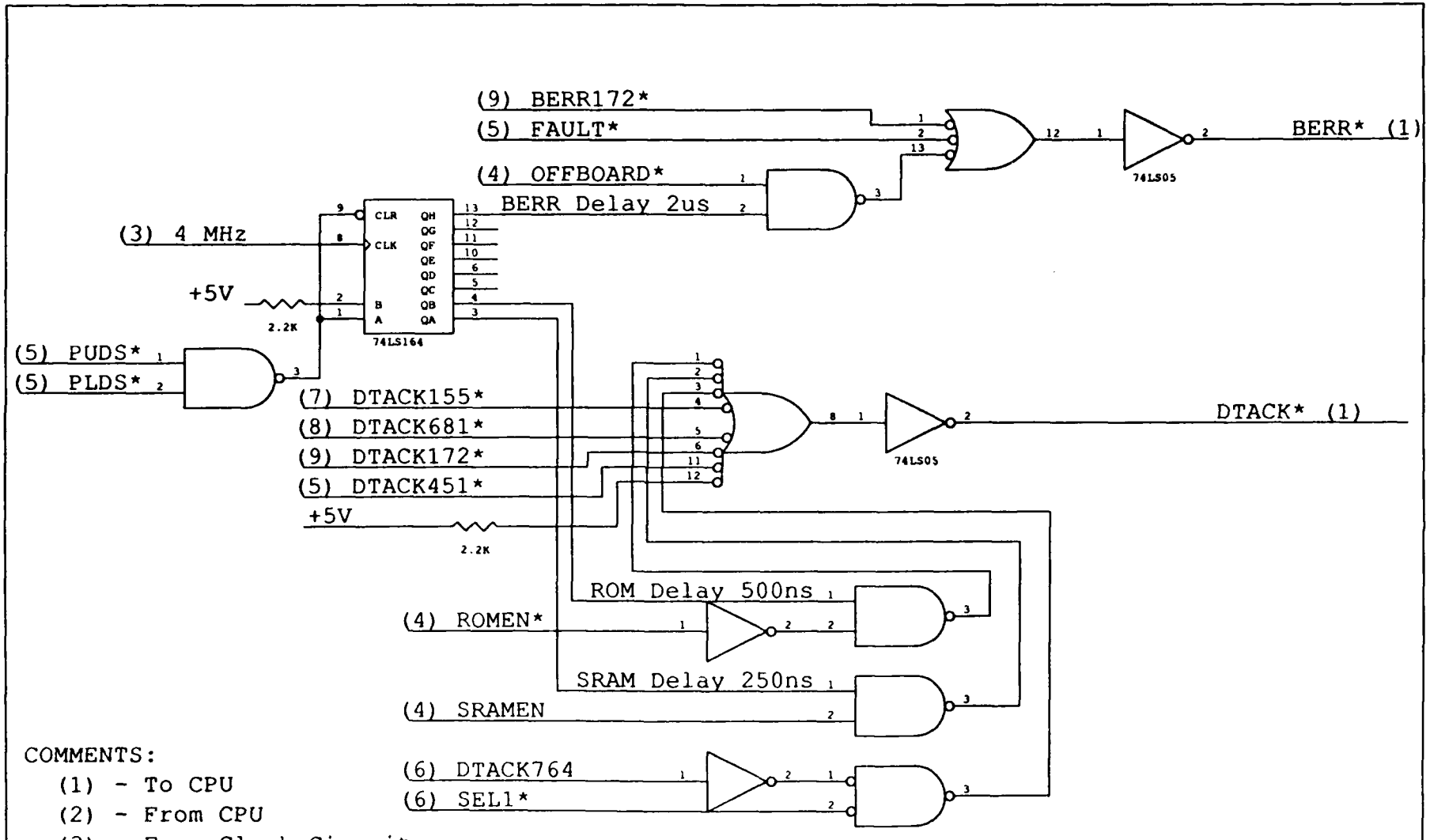
TITLE: Dual-port Asynchronous Receiver/ Transmitter Serial Port Circuitry

