

University of Bath



PHD

Educational interface board for multi-family microprocessor teaching

Bakbak, Sami Ibrahim

Award date:
1988

Awarding institution:
University of Bath

[Link to publication](#)

General rights

Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal ?

Take down policy

If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

Download date: 13. May. 2019

**Educational Interface Board For
multi-Family Microprocessor Teaching**

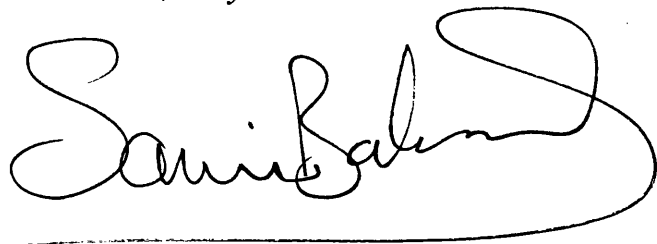
Submitted by Sami Ibrahim Bakbak
for the degree of Ph.D
of the University of Bath
1988

© COPYRIGHT

*Attention is drawn to the fact that copyright of this thesis rest with its author.
This copy of the thesis has been supplied on condition that anyone who consults it
is understood to recognise that its copyright rests with its author and that no
quotation from the thesis and no information derived from it may be published
without the prior written consent of the author.*

*This thesis may be made available for consultation within the University Library
and may be photocopied or lent to other libraries for the purposes of consultation.*

Bath, May 1988



A handwritten signature in cursive script, reading "Sami Ibrahim Bakbak". The signature is written in black ink and is positioned below the date. A horizontal line is drawn underneath the signature.

UMI Number: U005688

All rights reserved

INFORMATION TO ALL USERS

The quality of this reproduction is dependent upon the quality of the copy submitted.

In the unlikely event that the author did not send a complete manuscript and there are missing pages, these will be noted. Also, if material had to be removed, a note will indicate the deletion.



UMI U005688

Published by ProQuest LLC 2013. Copyright in the Dissertation held by the Author.
Microform Edition © ProQuest LLC.

All rights reserved. This work is protected against
unauthorized copying under Title 17, United States Code.



ProQuest LLC
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106-1346

UNIVERSITY OF BATH LIBRARY		
33	14 SEP 1988	
ELE		

S021540

SUMMARY

The rapid growth in the microprocessor population and the increasing use of microprocessors in education has resulted in many different approaches to the problem of microprocessor teaching and development.

This thesis examines the various common use techniques for microprocessor education and discusses, compares the advantages and disadvantages of each approach. A design and implementation of an educational environment, for users to investigate and learn about various currently available microprocessor families, is shown.

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to the following people who provided me with assistance and encouragement during the course of this project.

Mr. A.R.Daniels, for his supervision and constant guidance.

Dr. P.F.Whitworth, for his interest and help.

To all member of staff and colleagues at the school of Electrical Engineering, University of Bath.

Finally, I would like to specially thank my family who gave me encouragement and support throughout these years.

CONTENTS

SUMMARY

ACKNOWLEDGEMENTS

1. INTRODUCTION	1
1.1 Microprocessor background	1
1.2 Microprocessor education requirements	3
2. A REVIEW OF MICROCOMPUTER EDUCATIONAL SYSTEMS	7
2.1 Evaluation Kits	7
2.2 Single Board Microcomputers	8
2.3 Self contained microcomputers	9
2.4 Computer simulation	10
2.5 In-circuit emulators	11
2.6 Mini/Micro communication	14
3. MICROPROCESSOR BUS STRUCTURES	17
3.1 Introduction	17
3.1.1 Virtual memory	21
3.2 Motorola MC68000	22

3.3	The Intel 8086 Microprocessor	25
3.4	The Intel 80286	28
3.5	Zilog Z80 microprocessor	28
3.6	The Motorola MC6800 microprocessor	30
3.7	The Motorola M6809 Microprocessor	31
3.8	The Mos Technology 6502 Microprocessor	32
3.9	The Texas 9900 microprocessor	34
3.10	The Zilog Z8000 Microprocessor	35
3.11	Zilog Z80000	37
3.12	The MC68020 microprocessor	39
3.13	The Intel 80386	40
3.14	Summary	42
4.	MC68000 COMPUTER SYSTEM	46
4.1	The supportive processor overview	46
4.2	The M68010 Microprocessor	49
4.3	The M68451 Memory Management Unit	50

4.4	The Hitachi HD68450 Direct Memory Access Controller	51
4.5	The MC68000 Multi-board computer system	52
4.5.1	The Central Processing Unit	53
4.5.2	The Memory Board	54
4.5.3	The EPROM/ROM Board	55
4.5.4	The Floppy Disc Controller Board	56
4.5.5	Hard Disc Interface Board	56
4.5.6	The Bus Display and Peripherals Board	56
4.5.7	Additional boards	57
4.6	The MC68000 Single Board Computer	58
5.	THE SOFTWARE ENVIRONMENTS	66
5.1	Introduction	66
5.2	The TRIPOS environment	67
5.2.1	TRIPOS filing system	67
5.2.2	TRIPOS Tasks	68
5.2.3	Inter-task communication	70

5.2.4	TRIPOS device drivers	71
5.3	The BCPL programming language	72
5.4	The UNIX environment	73
5.4.1	The development of UNIX	73
5.4.2	The Structure of the UNIX operating system	75
5.4.2.1	The UNIX Kernel	75
5.4.2.2	The UNIX process	76
5.4.2.3	Interrupts and Exceptions	78
5.4.2.4	Inter-process communication	79
5.4.2.5	The UNIX I/O System	81
5.4.2.6	The UNIX file system	83
5.4.2.7	Directory structure	84
5.4.2.8	The UNIX shell	84
5.4.2.9	System boot	86
5.4.2.10	UNIX utilities	87
5.5	The C programming language	88
6.	THE EDUCATIONAL INTERFACE BOARD	89

6.1	Interface specification	92
6.2	Hardware design	93
6.2.1	Address decoding logic	94
6.2.2	Arbitration logic	96
6.2.3	DTACK generation circuit	96
6.2.4	I/O controller	97
7.	Target systems	109
7.1	Target system specification	109
7.2	The Z80 target system	110
7.2.1	Circuit description	111
7.2.1.1	The target I/O facility	113
7.3	The Z80 personality module card	115
7.4	The Z80 target interface	116
7.4.1	Master to Z80 target memory access	117
7.4.2	The Z80 target to shared memory access	119
7.4.3	The Z80 target interrupts	119
7.5	The MC68000 target system	121

7.5.1	Memory maps and manipulation	122
7.6	The M68000 Personality Module Card	126
7.7	Master to MC68000 target memory access	126
7.8	M68000 target to shared memory access	128
7.9	Target Interrupts	129
8.	SOFTWARE/HARDWARE INTEGRATION	144
8.1	The UNIX development software environment	144
8.1.1	Operation procedure	148
8.1.2	Support software	149
8.1.3	The software development cycle	150
8.2	The TRIPOS development software environment	152
9.	CONCLUSIONS	154
	APPENDIX A : Supportive System Bus Specification	159
	APPENDIX B : Z80 Target Bus Specification	163
	APPENDIX C : M68000 Target Bus Specification	164
	APPENDIX D : PAL Equations	165

APPENDIX E : The Educational Interface Board Circuit

Diagram 168

REFERENCES 169

1. INTRODUCTION

1.1 Microprocessor background

The advancement of large-scale integration (LSI) and very large-scale integration (VLSI) technologies have led to the integration of over one million components on a single silicon chip, and the implementation of most functional units of a traditional processor in a small piece of silicon has led to a chip called a "*microprocessor*". A microprocessor is the central arithmetic and logic unit of a computer, which is responsible for the fundamental operations upon which all computer intelligence is based. The term was first introduced in 1972, after the era of microprocessors was heralded in 1971 with the introduction of the Intel 4004, a "*micro-programmable computer on a chip*" [1]. The 4-bit 4004 Central Processing Unit (CPU) contained 2300 transistors and could execute 45 different instructions.

As the earliest microprocessors were 4-bit devices of limited capabilities they were soon followed by 8-bit microprocessors that generally contained a central processing unit control circuitry for the central processing unit, an arithmetic logic unit (ALU) which could perform mathematical calculations, two 8-bit accumulators which are used in "*number crunching*" tasks, a 16-bit index register to access the memory, an 8-bit condition code register which displays the results of the previously executed instruction, a stack pointer which remembers where stored information was held during an interrupt and a program counter that allowed the microprocessor to know

where it is in the program. In order for these microprocessors to perform their functions efficiently, they utilize their instructions in several addressing modes [3].

Although the second generation commenced with the introduction of the Intel 8008 in 1972, the domain of 8-bit microprocessors witnessed several significant improvements in hardware and system concepts with the introduction of the Intel 8080 and the Motorola 6800 in mid 1974 [3]. The advanced 8-bit microprocessors with their 8-bit external data bus usually contain 16-bit internal registers and can easily handle 16-bit words.

The need for increased performance and capabilities called for 16-bit microprocessors. The development of 16-bit microprocessors began in 1974 with the introduction of the PACE chip by National Semiconductor. The Texas Instruments TMS 9900 was introduced two years later. Subsequently, the Intel 8086 became commercially available in 1978, the Zilog Z8000 in 1979, and the Motorola MC68000 in 1980[2].

For most of the present requirements and applications, 8 and 16-bit microprocessors have been successful. They have been used to build systems ranging from simple controllers to complex graphic design workstations. However, there are some applications where more processor speed, larger address space, improved performance, high reliability and functionality are required which can only be obtained by the use of 32-bit processors [4].

Microprocessors with 32-bit internal paths have been in existence since 1980. However, the era of true 32-bit microprocessors begins in 1981 with the commercial introduction of the Intel iAPX 432^[2] (it has been now withdrawn from the market due to its poor sales due to the radical nature of its object oriented architecture). National Semiconductor was one of first manufacturers to introduce a monolithic 32-bit microprocessor the 32032. Soon after that many powerful 32-bit microprocessors came to existence, like the Intel 80386, Zilog Z80,000, the Motorola MC68020 and recently the Motorola MC68030.

The early microprocessors performed basic CPU functions only. However as the microprocessor technology advanced, the integration of a large number of auxiliary functions on the same microprocessor chip became possible resulting in the increase popularity of computers built with very few chips.

1.2 Microprocessor education requirements

During their fifteen years of existence, microprocessors have evolved at a dramatic increase in terms of numbers, technology, complexity, power, functionality and applications. In conjunction with their progress, the power of the processor peripherals and support devices increased rapidly . This enormous technological achievement has introduced major problems and difficulties into the teaching of microprocessor technology to students. The same problem can also be felt by those who educate students in this field.

The design and implementation of any prototype microprocessor based system has to pass through several education and development phases before it can reach the production line and the skills and understanding an Electrical Engineering student requires in order to design a system involving a microprocessor or a microcomputer must be defined.

Preparation and learning is the first step, where it is necessary for the student or the engineer to be familiar with digital techniques. That is, the basic understanding of the functions of logic gates and circuits, switching theory, combinational and sequential logic, wave shaping circuits etc. This stage also requires some knowledge of computer organization and microprocessor design techniques. Usually, the theoretical teaching of the subject to student is well established in the undergraduate curriculum programme. The same knowledge can be gained from the special courses offered to engineers who are without prior knowledge of the subject, or in some cases, by self education. Such courses cannot cover all the available devices nor can they examine all the possible approaches to problem solving. Microprocessor literature, microprocessor and computer magazines, and manufacturers manuals should be consulted regularly for up-to date knowledge.

The next phase is associated with the selection of the most suitable microprocessor for the application.