Figure: E.17



TITLE: Interrupt Handler Circuitry

Figure: E.18

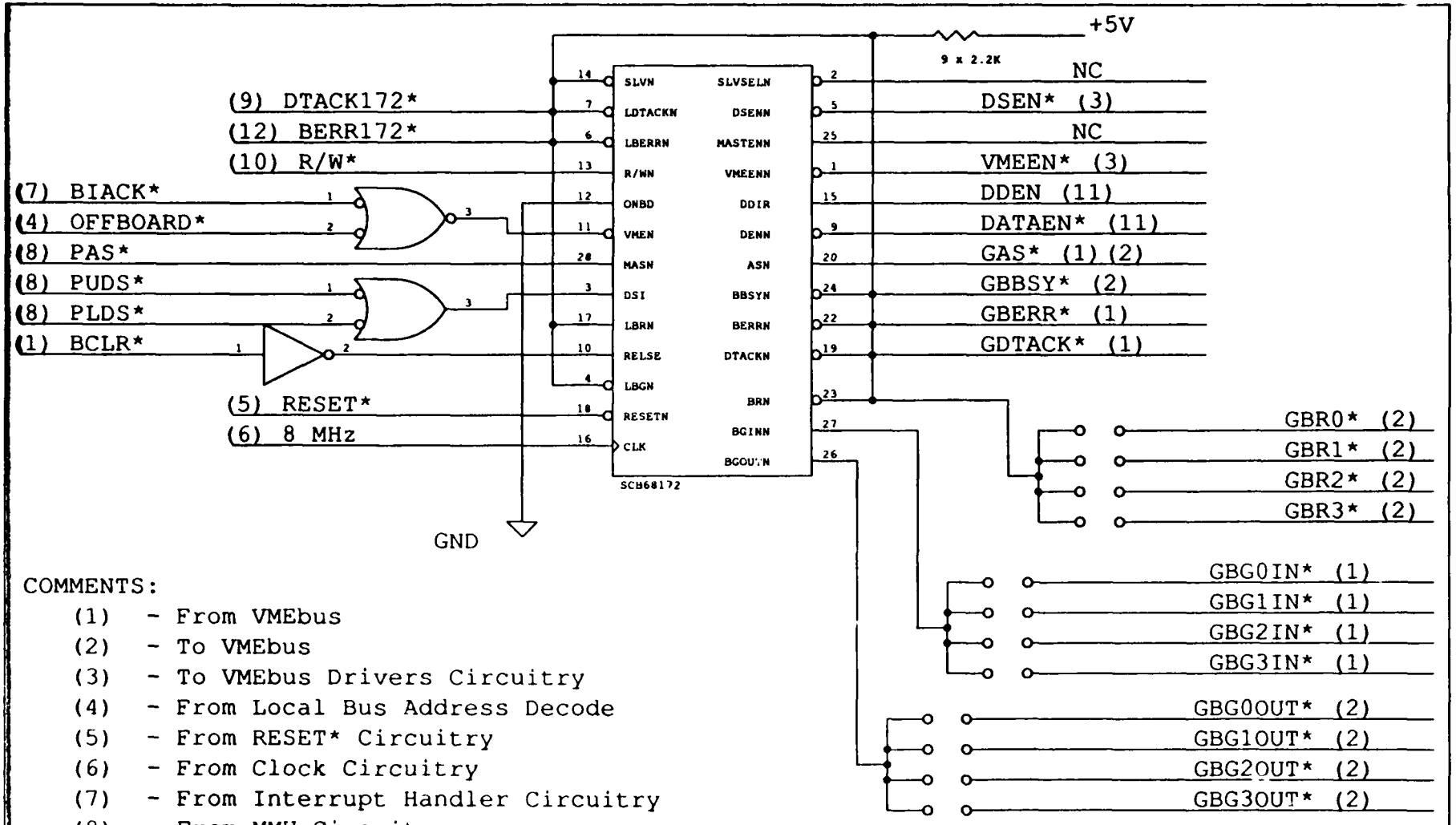


COMMENTS:

(1) - To CPU
(2) - From CPU
(3) - From Clock Circuitry
(4) - From Local Bus Address Decode Circuitry
(5) - From MMU Circuitry
(6) - From Dual-port DRAM Controller Circuitry
(7) - From Interrupt Handler Circuitry
(8) - From DUART Circuitry
(9) - From VMEbus Controller Circuitry

TITLE: DTACK* and BERR* Generation Circuitry

Figure E.19

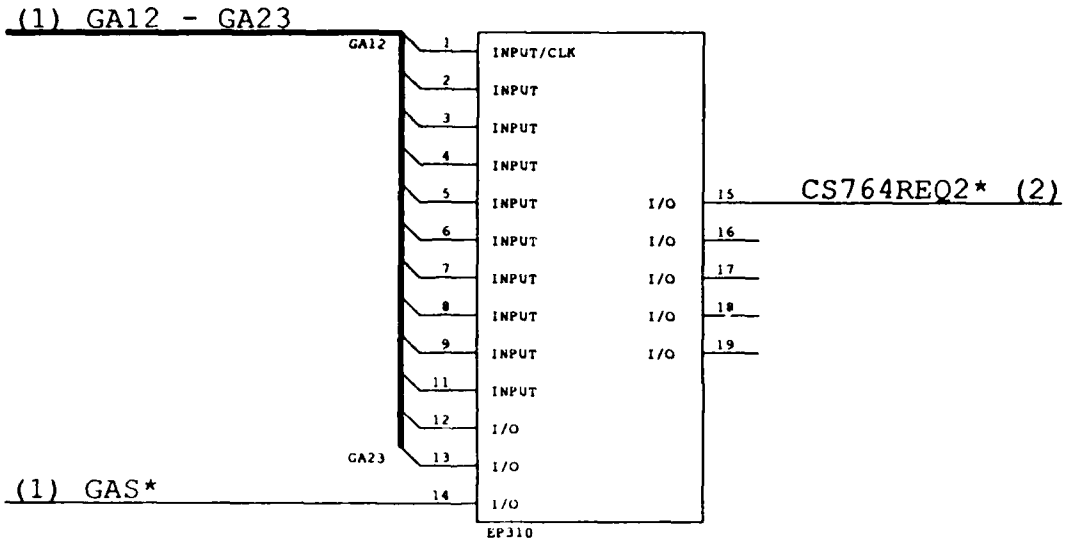


COMMENTS:

(1) - From VMEbus
(2) - To VMEbus
(3) - To VMEbus Drivers Circuitry
(4) - From Local Bus Address Decode
(5) - From RESET* Circuitry
(6) - From Clock Circuitry
(7) - From Interrupt Handler Circuitry
(8) - From MMU Circuitry
(9) - To DTACK* Circuitry
(10) - From CPU
(11) - To Dual-port DRAM Controller Circuitry
(12) - To BERR* Circuitry
NC = Not Connected

TITLE: VMEbus Controller Circuitry

Figure: E.20



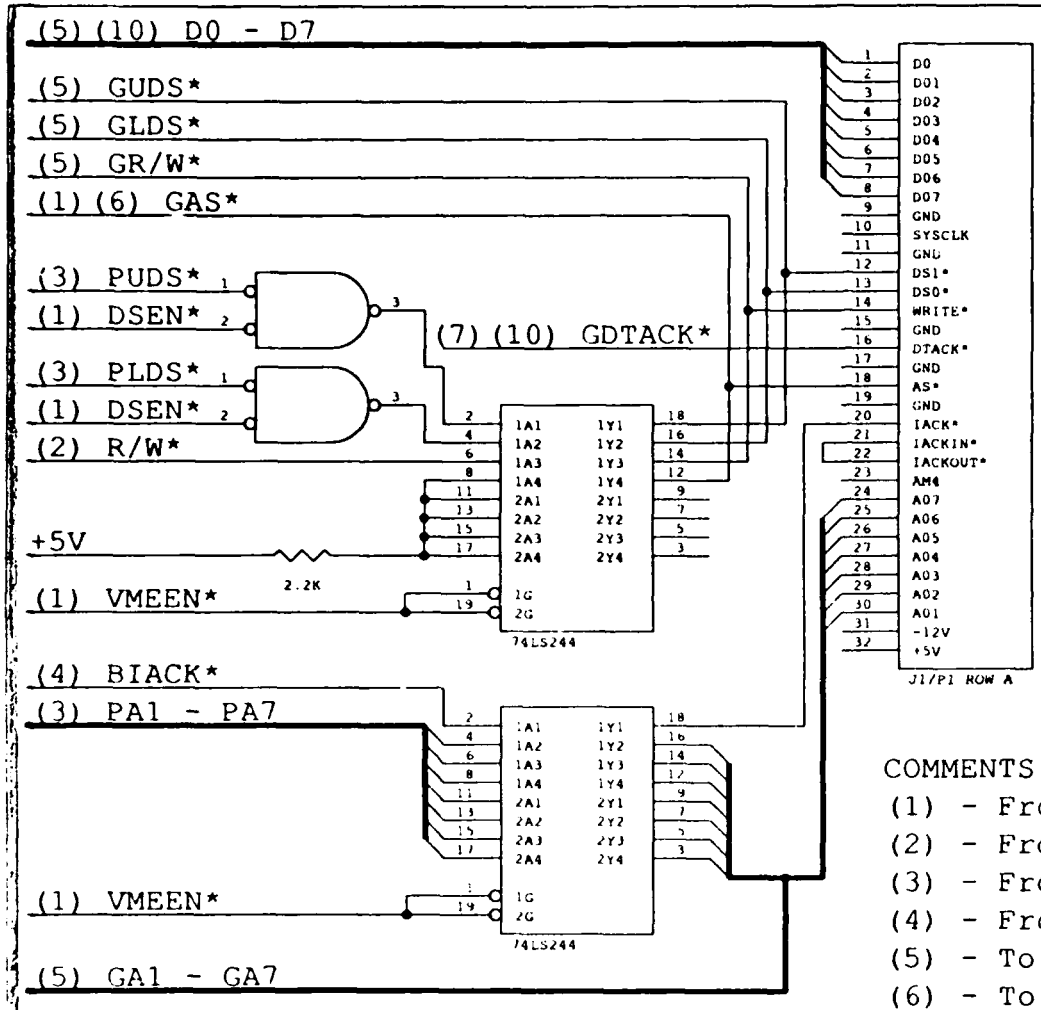
COMMENTS:

(1) - From VMEbus

(2) - Chip Select Request Line 2 of 74F764; To Dual-port DRAM Controller Circuitry

TITLE: VMEbus Address Decode Circuitry

Figure: E.21

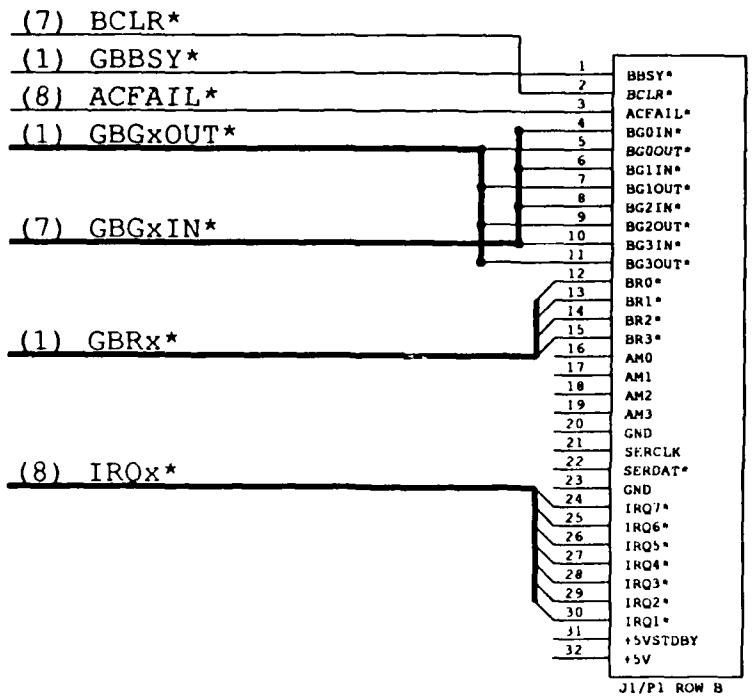


COMMENTS:

(1) - From VMEbus Controller Circuitry
(2) - From CPU
(3) - From MMU Circuitry
(4) - From Interrupt Handler Circuitry
(5) - To Dual-port DRAM Controller Circuitry
(6) - To VMEbus Address Decode Circuitry
(7) - To VMEbus Controller Circuitry
(8) - To Interrupt Handler Circuitry
(9) - To HALT* and RESET* Circuitry
(10)- From Dual-port DRAM Controller Circuitry