Since there are so many microprocessors available, one should reach a certain level in appreciation of the abilities of as many

microprocessors as possible before a processor is selected. In order to select the most suitable microprocessor for the job, it is required for the student or the engineer to examine several microprocessor families. If a student or an engineer is only familiar with one processor, he will check whether that processor can cope with the job or not. If that is the case, the processor will be used regardless of its suitability.

However, if the student or the engineer has been introduced to several microprocessor families, he will have the skills and experience to choose the most suitable processor and further can examine new devices for their suitability.

In order to expose students to different range of microprocessor families suitable equipment for the practical sessions in software and hardware development is needed a so called development system. A proper development system may have a keyboard and monitor for input and output, floppy disc drives for storage, system modules such as CPU module, memory module, in-circuit emulator, floppy disc controller module, system firmware and monitoring modules. The cost of such a system is usually high and it is essential to provide sufficient sets of development system for each microprocessor family, for the number of users. The number of users could be high, resulting in unnecessary large investment in equipment. This stage of the development represents the central discussion of this thesis.

Once the processor is selected, a set of questions concerning hardware versus software tradeoffs should be answered. Only then

can the detailed hardware design be started.

The next phase is related to software design, since the highest performance of a microprocessor based system is dependent on the quality of the software provided. This stage could be accomplished by the designer himself or by a software expert. The hardware designer should at least provide the necessary software required for testing and debugging the prototype system.

The last phase of the development is related to the production of a working system which successfully performs the required functions.

The work described in this dissertation can be divided to the following three main stages:

- i. Examination of the currently available microprocessor teaching techniques.
- ii. Design and implementation of the new adapted approach.
- iii. Integration of the hardware with development software environment.

2. A REVIEW OF MICROCOMPUTER EDUCATIONAL SYSTEMS

The rapid growth in the microprocessor population and the increasing use of microprocessors in education has resulted in many different approaches to the problem of microprocessor teaching and development ranging from simple evaluation kits to more complex in-circuit emulators.

This chapter examines the various available techniques for providing microcomputer education and development and discuss the strengths and weaknesses of each approach.

2.1 Evaluation Kits

Evaluation Kits, like the Motorola MEK6800D2 evaluation kit^[5], were originally introduced by the microprocessor manufacturers . They are used to familiarise users with the fundamentals of a specific microprocessor family . In addition to the microprocessor, they contain a small amounts of Random Access Memory (RAM), a resident de-bug monitor, a hexadecimal keypad which is used for input and a multisegment light-emitting display used for output. Only a very limited amount of information can be displayed at any one time, and programs have to be entered in hexadecimal code. The lack of a real editing facilities, together with the limited amount of diagnostic informations that can be displayed, and the necessity of entering programs in machine code, increases the possibility of keying errors and often leads to students frustration.

Although evaluation kits are suitable for gaining familiarity with a particular microprocessor family, they are not really suitable for system educational development or practical applications. As they are designed to be as cheap as possible, they are very difficult to expand.

2.2 Single Board Microcomputers

Like the evaluation kits, the major microprocessor manufacturers all offer single board computer families based on their own products^[7]. The earliest generation of the single board computers had similar facilities to the evaluation kits. With the reduced cost of all types of memory and, with the support of sixteen bit microprocessors, an improved software features and peripheral devices are included in the latest version of the single board computers. The minimum software development facilities for single board computers would include a monitor program to allow users to single-step their programs and if required examine the microprocessor's registers and change memory locations , breakpoint setting etc. An assembler and an editor is provided to construct and correct the input programs. Some single board computer manufacturers support high level languages such as BASIC. Usually such systems would be provided with full QWERTY keyboard, video and storage facility interface.

Single board computers would appear to be a low cost approach to providing a microprocessor educational system, but, by the time additional facilities (such as the QWERTY keyboard, VDU interface,

T.V or a monitor, floppy disc drive, application modules, etc.) are added, they are no longer cost-effective specially if many stations are to be provided for a group of twenty to thirty students. Incidentally, as a result of the added facilities, the board complexity will increase. The true microprocessor architecture will be hidden if students adopt high level language, such as BASIC, at the early stage of microprocessor learning.

2.3 Self contained microcomputers

A number of self contained microcomputers, sometimes called "*boxed computers*" or "*personal computers*", such as Apple II, ACORN BBC, and IBM PC, are in common use now in homes, schools, and universities. They were designed primarily for use as general purpose processors of information. Typical systems consist of the microprocessor, up to 64K bytes of random access memory for 8 bit processors, read only memory (for the operating system and language compiler/interpreter), VDU with graphics capabilities and cassettes/floppy drives for storage and retrieval of information. Sixteen bit processor self contained systems available today have built in Winchester technology disk drives up to forty megabyte, as much as megabytes of random access memories, and can support simultaneously several users. They support many high level languages such as BASIC, Fortran, Pascal and C. Such systems are increasingly being used for universities and business applications in a stand alone mode or connected to a host mainframe/mini computer system. The hobbyist market is also growing for such systems, and

many companies develop extensive software products for use on such systems.

The boxed computer systems tend to provide better debugging facilities than the systems described previously.

The single unit nature of these systems and their compactness makes it difficult to expand them and difficult to interface their bus to external devices for direct memory access purposes.

Such systems are useful for teaching computer concepts and high level languages. But, due to their capital cost, they are not suitable if more than one microprocessor family is to be studied.

2.4 Computer simulation

For some universities, instead of providing microcomputers to allow students to approach the problem of microprocessor education, have run simulators, such as MicroSim^[6], on a host computer system. This approach can successfully make a variety of assembly languages available to the student and can allow a number of students to access the simulator without any difficulty in a multi-user environment. But true input/output programming can not be achieved, nor does such a system provide the student with exposure to the hardware or to the peripheral devices connected to the computer. Although simulators can provide useful software support at all levels, the majority of them cannot provide debugging facilities such as single step or trace capability.

2.5 In-circuit emulators

The introduction of the microcomputer was quickly followed by the realization that highly specialized design aids are required to support microcomputer-based development efforts.

The in-circuit emulators provide the ability to emulate microprocessor operation in real time, where the system operation is intended to be at clock speed and to display register and memory contents to the user for inspection. With accompanying software, emulators can provide an efficient and powerful development tool to integrate hardware and software development during all phases of the development cycle^[8].

There are three categories of in-circuit emulators available^[9].

The first of these is the stand-alone emulator, which can operate independently of the microprocessor development system or the host computer which is used to develop the microprocessor software. Normally, by using RS-232C links the user can download the target-system software into the stand-alone emulator, then he can detach the emulator and use a CRT terminal to control the emulator's operation. This method offers the benefit of freeing a host computer or a microprocessor development system for additional software development while hardware/software integration is proceeding with the emulator.

The second form is the computer-hosted emulator, where a host computer is required for the emulation of a target processor operation. Such emulators like the Microcosm family, supporting the

Intel 8086/186 family of processors, can receive control from an IBM Personal Computer, a DEC VAX or an Intel Series III development system.

The final form is the in-circuit emulator based-development system, where a microprocessor development system is required for the emulation operation. The in circuit emulator is built into the development system to allow the development system to be connected to the microprocessor target board under investigation. Through an emulator cable, which plugs into the target microprocessor socket, the in- circuit emulation based development system can emulate the target microprocessor and have control at operational speeds over all the signals normally controlled by the microprocessor. This powerful technique allows program execution in the system under test to be traced and interrupted by the user at the console of the development system. Furthermore, resources of the development system such as memory and I/O ports can selectively be made available to the target system.

The development system usually provide sophisticated debugging facilities such as single stepping, software tracing, breakpoints setting, and real time tracing. With single stepping, the program is executed one instruction at a time, where memory contents, processor register contents and the next several instructions, after each step, are displayed to the user to check that the results are those expected. Single stepping is a powerful method of preliminary testing, because bugs can be discovered before they can cause any

damage to the program or data. With the updated display the user can alter register and memory contents while stepping through the same segment of code repeatedly and hence can test the code for operation under different conditions. A breakpoint is a trap, set in a program, which allows the program to be executed at full speed or in a trace mode up to where the breakpoint has been set. When the program reach the breakpoint, execution is halted and the development system debugger is in control. Setting breakpoints implies that the user can locate the correct memory address for the breakpoint. Symbolic debugging is a different technique, used by more advanced debuggers, where addresses are referred to as symbolic names, which are defined by the user in his original symbolic source program.

Since microprocessor design is critically dependent on operation in real time, and since single step and software tracing do not provide complete debugging facilities, many development system manufacturers offer logic analyzer or real time trace facilities in to the development system. In a real time trace, the user can connect test leads to a selected number of points on the target board and run the test program to capture data in real time; the traced data cannot include the contents of internal microprocessor registers or of memory.

The latest emulators, such as the SDT816 (Symbolic Debugging Tool for 8 and 16 bit microprocessors) manufactured by Positron Computer Limited U.K, have the ability to assist in symbolic debugging where a symbol table is stored locally. Up to thirty two

hardware breakpoints can be set and the system also provides real time trace facilities. Additional to the system microprocessor, the SDT816 can also emulate coprocessor chips and other system chips as well. As there is an emulator for every related type of microprocessor family, the programs can even be executed and tested before any target board is built.

The in-circuit emulator based-development system is the most powerful technique available today to the problem of multi-family microprocessor education and development, and also the most expensive approach where prices ranging from approximately five thousand pounds to well beyond fifteen thousand pounds per station. The high cost is due to their hardware and software complexity. As a result of the high cost, their main use is in commercial firms, while their use in education is very limited . Another disadvantage is that the in-circuit emulators are sold as a complete package, which will tie the user to the development system manufacturer software only.

2.6 Mini/Micro communication

Another approach to the problem of multifamily microprocessor teaching is where a minicomputer is connected to a target processor by the use of serial or parallel link. Through a terminal, the student can get access to the host computer, run the editor and the cross assembler, then down load the object file to the target processor board for stand-alone execution of programs. In

stand-alone operation, the target board will be connected to a terminal and the student can use a resident monitor program to examine and control the execution of programs [10].

The disadvantage of this approach lies in the capital cost of the host computer plus the complexity of the target hardware.

Another Mini/Micro communication approach is where the target processor board is interfaced to a minicomputer through an externally controlled DMA channel. The technique is based on the processor initializing a counter system which will provide an address for data to be stored or retrieved when ever this is requested. When a transfer is requested, the counter system will take control over the processor bus and will provide the necessary address and control signals needed to complete the transfer.

Such an approach is described by Holdstock [12], where a Motorola M6800 target board is interfaced to a Digital Equipment Corporation PDP 11/20 . Due to the high performance and the high speed of the host computer used, the implementation of the interface to the target processor is forced to be as an input/output device using a DR11C 16 bit input/16 bit output parallel port. This has the disadvantage that the user is tied to a manufacturer supplied cards to provide TTL compatible lines for the target. The high capital cost of the minicomputer is another disadvantage.

The method examined and implemented in this thesis is similar to the one suggested by Whitworth [13]. The suggested technique was

based on providing an interface between a Z80 microcomputer and a target system such that the address, data and control lines of each can be translated to the timings and levels expected by devices attached to the other.

This work describes the design and implementation of an Educational Interface Board (EIB) which will allow the MC68000 based-computer system (master) running the UNIX operating system to communicate to any eight, sixteen , or thirty two bit microprocessor based system (target).

3. MICROPROCESSOR BUS STRUCTURES

3.1 Introduction

As the function of the Educational Interface Board is to provide a healthy environment for any two dissimilar system buses to communicate, this chapter begins with a discussion of the basic requirement needed for a bus to bus interface followed by a study of various currently available microprocessor bus structures.

The basic structure for any computer system would include the following three major components, the microprocessor unit (MPU) or the central processor unit (CPU) (for arithmetic, logic, and control functions), memory, input/output interface for peripheral control, and three system buses, the data bus, the address bus and the control bus.

Regardless of the number of lines the processor may have, the address, data, and control signals must be available.