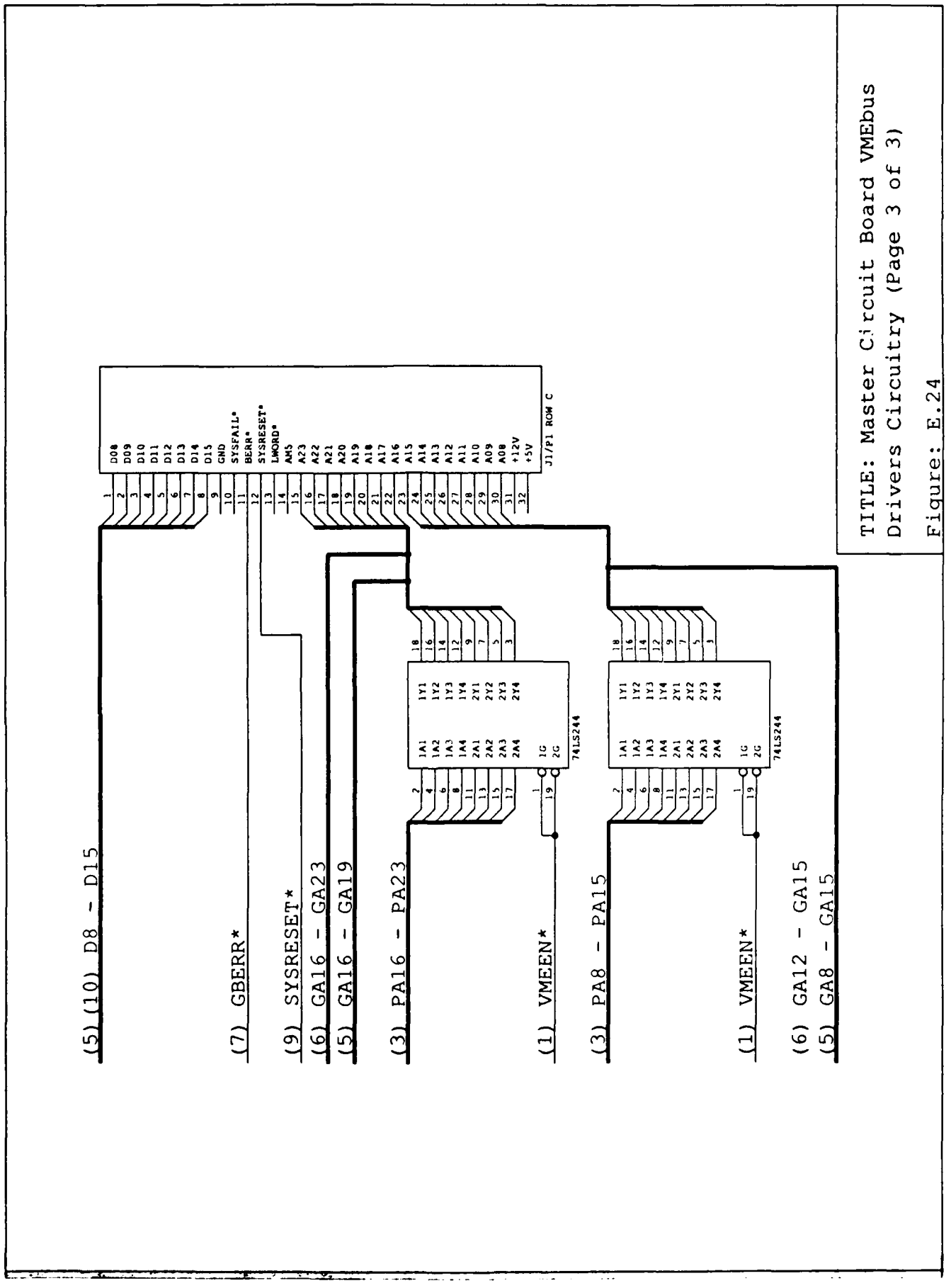
TITLE: Master Circuit Board VMEbus Drivers Circuitry (Page 1 of 3)