The address bus is used by the processor to inform memory and other peripherals of the location it requires to access.

The data bus transfers information between the processor and all the peripherals, including the memory.

The control lines carry all the control signals between the control unit of the processor and all other devices that make use of such signals.

Bus communication can be divided into two categories, synchronous and asynchronous.

The synchronous bus requires the information to be present on the bus at the appropriate time. This procedure implies that the timing mechanisms of the source and destination devices are synchronized. Such systems have to be designed to operate sufficiently slowly in order to accommodate even the slowest devices. The disadvantage of such systems is that the timing of the information transfer is determined by the slowest device in the system, hence preventing fast devices from communicating at their high speed. The principle advantage of such systems is their simple structure with less control signals required.

In the alternative approach, the asynchronous communication bus, the bus transactions are terminated as soon as the required data has been passed. An additional control signal is required in such systems in order for the device being accessed to inform the accessing device that the data is available. The accessing device may respond with another control signal to acknowledge the acceptance of data. The timing of the data transfer depends on the speed of the communicating device. This flexibility is accomplished at the expense of a more complex bus control structure.

As the microprocessor fetches and decodes instructions from memory, a number of control signals will be generated to enable the processor to synchronize its functions with the other components in the system. The number and nature of the control signals varies from

one microprocessor manufacturer to another. However, the following control signals are common to most microprocessors.

- a. control signal (or signals) to determine the direction of the data transfer.
- b. A means of a request signal by some peripheral devices to take control over the system bus for direct memory accesses.
- c. A means of grant signal that is to be used by the processor to acknowledge transfer of control to external device.
- d. A means of control signal that is to be used by slow memory or I/O devices to effectively slow down the processor in order for the slow device to complete its task.
- e. A means of interrupting the processor to demand attention and to direct the program from its normal activity into another higher priority service program.
- f. A means of gaining absolute control over the processor. This is usually achieved by the use of RESET control signal, which resets most of the processor internal registers to zero before setting up the program counter to point at some pre-determined location.

As the Educational Interface Board was designed to provide a demonstrative and supportive tool for users, any features which are available on the target processor should be available for investigation. Also, it should be able to monitor and activate all the target control

lines.

In general, the sequence of events involved in processor to memory or I/O transfer is as follow :

- i. A means of selecting a memory location to be accessed by initiating a bus cycle by the processor.
- ii. A means of informing the memory device of the direction of the transfer i.e a read cycle or write cycle.
- iii. Indicators to show that the address and/or the data lines are in a stable state.
- iv. An acknowledge indicator to inform the processor that the device is accepting the transfer.

As there are many processor/memory protocol techniques used, the above features have to be provided by one method or another.

When two dissimilar buses are to communicate, the following points require special consideration :

1. The width of the address and data buses of the two microprocessors concerned.
2. Whether any of the processors is using a multiplexed data and address bus. If this is the case, the interface board should be able to demultiplex and multiplex the buses as required.
3. The basic control signals for bus to bus interface must be available.

4. Voltage compatibility is an important factor to be considered. But since the common microprocessor interfaces and drivers available today are found to be TTL compatible, it is likely that this will be of any major problem.

The next subsection gives a brief introduction to virtual memory, memory management, paging and segmentation. These terms are to be mentioned later when describing various microprocessor bus structures.

3.1.1 Virtual memory

The first generation of microprocessor-based systems were implemented around 8-bit microprocessors. The address range of such systems was limited to 64 Kbytes. But with the decline cost of dynamic RAM (DRAM), it became economically possible to cover the entire address space, not including those spaces which are already occupied by ROM or I/O devices, with RAM. Such implementation has offered one to one correspondence between logical addresses (which are generated by the processor over its address bus) and physical addresses (which are real memory locations where data is read from or stored at). But soon the limitation of 64 Kbytes address range was realized. With today's 16 and 32 bit microprocessors, this address range is no longer a problem, however, other variables (such as cost, size and power consumption) are to be considered. Taking these factors into account, it becomes impractical to cover the address range of such processors with RAM.

Virtual memory has been used to solve such a problem. The virtual memory technique is based on swapping the unused parts of a program between main memory and a secondary memory (such as a hard disk unit) as required. A memory management unit (MMU) is usually used for this purpose. One of its main functions is to translate the logical addresses into physical addresses to give the user the illusion that all the logical addresses are actually implemented.

Two common techniques are used by most memory management systems, they are paging and segmentation. With paging the memory is divided into fixed size blocks, pages, usually between 512 bytes and 2K bytes long^[44]. With the segmentation approach the logical space is divided into segments of varying length. Each scheme has its advantages, paging simplifies the allocation of memory to users and segmentation simplifies protection of different areas of user memory space^[45].

The next sections will present the study of several popular 8, 16 and 32-bit microprocessor bus structures.

3.2 Motorola MC68000

The Motorola MC68000 ^[16] was the first microprocessor to provide a true 32-bit internal architecture for its address and data paths. Externally, the MC68000 has a 16-bit (D_0-D_{15}) bidirectional data bus and a 23-bit (A_1-A_{23}) address bus that directly accesses 16 megabytes of memory. A_0 , the least significant bit of the address bus is used internally to the processor to generate the data size specified

by each instruction.

In simple systems, the MC68000 requires only four output signals to initiate data movement between memory and the processor. These signals are the address strobe (\overline{AS}), the upper data strobe (\overline{UDS}), the lower data strobe (\overline{LDS}), and the read/write signal (R/\overline{W}). In a read cycle, the address strobe signal (\overline{AS}) is asserted to indicate that a valid data address is being output on the address bus. Simultaneously, the \overline{UDS} and \overline{LDS} signals are asserted to enable the selection of either the lower/upper data byte or both bytes. The processor now waits for the participant memory or I/O device to present its data on the data bus and to assert the Data Transfer ACKnowledge (\overline{DTACK}) signal to indicate the completion of the data transfer. This technique is the inverse logic used by most other microprocessors where the processor will complete the read/write cycle within a fixed time, unless the input wait signal is asserted. The major bus interface of the MC68000 is the asynchronous timing of the data bus transfers. With this interface flexibility, the access timing of the processor is dynamically controlled on each bus cycle by the device being accessed via the handshake signal \overline{DTACK} . Thus, devices with vastly different access times can be mixed to perform at maximum speed.

The asynchronous bus structure also handles hardware failures and invalid memory accesses. If an access is made to invalid memory or I/O location, the \overline{DTACK} signal will not be asserted. The MC68000 processor provides a mechanism to ensure that the processor will not

be hung up indefinitely by a device that fails to respond. This is provided by the input signal, \overline{BERR} , which when asserted, causes the processor to enter exception processing to handle the error.

There are three signals associated with the MC68000 bus arbitration scheme, Bus Request (\overline{BR}), Bus Grant (\overline{BG}), and Bus Grant ACKnowledge (\overline{BGACK}). When the processor receives a bus request signal, it responds by asserting the \overline{BG} signal which indicates the bus will be available as soon as the current bus cycle is completed. The external device must wait for the \overline{AS} , \overline{DTACK} and \overline{BGACK} signals to be inactive before it can assert \overline{BGACK} signal and negating \overline{BR} signal to claim the mastership of the bus. The processor output lines then will enter a high impedance state until \overline{BGACK} signal is released by the external device indicating that it is through with the bus. The bus arbitration logic provided by the MC68000 processor is the most comprehensive and straight forward technique to date.

The MC68000 processor can also take advantage of existing M6800 support devices. To ensure bus compatibility, the MC68000 uses three special lines to access the 8 bit M6800 family of synchronous peripherals, they are Valid Peripheral Address (\overline{VPA}), clock Enable (E), and Valid Memory Address (\overline{VMA}) lines. When the M6800 peripheral address is decoded, the \overline{VPA} signal is asserted instead of the normal handshaking signal \overline{DTACK} . The assertion of \overline{VPA} signal informs the processor to become compatible with the M6800 family by waiting for the proper phase of the E clock and then asserting the \overline{VMA} signal to ensure the transfer. The \overline{VPA} signal

serves a different purpose when asserted during an interrupt acknowledge cycle. It indicates to the processor that it should obtain a vector from its table rather than the interrupting device.

The bidirectional \overline{HALT} line of the processor can perform several functions. Like any other microprocessor halt or hold signal, it is used to disable the processor. It is used in conjunction with bus error (\overline{BERR}) signal to indicate to the processor to try running the bus cycle again. Also, the halt signal can be used with the \overline{RESET} line to initialize the MC68000 processor.

3.3 The Intel 8086 Microprocessor

The 8086 was the first 16-bit microprocessor to be produced by Intel. With it came a new generation of business and personal computer era supported by several manufacturers, notably IBM.

The 8088 is another member of the Intel family which is closely related to the Intel 8086. Both support sixteen bit transfers within the processor, and both have twenty address lines to directly access one megabyte of memory. The address lines of the 8086 are multiplexed, like its predecessor the 8085, with sixteen data lines and four status lines in order to have a 40-pin package. However, the 8088 has only eight data lines restricting its transfers with memory and I/O devices to bytes only. Other than the data bus width, both processors support the same instruction set.

The 8086 central processing unit logic has been divided into an

Execution Unit (EU) and Bus Interface Unit (BIU), mainly to allow the processor to fetch new instructions from memory while it is busy executing some other instructions. The two units operate asynchronously and the processor can in most cases overlap the instruction fetch with execution.

An important feature provided for the Intel 8086 is the provision for maximum or minimum mode system. To select the required mode, an input line (MN/\overline{MX}) is tied high or low. When in the minimum mode the processor provides the complete standard microprocessor control signals required to interface memory or I/O devices. In the maximum mode multi-processor system, the processor can support a variety of Intel's co-processors which include the 8089 input/output processor and the 8087 numeric data processor. The minimum mode bus configuration which has less circuit complexity is more appropriate for this study of microprocessor bus structures, so only the minimum mode will be referred to.

There are two address spaces provided by the processor, the Memory/IO (M/\overline{IO}) control signal indicates whether memory or an I/O port is being accessed. The sixteen bit data bus is divided into low and high bytes. A_0 (the least significant bit) of the address bus and \overline{BHE} (bus high enable) signals are used to select the low, the high or both bytes. An Address Latch Enable (ALE) signal is used to identify a valid memory address to allow system components to capture the address information before the same lines carry data information.

The data transmit/receive (DT/\overline{R}) and data enable (\overline{DEN}) are two

new signals not found in earlier Intel processors. They are used to control the direction and output enable of a bidirectional latched buffers on the data bus and are designed specifically to work with an 8236/8287 bus transceivers. The input HOLD signal is used by other devices to request the use of the system buses for direct memory accesses. When HOLD is asserted, the processor enters a hold state after completing its current bus cycle. The processor then asserts the HLDA (hold acknowledge) signal to acknowledge the hold request. The input signal READY informs the processor that the addressed memory or I/O device is ready to complete the current bus cycle. The bus continues to cycle until the READY signal is asserted. This signal is useful when the processor is to communicate with devices of different speeds.

Another innovative feature of the 8086 hardware design is the ability to use it in a wide range of microcomputer system configurations, from a simple one processor system to a multi-processor environment. The processor has built in logic to handle bus access priorities. The signal \overline{LOCK} , which is provided only in the maximum mode, is used in a multi-processor system to prevent a processor from accessing the bus while another processor is reading or writing a memory location. Software single step facility is another feature of the Intel 8086 to support the programming of multi-processor systems.

3.4 The Intel 80286

The 80286 is the second generation of sixteen bit microprocessors introduced by Intel. Several advanced features have been introduced in the 80286 processor, such as memory management mechanism and hardware provision of multi-tasking programming by operating in two modes real and protected. The 80286 is software compatible with its predecessors the Intel 8086 and 80186 when operated in real mode.

The processor uses separate (non-multiplexed) 16-bit data bus and 24-bit address bus. Additional to the buses width, the Intel 80286 has a similar bus structure to that of its predecessor the 8086.