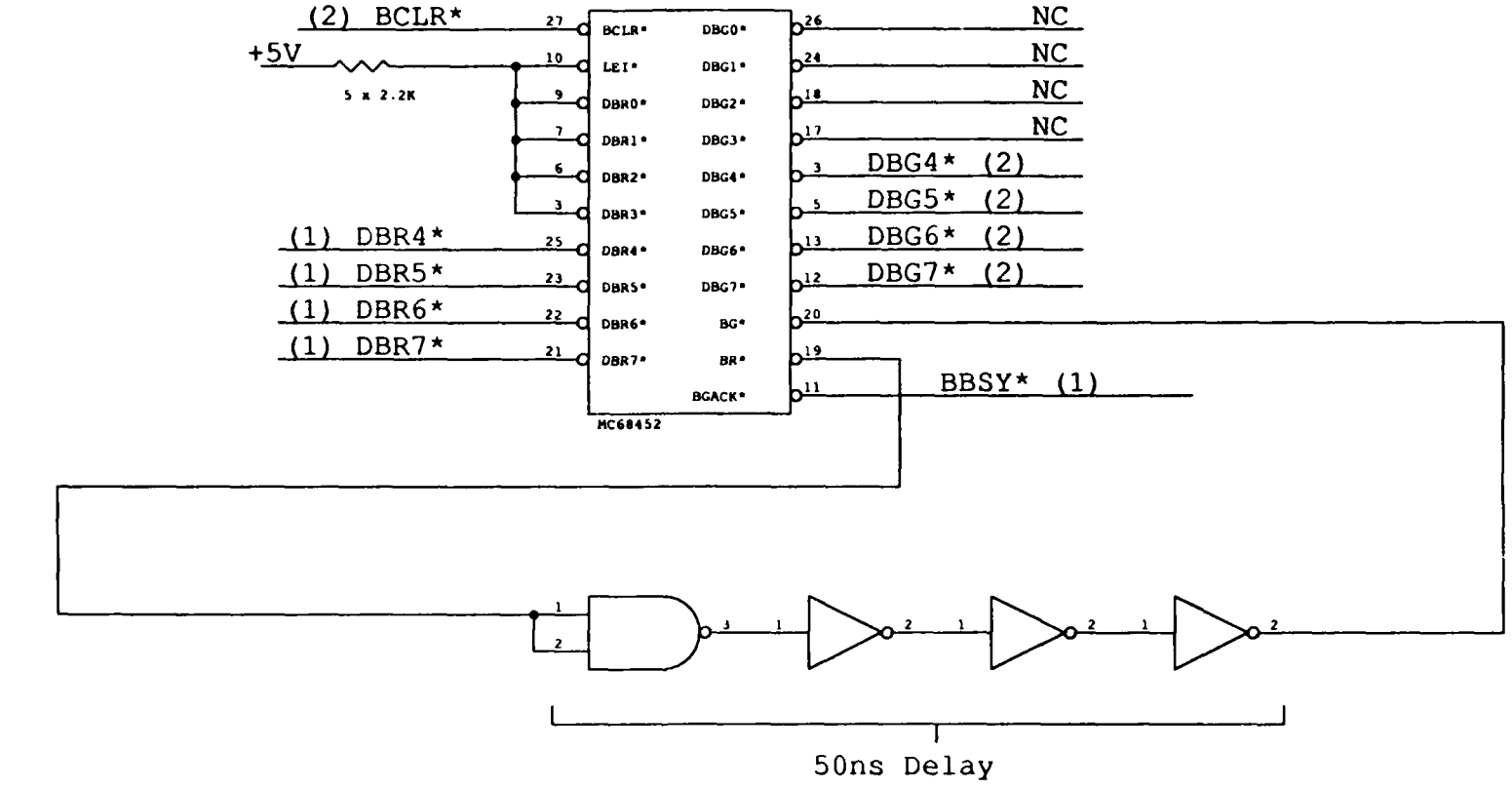
Figure: E.22



TITLE: Master Circuit Board VMEbus Drivers Circuitry (Page 2 of 3)

Figure: E.23


TITLE: Master Circuit Board VMEbus
Drivers Circuitry (Page 3 of 3)
Figure: E.24
J1/P1 ROW C
D08
D09
D10
D11
D12
D13
D14
D15
GND
SYSFAIL*
BERR*
SYSRESET*
LWORD*
AM5
A23
A22
A21
A20
A19
A18
A17
A16
A15
A14
A13
A12
A11
A10
A09
A08
+12V
+5V
74LS244
(5) (10) D8 - D15
(7) GBERR*
(9) SYSRESET*
(6) GA16 - GA23
(5) GA16 - GA19
(3) PA16 - PA23
(1) VMEEN*
(3) PA8 - PA15
(1) VMEEN*
(6) GA12 - GA15
(5) GA8 - GA15



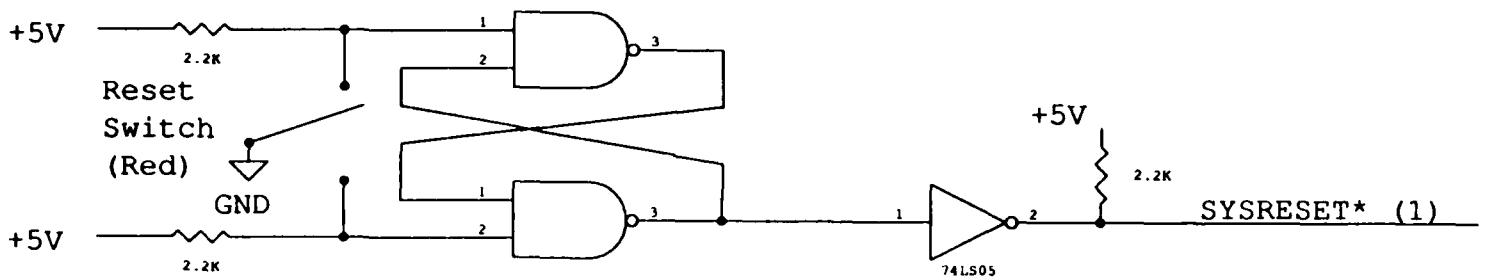
COMMENTS:

(1) - From VMEbus
(2) - To VMEbus
NC = Not Connected

TITLE: VMEbus Arbitration Circuitry

Figure: E.25

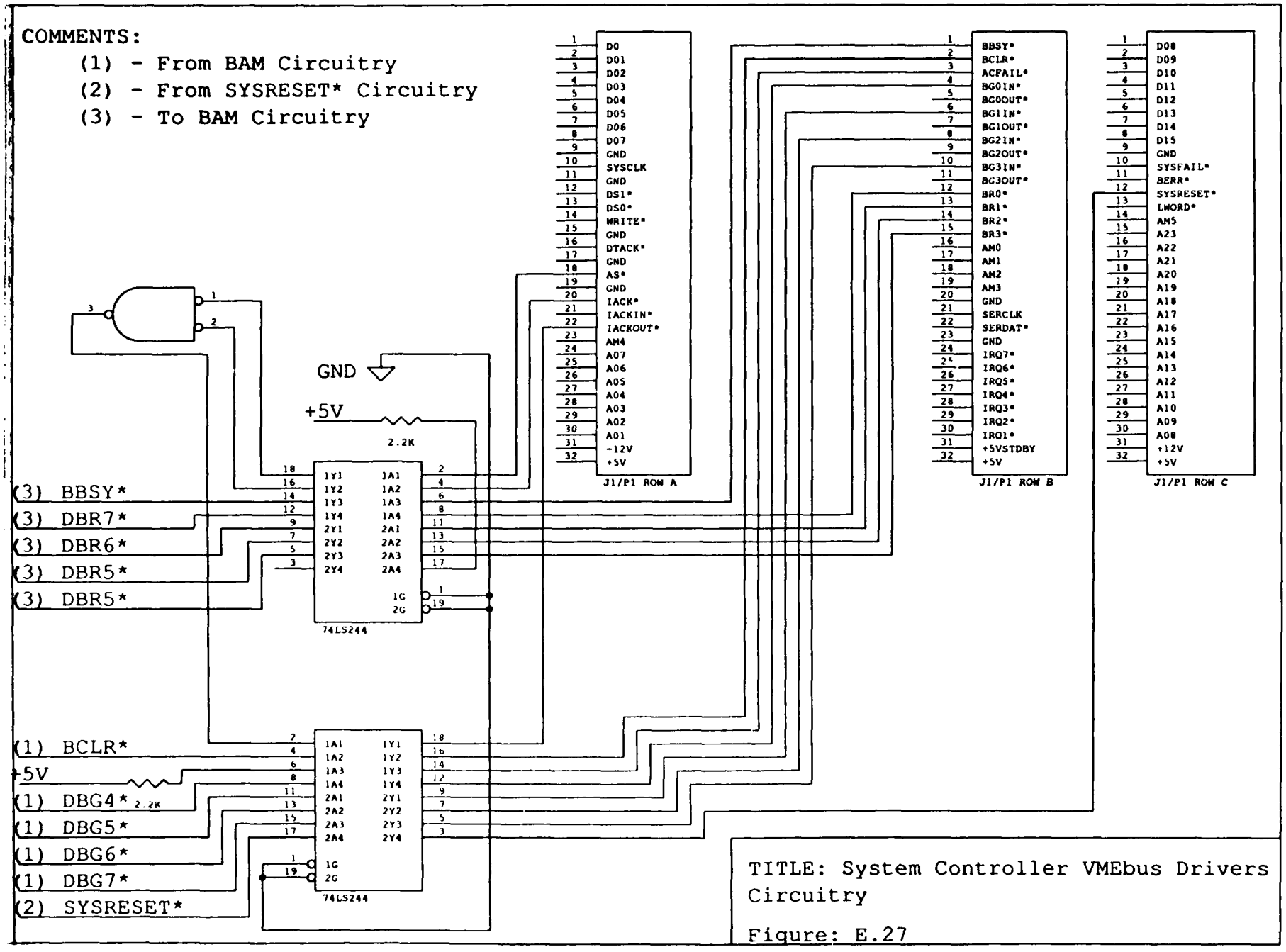
Debouncing Circuitry



COMMENT:
(1) - To VMEbus

TITLE: SYSRESET* Generation Circuitry

Figure: E.26


COMMENTS:
(1) - From BAM Circuitry
(2) - From SYSRESET* Circuitry
(3) - To BAM Circuitry
GND
+5V
2.2K
(3) BBSY*
(3) DBR7*
(3) DBR6*
(3) DBR5*
(3) DBR5*
(1) BCLR*
+5V
2.2K
(1) DBG4*
(1) DBG5*
(1) DBG6*
(1) DBG7*
(2) SYSRESET*
74LS244
74LS244
J1/P1 ROW A
J1/P1 ROW B
J1/P1 ROW C
TITLE: System Controller VMEbus Drivers Circuitry
Figure: E.27

LIST OF REFERENCES

1. Stone, H.S., High-Performance Computer Architecture, Addison-Wesley Publishing Company, Reading, Ma, 1987.

2. Clements, A., Microprocessor System Design: 68000 Hardware, Software, and Interfacing, PWS Publishers, Boston, Ma, 1987.

3. Borrill, P.L., "Microstandards Special Feature: A Comparison of 32-Bit Buses", IEEE MICRO, Vol. 5, No. 6, pp. 71-79, December 1985.