3.5 Zilog Z80 microprocessor

The Zilog Z80 was designed as an enhanced version of Intel's 8080 microprocessor. It is an eight bit microprocessor with sixteen address lines capable of direct access to sixty four kilobytes of memory space. The success of the Z80 processor is due to its capability to execute the entire range of Intel's 8080 software and in particular, to use the popular operating system CP/M (Control Program for Microcomputers) developed by Digital Research.

The Z80 has additional features over the Intel 8080, like an on board refresh counter for dynamic memory, a non-maskable interrupt facility, and a vectored interrupt priority structure. Several Z80 processors are available, offering a range of clock speeds of 2.5

MHZ, 4 MHZ and 6 MHZ. Another improvement over the Intel 8080 is that the Z80 requires only a single 5V supply and single phase clock input.

Similar to the approach development by Intel, the Z80 has separate memory and input/output address spaces. The \overline{MREQ} (memory request) signal is used to select a valid memory address and the \overline{IORQ} signal is used to select a valid input/output address space. \overline{RD} and \overline{WR} signals are used to control the direction of data transfers. In a typical memory read cycle, the \overline{MREQ} signal will asserted when the address bus is stable. Then the \overline{RD} signal is asserted to indicate that the data can be enabled onto the data bus. Depending on the accessing of memory or input/output devices, wait states can be inserted as required. However, not too many wait states can be inserted, if the role of dynamic memory refresh is not to be affected.

The Z80 was the first microprocessor to include a hardware facility for automatic dynamic memory refresh. After each instruction fetch cycle, the refresh control signal becomes active to indicate the start of dynamic memory refresh.

When the Z80 processor has to give up its bus to an external device, the \overline{BUSRQ} signal must be asserted first to request the bus. When the Z80 complete its current bus cycle it sets its address, data, \overline{MREQ} , \overline{IORQ} , \overline{RD} and \overline{WR} lines to the high impedance state and activates \overline{BUSAK} signal to acknowledge the request.

3.6 The Motorola MC6800 microprocessor

The MC6800 was Motorola's first eight bit microprocessor. It has 8 lines of data bus, and 16 address lines to access up to 64 kilobytes of memory space. Unlike the Z80 bus timings, the MC6800 requires a two phase non-overlapping clock $\phi 1$ and $\phi 2$. $\phi 1$ and $\phi 2$ are used as address and data validators respectively. During the first phase of the clock, an address will be placed on the bus by the processor. During the second phase of the clock, the data bus will be active. The implementation of direct memory accesses, refreshing dynamic memories, or accommodating slow memories rely heavily on the clock signal manipulations.

For direct memory access operations, the Three State Control (TSC) input signal can be used. With TSC activated (high), the address bus and R/\bar{W} signal are placed in high impedance state. The Valid Memory Address (VMA) and Bus Available (BA) signals are forced low in order to prevent any incorrect read or write data on any device enabled by the VMA signal. While the TSC line is active, the $\phi 1$ and $\phi 2$ clocks must be held high and low, respectively, in order to delay program execution for DMA operation to take place. But since the MC6800 processor is a dynamic device, internal memories require periodic clock cycles to maintain correct data and the clocks can be stretched for no more than the required periodic cycle of 10 microseconds.

Direct memory access operation can also be provided by completely halting the processor, using the input \overline{HALT} signal, which

stops program execution. The required periodic cycle of the clock inputs of 10 microseconds has to be maintained.

As the $\phi 1$ and $\phi 2$ clocks time the entire M6800 system, any processor that accesses the M6800 system will be affected by the action of the M6800 clock. In the master/slave configuration, the master direct memory access cycle to the M6800 target system must complete within the 10 microseconds limit.

3.7 The Motorola M6809 Microprocessor

The Motorola M6809 is an enhanced version of the M6800 family of microprocessors. The changes are mainly to improve its available software facilities.

The M6809 is an 8-bit microprocessor with 16 bit address bus. Unlike the MC6800, which uses a two phase non-overlapping clock, the M6809 has an internal clock, which is triggered by an external crystal, to generate two quadrature output clocks E and Q. The E clock phase, which is identical to $\phi 2$ of the M6800, gives a synchronizing signal to be used as the system's clock for support devices. The Q phase of the clock is available to signal that the address and data leading edge of Q and data is latched on the falling edge of E.

The input signal MRDY is used to stretch the E and Q clock signals to enable the processor to interface with slow memory devices.

For direct memory accesses, the input signal DMA/BREQ is used. When activated, it causes the processor to be suspended at the end of the current instruction to enable direct memory access operations. The direct memory access operation is timed with the E and Q clock signals, so that the required periodic cycle of 10 microseconds is still applied.

A second version of the M6809 family is the M6809E, which has external clock inputs E and Q. The M6809E uses the TSC input signal to force the processor into high impedance state for direct memory accesses or dynamic memory refresh.

Three other status signals are available for the M6809E. The Last Instruction Cycle (LIC) output signal is activated during the last cycle of an instruction. The BUSY signal is used to indicate that the processor is performing functions which should not be interrupted by other external devices. The Advanced Valid Memory Address (AVMA) output line will inform that the processor will use the buses in the following cycle and efficient bus sharing in multi-processor configuration can be allowed.

3.8 The Mos Technology 6502 Microprocessor

The 6502 is the most popular of the 6500 family of 8-bit microprocessors manufactured by MOS Technology. The 6502 processor has made its major success in the home computer market with the leading manufacturers Apple, Acorn BBC and Commodore.

The 6502 was produced as an enhancement of Motorola MC6800 microprocessor. It has similar CPU concepts and bus structure. The 6502 popularity was due to its increased performance with two index registers and a more powerful set of addressing modes.

Similar to all 8-bit microprocessors, the 6502 has eight data lines and sixteen address bus. In a similar way to the M6800 the 6502 uses two phase non-overlapping clock signals to control system timing. During the first phase the processor sets up a valid memory address and selects the data direction using the R/\bar{W} line. Data is then transferred during the second clock phase.

There are two major differences between the bus structure of the 6502 and the MC6800. Unlike the M6800, the clock pulses of the 6502 can not be stretched, therefore, the 6502 has to use a different accesses or refreshing dynamic memories. The 6502 control input signal RDY, which performs the task of M6800 TSC, DBE and \overline{HALT} signals, causes extra machine cycles, wait states, to be inserted within the normal machine cycle. For wait machine cycles to occur, the RDY signal must make a high-to-low transition during a phase one high clock pulse. The external device can hold RDY signal low for any required time delay. In addition the 6502 processor has no control signals that can force it into high impedance state, and the processor address and data buses must be latched during any direct memory access operation.

The SYNC output signal is used by the processor as an indication of the instruction fetch cycle.

3.9 The Texas 9900 microprocessor

One of the early 16-bit microprocessors to appear on the market was the 9900 produced by Texas Instruments. The 9900 has been designed as a one-chip implementation of the CPU of the 990 series minicomputer. The 9900 has been a very effective processor for signal processing applications.

The 9900 has a separate 15-bit address bus and a 16-bit data bus in a 64-pin integrated circuit package. The 9980 is a reduced pin count version of the 9900 with only 8-bit data bus and 15-bit address bus.

The standard 9900 processor requires a four phase non-overlapping clock, where none of them is used for address or data validator signal.

For a typical read cycle, \overline{MEMEN} (MEMory ENable) output signal is used to indicate its a memory access cycle. It is also used to differentiate between memory or I/O accesses. The DBIN (Data Bus In) signal is activated to indicate the beginning of a memory read cycle, when data should be placed on the bus by the memory device. The \overline{WE} (Write Enable) signal is used if it is a memory write cycle to validate data to be written to memory.

The \overline{HOLD} input signal is used to force the processor into high impedance state for direct memory accesses; the external device can assert the \overline{HOLD} line active for as long time as it require. The processor acknowledge the request by asserting HOLDA (HOLD

Acknowledge) line.

To accommodate slow memory devices, the READY input signal is used to indicate to the processor to insert wait state cycles as required.

3.10 The Zilog Z8000 Microprocessor

The Z8000 family of sixteen bit microprocessors is available in two versions, the Z8001 and the Z8002. The Z8002, 40-pin package device, can directly access sixty four kilobytes of memory. The Z8001, 48-pin package, is a more advanced processor and capable of addressing up to eight megabytes of external memory. Other than the difference in the address range, the two processors are closely related to each other. Both processors have time-multiplexed address and data bus to minimize the pin count of the microprocessor package.

The Z8000 processor architecture utilises a sixteen bit word organization. Each word of memory is made up of two independently accessible bytes. The Z8002 uses a sixteen bit address to specify one of 32K words of memory, where both bytes in a word are independently accessible. The least significant bit of the address bus, A_0 , is used to specify an even address byte ($A_0=0$) or an odd address byte ($A_0=1$). The Z8001 uses the concept of segmentation to increase its address space, where the address map is seen to consist of 128 memory segments per memory address space with each segment having a 64K bytes. The Intel 8080 provides similar segmentation

facility, but can only access up to one megabytes of memory.

The Z8000 processor can operate in one of two different modes, system mode or normal mode. A control bit in the flag and control word (FCW) indicates the operation mode. In the system mode all instructions can be executed, while in the normal mode only unprivileged instructions are executed. The distinction between the system and normal modes of operations allows the implementation of a multi-tasking facility, where instructions that can directly affect the system hardware or can terminate all the programs are privileged instructions, which should not be executed by the user. In contrast, the Intel 8086 offers no equivalent hardware logic for multi-tasking facilities but does provide similar facilities using software method.

The Z8000 processor has two special control lines dedicated to a multi-processor environment. The Multi-micro Input (\overline{MI}) signal is used by the Z8000 processor to prevent other processors from accessing the bus while it is performing critical manipulations, and the Multi-micro Output (\overline{MO}) signal is used to disable the Z8000 processor while another processor is in charge of the bus.

For a typical data transfer cycle, an Address Strobe (\overline{AS}) signal indicate a valid address, a Data Strobe (\overline{DS}) signal shows valid data, a Read/Write (R/\overline{W}) signal is used to select the direction of the transfer, a Byte/Word (B/\overline{W}) signal to select the size of the data field being transferred, and a Normal/System (N/\overline{S}) signal is used to indicate the current operation mode of the processor. Unlike the Z80, the Z8000 Memory Request (\overline{MREQ}) signal is used to select a memory

space or input/output space.

This processor has a more flexible dynamic memory refresh capabilities than its predecessor the Z80 microprocessor.

For direct memory accesses, the external device can request the control of the bus by asserting a Bus Request (\overline{BUSRQ}) signal, and, when the processor is ready to relinquish the bus, it activates the Bus Acknowledge (\overline{BUSAK}) signal to acknowledge the request. The combination of the \overline{BUSRQ} and \overline{BUSAK} signals provide the processor with hold state logic^[18]. The \overline{WAIT} signal can be used by external devices to increase the delay between the address strobe and data strobe during bus transactions.

The \overline{STOP} input signal can be used to halt the processor operations and can also be used to provide externally single stepping logic for programs under development.

3.11 Zilog Z80000

The Z80000 is the latest generation of Zilog microprocessors. The processor features 32-bit advanced architecture which directly supports operating systems and high level languages. The processor characteristics and facilities are merely an extension of the Z8000 family with new added features as on-chip memory management and small on-chip cache memory. The Z80000 has full 32-bit address and data time multiplexed lines, and can directly address up to 4 gigabyte of memory.

The bus status and time signals used by the processor to perform asynchronous data transfers are similar to those used by its predecessor the Z8000. The address strobe (\overline{AS}) signal indicates that the address and bus status signals are valid. The data strobe (\overline{DS}) signal is used to time all data transfers. The read/write (R/\overline{W}) signal is used to select the transfer direction. Two status signals (BL/\overline{W} and BW/\overline{L}) are driven by the processor to specify the size of the data (byte, word or long word) involved in the transfer operation. Four status output signals (ST0-ST3) are used to encode the type of bus cycle (such as internal operation, I/O transaction, halt and \overline{NMI} acknowledge) performed by the processor. The external logic can then decode this information and respond in a number of different ways.

The processor architecture supports two control buses, local and global. The local bus consists of the two familiar signals \overline{BUSREQ} and \overline{BUSACK} that are used by external devices to gain mastership over the processor buses for direct memory accesses. In a multiprocessor environment, the Z80000 can request the mastership of a global bus by asserting the global bus request line (\overline{GREQ}) and obtains the response of the bus arbiter via the global acknowledge signal (\overline{GACK}).

During each data transfer, two response (RSP0-RSP1) signals are used by external hardware to return a code to the processor indicating ready, wait, bus error, or bus retry. The ready response informs the processor of a successful transfer. The wait response tells the processor that the responding device requires more time to complete the transfer, other wait cycle is then added before the sampling of the

response lines again. Wait states can also be inserted by programming the hardware interface control register (HICR).

The Z80000 architecture includes 256 byte of high speed cache memory used to speed up the processor operations. The cache memory can be disabled for debugging purposes by using the control bit in the system configuration control long word register (SCCL).

3.12 The MC68020 microprocessor

The MC68020 microprocessor is the full 32-bit implementation of the M68000 family architecture. Its address bus is capable of accessing a large linear (not segmented) address space of four gigabyte of memory. The MC68020 architecture is merely an extension of earlier processors in the family.

As the MC68000 processor, the asynchronous bus structure of the MC68020 uses a 32-bit address and data buses that are non-multiplexed for simple interface design and high performance.

The MC68020 bus interface includes a new feature, dynamic bus sizing, which allows the processor to communicate with 8, 16 or 32-bit devices through the use of the Data transfer and Size ACKnowledge input signals ($\overline{DSACK0}$ and $\overline{DSACK1}$). The $\overline{DSACK0}$ and $\overline{DSACK1}$ signals replace and perform the same function as the \overline{DTACK} control signal of the MC68000 processor, they also inform the CPU of size of the port being accessed. Full compatibility with the reduced data buses of earlier processors in the family has been maintained by

the dynamic bus sizing facility.

The MC68020 contains an instruction on-chip cache memory which improves the overall performance of the processor by reducing instruction access time. A cache disable signal (\overline{CDIS}) is used to disable the activity of the cache memory. The cache memory can also be disabled by programming the cache control register (CACR). For debugging purposes, when the processor is forced to access the external memory to monitor the behaviour of the software and hardware under test, it is essential for the cache memory to be disabled.

The MC68020 has a similar bus operation as that described earlier for its predecessor the MC68000 microprocessor.

3.13 The Intel 80386

The 80386 is the full 32-bit implementation of high performance microprocessors developed by Intel.

The processor internal structure is divided into six functional units.

- i. The bus unit. Interfaces the CPU to the external system bus and controls all address, data, and control signals to and from the processor.
- ii. The prefetch unit. Responsible for fetching instructions from memory.

- iii. The decode unit. Prepares instructions for processing by the execution unit.
- iv. The execution unit. Executes the micro instructions.
- v. The segment unit. Translates logical addresses to linear addresses and performs bus cycle segmentation violation checks.
- vi. The paging unit. Translates the linear addresses generated by the segmentation or prefetch unit into physical addresses.

The internal units of advanced processors, such as the M68020 and the Intel 80386, are normally pipelined in order to enable them to operate in parallel on different instructions.

The 80386 has separate 32-bit address and data buses. Its data bus can be switched between 16 and 32 bits to allow existing 16 bit devices to communicate with the processor. The instruction set of the 80386 supports byte, word and long word transfers. Four byte enable signals (BE0-BE3) are used with the address bus to specify the data bytes that are active.

The 80386 uses only one signal, address status (\overline{ADS}), to inform external logic of the beginning of a normal bus cycle. The processor, then, defines the type of bus cycle with the W/\overline{R} , M/\overline{IO} and D/\overline{C} signals.

The 80386 provides bus lock (\overline{LOCK}) signal of multiprocessor applications. The lock signal informs other bus masters that the processor is performing a multiple bus cycle operation that must not be interrupted.

The processor can run two kinds of bus cycles, non-pipelined and pipelined. The non-pipelined bus cycle is used when the processor is communicating with high speed memories. The pipelined bus cycle is used to give slow memory systems more time to respond to a bus cycle. Pipelining is enabled as external devices assert the next address signal (\overline{NA}).

The 80386 uses the \overline{READY} , HOLD and HLDA signals in similar way as described for the Intel 8086.

3.14 Summary

In this chapter, various microprocessor bus structures have been examined.

For some microprocessors the data and address bus organization is multiplexed. Such processors (e.g the Intel 8086) transmit instructions and addresses over a single 8 or 16 bit system bus. In all cases, multiplexing is used to reduce pin requirements of the chip package. Extra hardware interface would be required in order to communicate to such devices.

Almost all current microprocessors have provision for a direct memory access facility to allow transfers between devices and memory without processor intervention.

Some microprocessors, such as the M6800 processor, use memory-mapped input/output. Others, as the Z80, use certain control lines to distinguish between memory and I/O operations.

The Z80 microprocessor was used by Whitworth [13], in a similar study, as the supportive processor. The Z80 is short of many important hardware features which include the following :

- i. The Z80 address range is limited to access only 64K bytes of memory. Without implementing any form of memory management unit (MMU), the processor would be unable to access the address range of 16-bit processors. If MMU is to be implemented, the hardware interface complexity would increase further.
- ii. The Z80 data bus lacks the ability to store and manipulate different types of data. This would make it more difficult for the Z80 to communicate with 16-bit data buses. To provide this facility, the hardware interface complexity would increase even further.
- iii. The Z80 arbitration circuitry lacks the facility to connect several processors into one system.

Therefore, 8-bit processors in general are not suitable of supporting the multi-family microprocessor teaching project.

In this study, the MC68000 microprocessor is used as the supportive processor for the following reasons :

1. The MC68000 has one of the most comprehensive non-multiplexed bus structure available to date.

2. Its powerful addressing capability enable it to access any target memory location.
3. The ability of the processor to store and manipulate different types of data enable it to support 8-bit devices on its 16-bit data bus.
4. The asynchronous timing of the MC68000 bus enables even the slowest target memory to communicate with the supportive processor.
5. The synchronous interface option provided by the MC68000 allows the MC6800 peripherals to interface with the supportive processor.
6. The processor bus arbitration logic enables multiple processors and DMA controllers to share the same bus.
7. Halt and Bus error signals are available that may used to single step the bus, abort illegal or invalid access attempts. This is vital to successfully recover in the event that interface circuits cause a deadly embrace.
8. A 3-bit function code signal is present that identifies the purpose and privilege level of each bus cycle.
9. The available 3-bit encoded interrupt request input allows six prioritized, maskable interrupts and one non-maskable interrupt, with 255 vectors to transfer control to the proper interrupt handler routine.

All the above features have dramatically contributed toward the reduction of the interface hardware complexity and the design of this interface will be discussed in chapter 6.

Next chapter will provide a hardware description of the supportive system, the MC68000 computer system.

4. MC68000 COMPUTER SYSTEM

This study was carried out using a multi-board MC68000 computer system as the supportive system in this master/slave multi-family microprocessor teaching project. The multi-board system, which was commercially marketed under the name of DARKSTAR, had been designed by the school of Electrical Engineering at Bath University. The system was well established, and considerable hardware and software support is already available.

This chapter gives an overview of some members of the M68000 family of microprocessors, followed by general description of the main hardware elements which make up the multi-board computer system, and finally a glance at the Single Board Computer (SBC) system.

4.1 The supportive processor overview

The supportive system is based on a powerful 16-bit MC68000 microprocessor. The MC68000 microprocessor was the first member of the M68000 family of microprocessors to be introduced by Motorola in 1979 [17].

The MC68000 microprocessor provide a true 32-bit internal architecture, while externally it has a 16-bit data bus and 24-bit address bus. The processor can run at up to 12.5 MHZ clock rate. The bus structures of the MC68000 processor has been discussed in the previous chapter section 3.2.

Internally the processor offers eight 32-bit data registers (D_0 - D_7), eight 32-bit address registers (A_0 - A_7), two 32-bit stack pointers, a 32-bit program counter, and 16-bit status register. The data and address registers are all general purpose and are not dedicated to specific tasks, with the exception of A_7 which is defined as the hardware stack pointer. Any data register can be used as an accumulator and any address register can be an index register.

The MC68000 processor supports 56 powerful instruction types, which can operate on five different data types, namely individual bits, binary coded decimal (BCD) digits, 8-bit bytes, 16-bit words and 32-bit long words. The instruction set contains no increment or decrement commands. Such features can be achieved by the use of ADD and SUB instructions where the destination operand can be any register (data or address) or a location in memory.

The instruction set of the processor contains instructions to perform, data movement, integer arithmetic, binary coded decimal arithmetic, logical operations, shift and rotate operations, bit manipulation operations, program control operations and system control operations. The processor also supports 14 different addressing modes which fall into several basic types, register direct, register indirect, immediate, and implied. More detailed information about the instruction set and the addressing modes can be found in references [18-20].

All I/O devices in an M68000 system are memory mapped, where no special I/O instructions or separate I/O bus is required.

The processor has a powerful interrupt structure of seven priority levels with 256 interrupt vectors, most of which are available for handling vectored interrupt exceptions. Exceptions can be divided into two priority groups. The highest priority group of exceptions are reset, bus error (when an accessed location fails to respond) and address error (when the processor attempts to access a word or a long word at an odd address). These force exception processing to start at the next bus cycle. The lower priority group of exception are caused by trace conditions (which provide useful software debugging facility by setting of breakpoints and single stepping), hardware interrupts, illegal instructions, instruction traps and privilege violations. When an exception occurs, the processor calls a service routine to handle that exception. The lowest 1024 bytes of the MC68000 memory are reserved for holding the addresses for all these routines, where each address is held in 32 bits long slot known as an exception vector. Each vector has a number associated with it which represent its byte address divided by four. During an interrupt acknowledge cycle, an 8-bit vector must be supplied, on the data bus (D_0-D_7), by the interrupting device in order for the processor to locate the interrupt service routine.

The MC68000 operates in one of two privileged states, user or supervisor. The supervisor state is the more privileged, and any instruction can be executed while in this state. Usually, programs that are associated with the operating system only are run in the supervisor state. All other programs can be run in the user state

except several key instructions (such as STOP and RESET), which are protected from access by the user. Any attempt to execute them will cause a trap which will pass control back the operating system. This privilege distinction is very useful in an operating system environment where the user should not have direct access to operating system handling information.

4.2 The M68010 Microprocessor

The MC68010 was the third member of the M68000 family to be introduced by Motorola in 1982. Internally, the M68010 has the same 32-bit M68000 architecture, and externally the 16-bit data and 24-bit address buses of the M68000. The processor has a slightly larger instruction set than the M68000, and instruction execution is generally faster on the M68010.

The MC68010 architecture design goal was to provide virtual memory and virtual machine support (which had been implemented in mini and mainframe computer for many years) for the M68000 family.

To have virtual memory capabilities, the processor has to be able to suspend any task at any point and then restart or continue the suspended task at a later time. In order to provide virtual memory support, a processor must be able to perform several basic functions. They include recognition of a fault, saving any fault related and internal information and execution of the exception handler, and restoring the saved state and resuming normal execution. The

MC68000 provided some of these features such as fault recognition, state save, and exception handler execution. The MC68010 has all these features together with the ability to save the complete internal state, restore the state and resume execution.