4. The VMEbus Specification, Printex Publishing, Inc., Tempe, Az, 1985.

5. Pri-Tal, S. and MacKenna, C., "Understanding VMEbus Architecture", Electronic Products, Vol. 27, No. 19, pp. 103-110, 15 March, 1985.

6. Stone, H.S., Microcomputer Interfacing, Addison-Wesley Publishing Company, Reading, Ma, 1983.

7. M68000 8-/16-/32-Bit Microprocessors Programmer's Reference Manual, Prentice-Hall, Englecliffs, N.J., 1986.

8. MC68452 Bus Arbitration Module Advance Information, Motorola Semiconductors, Phoenix, Az, 1985.

9. 68000/08/10 Cross Assembler, 2500AD Software, Inc., Aurora, Co, 1987.

10. MC68000 Educational Computer Board User's Manual, Motorola, Inc., Tempe, Az, 1982.

11. ABEL 2.0, Data I/O Corp., Santa Clara, Ca, 1986.

12. Brooks, S.L., "The Design of an Intelligent Multidisk Control Module for VME bus Based Systems", Master's Thesis, Naval Postgraduate School, Monterey, Ca, December 1987.

13. Memory Management Unit Advance Information, Motorola Semiconductors, Phoenix, Az, 1983.

14. Microprocessor Data Manual, Signetics Corp., Sunnyvale, Ca, 1986.

15. MC68681 Dual Asynchronous Receiver/Transmitter (DUART) Advance Information, Motorola Semiconductors, Phoenix, Az, 1985.

16. Altera Data Book, Altera Corporation, Santa Clara, Ca, 1987.

17. DRAM Dual-Ported Controllers Product Specification, Signetics Corp., Sunnyvale, Ca, 1987.

18. MC68010/MC68012 16-/32-Bit Virtual Memory Microprocessors Advance Information, Motorola Semiconductors, Phoenix, Az, 1985.

19. MacGregor D. and Mothersole D.S., "Virtual Memory and the MC68010", IEEE Micro, Vol. 3, No. 3, pp. 24-39, June 1983.

BIBLIOGRAPHY

1. 74LS764 DRAM Controller Product Specification, Signetics Corp., Sunnyvale, Ca, 1986.

2. Baliga, S., "Simplifying VMEbus System Design", VMEbus System Magazine, pp. 23-25, Fall/Winter 1986.

3. Baliga, S., "Three-Chip Control Set Trims VMEbus Logic For Asynchronous Systems", Electronic Design, Vol. 33, No. 2, pp. 207-214, 24 January 85.

4. Brown, G. and Harper, K., MC68008 Minimum Configuration System, Motorola, Inc., 1984.

5. Harper, K., A Terminal Interface, Printer Interface, And Background Printing For An MC68000-Based System Using The MC68681 DUART, Motorola, Inc., 1984.

6. MacKenna, C., "Bus Controller Chip Lets Processor Board Switch Master And Slave Roles", Electronic Design, Vol. 32, No. 13, pp. 243-254, 28 June 1984.

7. MC68000 Educational Computer Board User's Manual, Motorola, Inc., Tempe, Az, 1982.

8. MTT8 Course Notes, Motorola Semiconductors, Phoenix, Az, 1986.

9. Reddy, A., Dynamic Memory Refresh Considerations, Motorola, Inc., 1983.

10. Scales, H., An Evaluation Tool For The MC68451 MMU, Motorola Inc., 1982.

11. Scales, H., Virtual Memory Using The MC68000 And The MC68451 MMU, Motorola, Inc. 1982.

12. West, T., Dual-Ported RAM For The MC68000 Microprocessor, Motorola, Inc., 1982.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Library Code 0142 Naval Postgraduate School Monterey, California 93943-5002	2
2.	Chairman, Department of Electrical and Computer Engineering (Code EC) Naval Postgraduate School Monterey, California 93943-5000	1
3.	Dr. Larry Abbott 16047 Arborlea Dr. Friendswood, Texas 77546	1
4.	Professor Mitchell Cotton, Code EC/CO Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, California 93943-5000	1
5.	Professor Frederick Terman, Code EC/TZ Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, California 93943-5000	1
6.	Commanding Officer Attn: Lieutenant David M. Sendek, USN Naval Oceans Systems Center (NOSC), Code 845 271 Catalina Blvd. San Diego, California 92152-5000	1
7.	Roberto Ventura Crispino, LT. CN Apartado Aero 2845 Cartagena, Colombia	1
8.	Defense Technical Information Center Cameron Station Alexandria, Virginia 22304-6145	2