4.3 The M68451 Memory Management Unit

The M68451 Memory Management Unit (MMU) was designed, by Motorola, to work with the M68000 family of processors.

All memory management systems begin with memory mapping, the translation of logical addresses into physical addresses. Logical addresses are the addresses which are visible to the user and manipulated by the software. Physical addresses are the bit patterns transmitted to the memory to identify the memory location to be accessed. Memory management system translate logical to physical addresses according to mapping tables, which indicate that certain blocks of logical addresses are to be translated into certain blocks of physical addresses. The logical address space of the M68451 is divided into variable sized segments of 256 bytes or more. Each MMU device has 32 descriptors which can be used to define the segment size. For each segment an address translation is performed in order to produce a physical address. More than one M68451 can be combined in a system to provide more power and flexibility.

4.4 The Hitachi HD68450 Direct Memory Access Controller

The HD68450 Direct Memory Access Controller (DMAC) is designed to complement the performance and architectural capabilities of the M68000 family of microprocessors by moving blocks of data between memory and an external storage device, in a quick manner with minimum intervention from the processor. The HD68450 has four channels which work independently of each other, and has signals which are directly compatible with those of the M68000 bus and those of the M68451 memory management unit.

The main purpose of a direct memory access controller in any system is to transfer data at very high rates, usually much faster than a microprocessor, under software control. The term DMA is used to refer to the ability of a peripheral device to access memory in a system in the same way as a microprocessor does.

Direct memory access requests may be externally generated by a device or internally generated by the "auto-request" mechanism of the DMAC. Auto-requests may be generated either at the maximum rate, where the channel always has a request pending, or at a limited rate determined by selecting a portion of the bus width to be available for DMA activity. External requests can be either burst requests or cycle steal requests that are generated by the request signal associated with each channel. The rate of transfer of data is limited both by the memory response times and the device response times.

4.5 The MC68000 Multi-board computer system

The research of this project was carried out using the M68000 multi-board computer system running under two different software environments. The original system was implemented with the TRIPOS operating system. At a later stage the UNIX operating system was implemented. The system running under the TRIPOS operating system will be referred to as system A, while the system running under the UNIX operating system as system B.

For the purpose of this study, the minimum hardware elements required for system A would include one MC68000 based CPU board, a minimum of 256 Kbytes of dynamic random access memory (DRAM), monitor and/or bootstrapping firmware stored in erasable programmable read only memory (EPROM), a floppy disc controller board, an 8 inch floppy disc drive, a bus display and peripherals board with front panel offering reset and non-maskable interrupt facilities, and an RS232-C asynchronous serial port for terminal connection. System B would require the upgraded MC68010 CPU board with two memory management units (MMUs) and direct memory access controller (DMAC) on board, 1/2 Mbytes of DRAM, a hard disc interface board, a hard disc drive, bootstrap firmware stored in EPROMs, a bus display and peripherals board, and an RS232-C for terminal connection.

The multi-board computer system was implemented in a double Euro-card standard rack. These racks have either 9 or 22 slot double Euro-card to provide the necessary expansion space for future

development work. A system block diagram is shown in Figure 4.1.

In the following subsections, a brief hardware description of the main boards is given. More detailed description can be found in Tanner^[21] and Williams^[22].

4.5.1 The Central Processing Unit

At the heart of the system lies the Central Processing Unit (CPU) board which built around the Motorola MC68000 microprocessor (M68010 is used in later versions of the CPU board). The board contains all the bus drivers and controls to enable the processor to communicate and control operations on the backplane. The function of the processor board can be divided into the following logic sub-functions :-

- a. Address and data control.
- b. Control line generation.
- c. Halt, reset and interrupt acknowledge state machines.
- d. Function code and interrupt request decoding/encoding.
- e. Memory map decodes.
- f. Clock, bus timeout and timing strobes generation.
- g. Buffer control and interrupt acknowledge traps.
- h. Power up reset and halt.

The board occupies the first physical end position on the

backplane, with the necessary resistance terminators, to reduce the noise levels on the backplane, are provided. The interrupt request lines are "daisy-chained" through the backplane giving the highest priority interrupt level to cards nearest to the processor card.

The CPU card have the facility for two on-board MC68451 Memory Management Units. The Memory Management Unit (MMU) provide the address translation and protection over the whole of the MC68000's 16 megabyte address space. Each MMU provides 32 separate segments of variable sizes which can be used to separate User and Supervisor memory, program and data spaces. The MMU can also provide memory management facility for other bus masters such as direct memory access controllers. This type of facility provides the basis for multi-tasking/multi-user operating systems by providing full protection for individual users. The later version of the CPU board offers on-board HD68450 Direct Memory Access Controller (DMAC) for direct memory access facility. A simplified block diagram of the CPU board is shown in Figure 4.2.

4.5.2 The Memory Board

Next to the processor board lies a quarter of a megabyte of Random Access Memory (RAM) board. The main memory array consists of up to thirty-two 64K bit dynamic random access memory (DRAM) devices arranged in two banks of 64K x 16 bit words. Each bank has an additional bank of 64k x 6 DRAM devices which are used to store check words generated by the error detection and correction unit. The dynamic random memory devices used on the

memory board have access times of 200 nanoseconds and require a multiplexed address bus.

The memory board features full error detection and correction facility with various modes of operation. It can detect and correct errors without informing the processor, bus error the system if double or single bit errors are detected, or bus error the system only when double bit errors are detected. A typical access to the memory board will consist of the following sequence, read the memory array, detect and correct errors, generate new check bits, and then write back to the memory device. The error detection and correction adds a delay of 60 nanoseconds to the memory access time, making a typical memory cycle time of approximately 500 nanoseconds when an 8 MHz CPU board is used.

With the advancement of the memory technology, later memory boards had been designed with capacities of 1/2 megabyte, 1 megabyte and 2 megabyte with error detection facilities. A schematic diagram of the memory board is shown in Figure 4.3.

4.5.3 The EPROM/ROM Board

The Eprom/Rom board provides all the non-volatile data to the processing unit. The card was designed to contain upto 16 Eproms/Roms in any size from 1K by 8 to 8K by 8. The Eprom/Rom size is switch selectable and the board base address can be anywhere within the 16 megabyte address space on a boundary defined by the current memory size of the card. Thus, the Eprom/Rom board can

support a variety of Eprom/Rom devices to supply either 16K, 32K, 64K or 128K bytes of nonvolatile data. A block diagram of the EPROM/ROM board is shown in Figure 4.4.

4.5.4 The Floppy Disc Controller Board

The Floppy Disc Controller (FDC) board is based on a Western Digital chip set FD1793-02 Formatter/Controller device [24]. The controller is capable of supporting upto four 8 inch or 4.25 inch double or single sided disc drivers, with single or double density recording format. The controller support a wide range of controller functions such as disc formatting, single and multiple sector read or write, reading and writing of entire tracks and performing any head seeks required before read or write access. Figure 4.5 shows a schematic diagram of the FDC board.

4.5.5 Hard Disc Interface Board

A hard disc controller board has been designed to provide a mass storage facility for the system. The hard disc controller board contains a Marksman interface to control upto two Winchester technology disc drives with capacities of 40 or 160 mega-bytes. The later disc interface used the Adaptec ACB-4000 series Winchester disc controllers with the Maxtor XT-1000 series Winchester disks.

4.5.6 The Bus Display and Peripherals Board

At the other physical end of the backplane lies the bus display and peripherals board. This board contains two M6850 asynchronous

communication interface adapters (ACIAs) with eight individually switch selectable baud rates from 110 baud to 9600 baud. The ACIAs drives two serial I/O RS232 channels. A MC6840 Programmable Timer Module (PTM) is used on board to provide three independent counter/timer channels. These timers can be used as event counting, period measurement, frequency measurement or watchdog timers. Each timer can be clocked externally or internally connected to the 8 MHZ system E clock.

The bus display card is connected to a front panel and contains diagnostic light-emitting diodes to show the user the current logic state of the processor backplane. The front panel also contains Reset and Abort (non-maskable interrupt) switches which are debounced by the display card before passing to the backplane. A block diagram of the bus display and peripherals board is shown in Figure 4.6.

4.5.7 Additional boards

The following add-on peripheral boards are also have been designed for the MC68000 multi-board computer system.

- i. Floating point board, for additional mathematical ability. The board contains four AM9511/AM9512 Floating Point Processors.
- ii. General purpose I/O board with battery backed real time clock.
- iii. High resolution colour graphics controller board. The high resolution colour graphics board is based on the Thompson EF9366 colour graphics controller and features 2 pages of 512 x

512 pixels in 8 colours.

- iv. High speed interprocessor communications bus for multi-processor applications.
- v. SASI standard interface for secondary disc storage devices.
- vi. Multi-Link local area network card. Multi-link is a low cost ring type local area network which provides virtual character stream data links between network stations. The multi-link network provides a file transfer mechanism to other computers and access to shared printers and plotters.

4.6 The MC68000 Single Board Computer

During the course of this study a single board implementation of the multi-board system was designed by Dale [23]. The Single Board Computer (SBC) maintained compatibility with the multi-board system peripheral cards, and had used the original backplane as its communication medium.

The single board computer was designed to operate in a stand alone mode, either as a complete computer system, or as an intelligent controller. It was also designed to provide the necessary arbitration and inter-processing signalling to allow several single board computers to be fitted to a common backplane to produce a multi-processor system environment.

The single board computer contains the following hardware

facilities :-

- a. MC68000 or MC68010 microprocessor running at 12.5 Mhz clock.
- b. Two MC68451 MMU.
- c. HD68450 DMA controller - for high throughput to I/O devices and memory to memory copying.
- d. Parallel interface and timer providing a SASI interface.
- e. Floppy disc interface.
- f. Dual RS232 serial I/O channels.

More detailed information about the hardware development of the computer system can be found in Dale^[23].

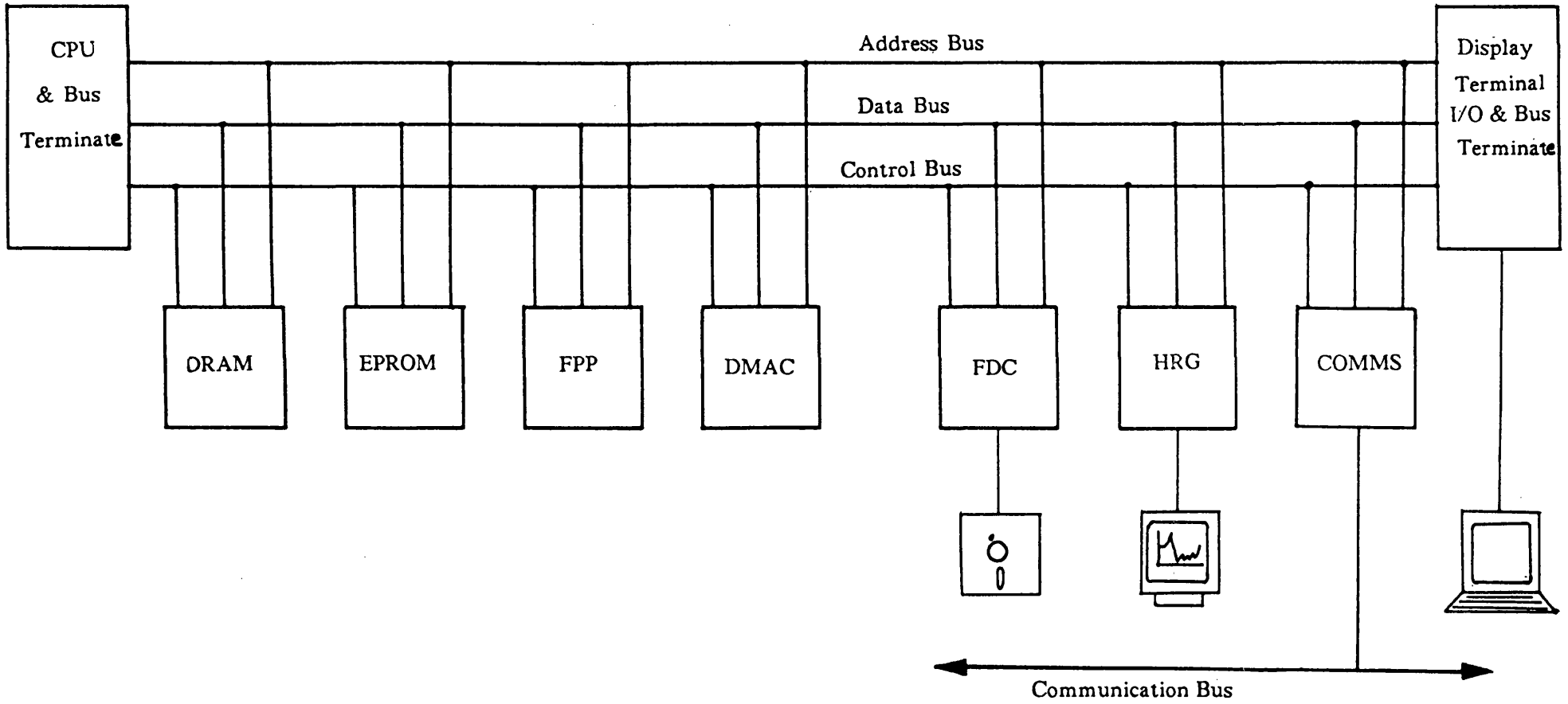


Figure 4.1 Multi-board System Architecture

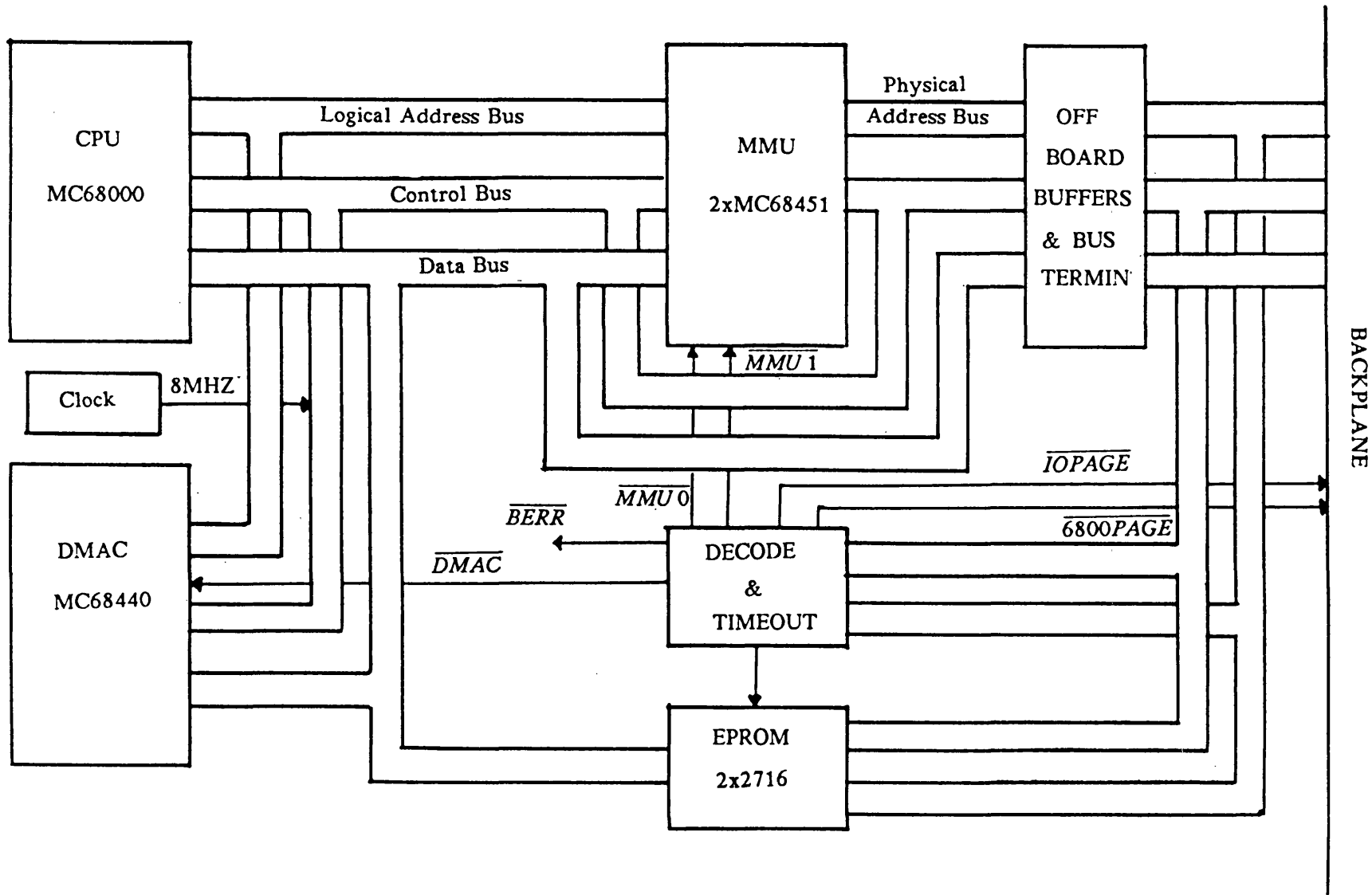


Figure 4.2 CPU Board Architecture

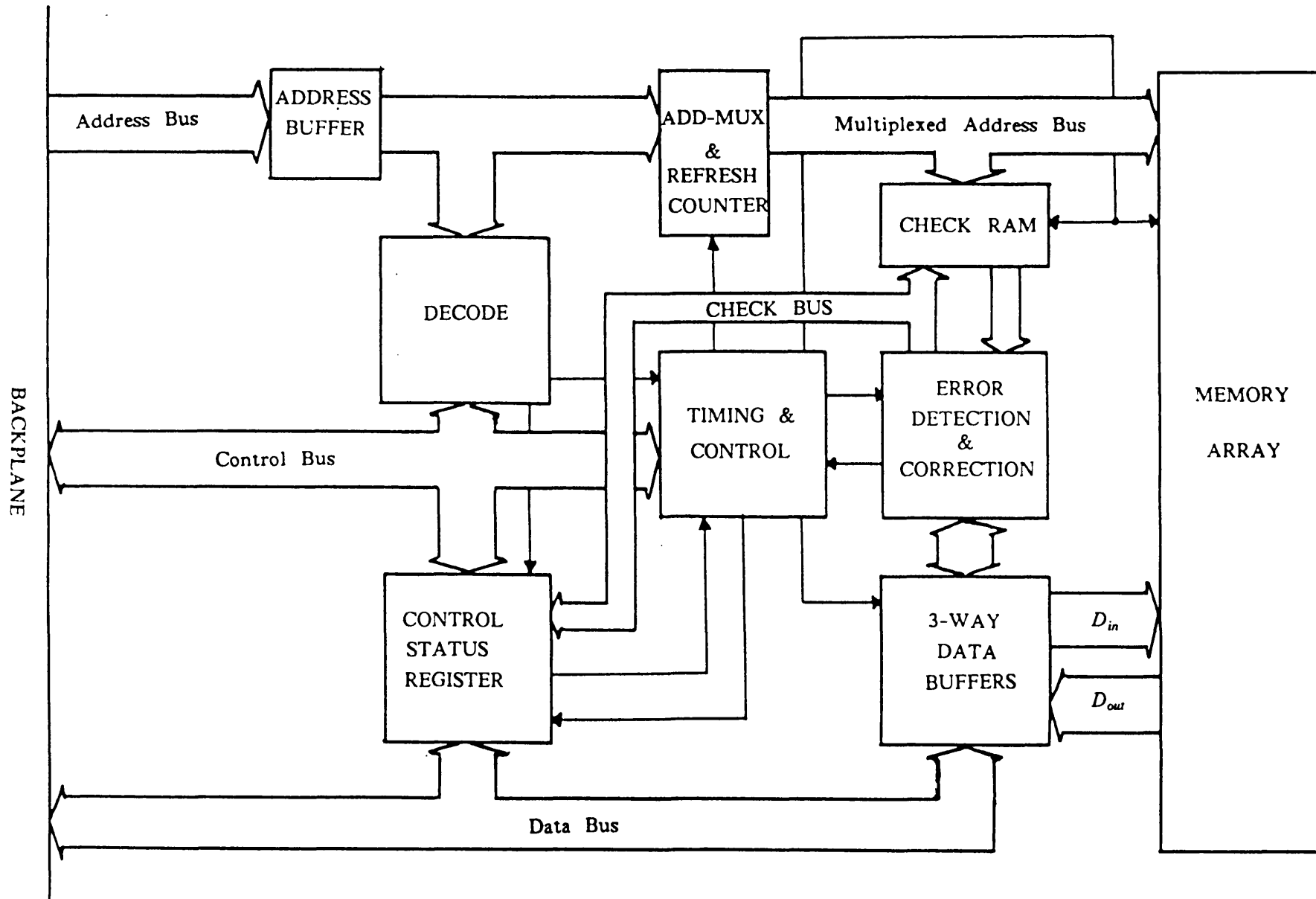


Figure 4.3 Memory Board

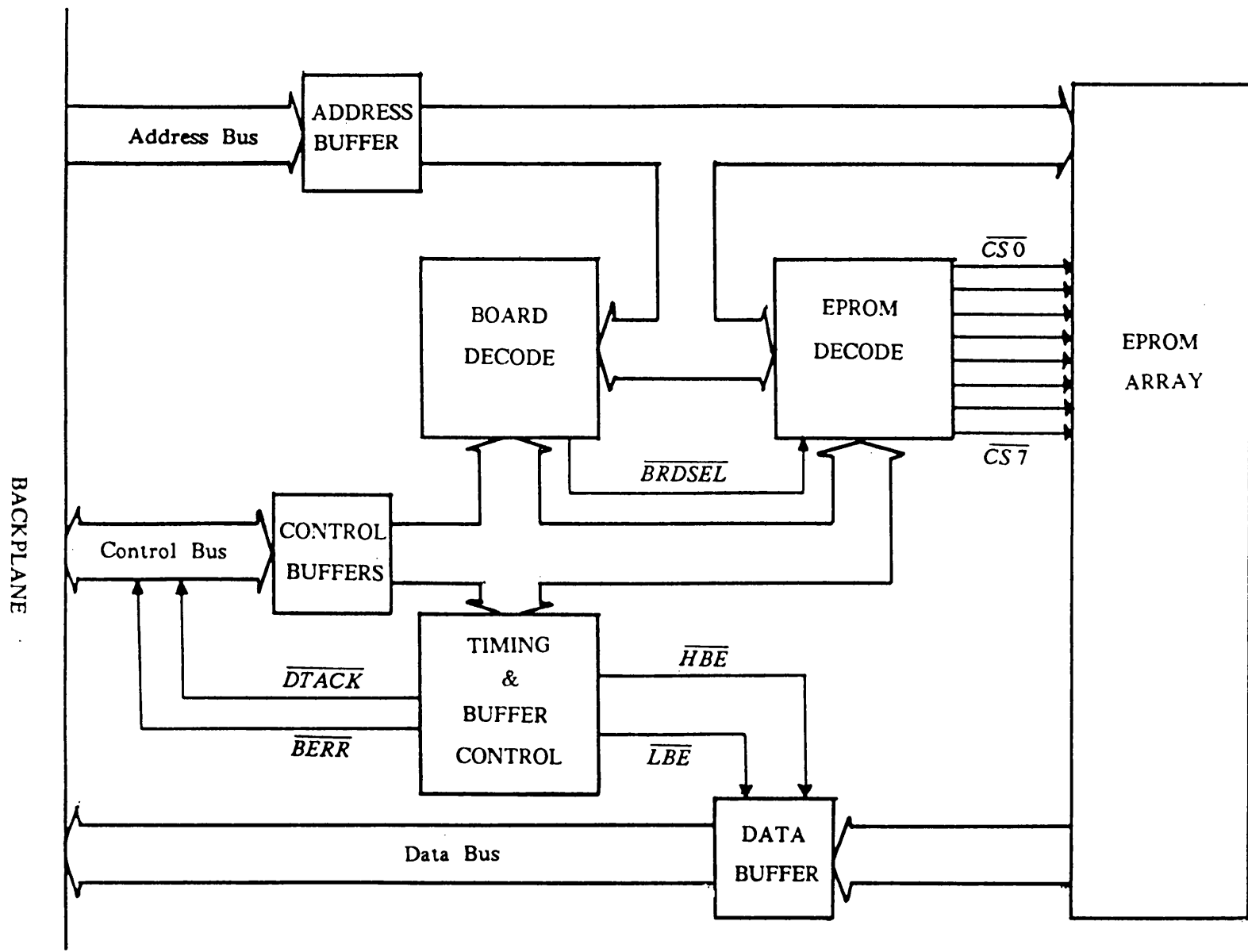


Figure 4.4 EPROM/ROM Board

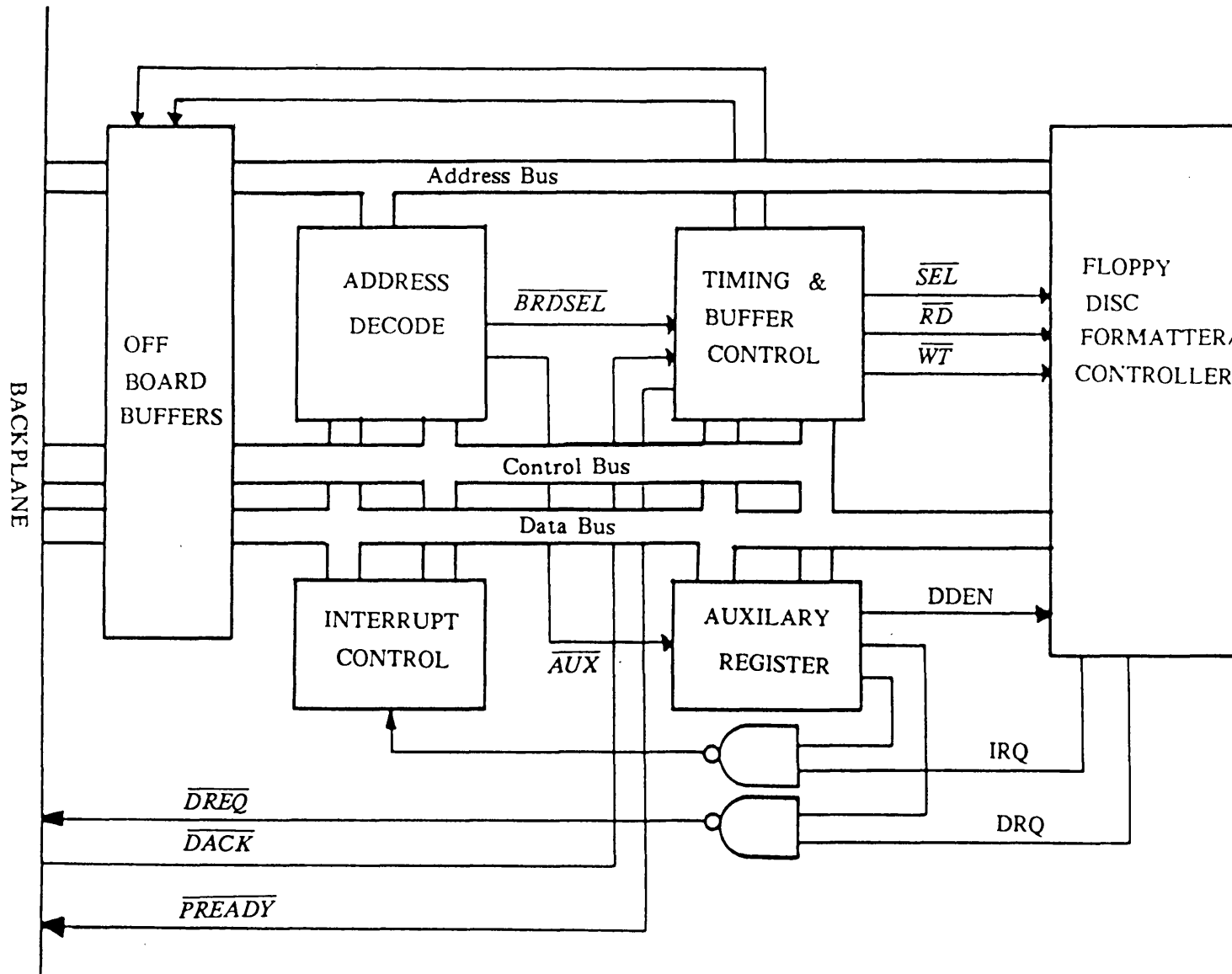


Figure 4.5 FDC Board

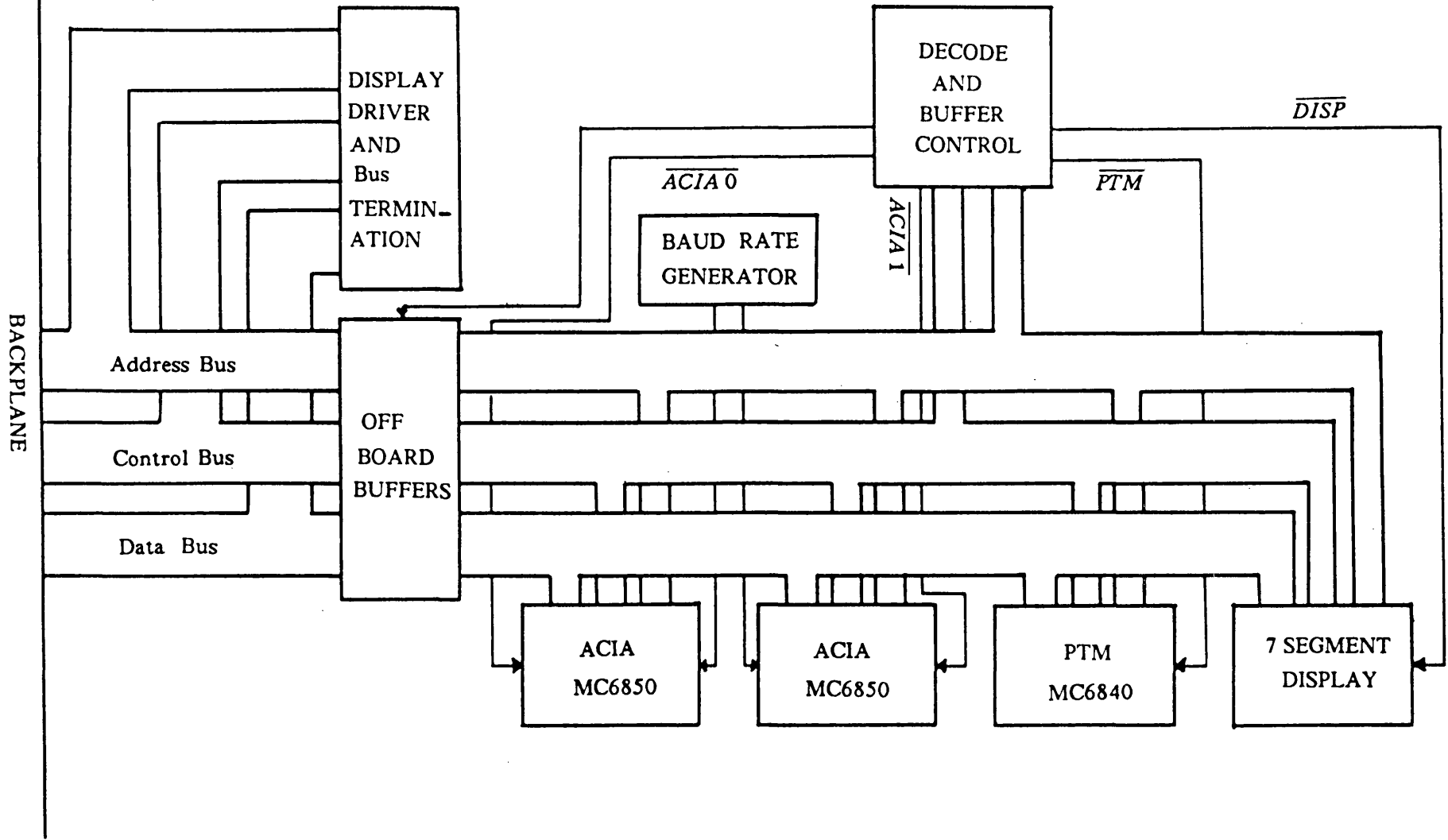


Figure 4.6 Display Driver and Peripheral Board

5. THE SOFTWARE ENVIRONMENTS

5.1 Introduction

An operating system has been defined as "*programs implemented in either software or firmware that make the hardware usable*" [43]. These are the programs which allow the interaction between the user and the machine. The operating system is also a resource manager; it manages processors, storage, input/output devices, and data.

The programs that the operating system consists of can be divided into two main categories, the system Kernel and the applications software. The Kernel consists of those programs which interact directly with the hardware, providing common services to programs such as processor and memory allocation, interrupt handling, I/O control and file management. The applications programs are those which perform general functions as editors, assemblers, compilers and text formatters.

The supportive system used in this research, the MC68000 computer system, offers the full power of two powerful operating systems, TRIPOS and UNIX. The two operating system environments were used for program development and hardware testing. Each of the software environments provide a powerful program editing and development. They also maintain a cross-assembler for each target microprocessor supported. Various of high level languages, such as BCPL and C, are also supported.

The two operating system environments and their programming languages are described in the following sections.

5.2 The TRIPOS environment

TRIPOS is a single-user multi-tasking operating system, originally developed at the Computing Science Laboratory at Cambridge. It was designed as a portable operating system in order to be implemented in different computer systems, such as LSI-4, PDP-11, Nova^[28] and MC68000 based computer systems. Most of the operating system is written in the system programming language, BCPL, with only some system primitives such as device drivers and the task scheduler written in assembly language.

A wide range of utilities and programming tools are supported by the TRIPOS environment. Compilers for languages other than BCPL, such as Fortran, ALGOL and Pascal, are available. A number of cross assemblers to support a variety of eight and sixteen bit microprocessors, other than the MC68000, are also available. The main utilities which were used in this project include the text editor, the BCPL compiler, the MC68000 Macro Assembler and the Z80 cross assembler. Detailed information about the TRIPOS utilities can be found in the Tripos User Guide^[25] and the Tripos Programming Guide^[26].

5.2.1 TRIPOS filing system

The objective of a filing system is to provide a facility for storing data in groups and to logically connect them in way such that

they can be easily accessed.

Any filing system should be able to provide the following general functions:

- i. Create and delete files.
- ii. Open and close files.
- iii. Read/write data from/to files.
- iv. List the contents of a file.
- v. Rename and copy files.

Additional to the general features mentioned above, TRIPOS offers a tree structure filing system for both directory and user files. The filing system is implemented as a task called the filing system task or handler, which is responsible for managing data files on secondary devices such as floppy discs and Winchester discs.

5.2.2 TRIPOS Tasks

The standard TRIPOS operating system is generally loaded with the following tasks:

- i. Command Line Interpreter (CLI).
- ii. Debug task.
- iii. Console Handler.
- iv. Filing system task.

A user interacts with the operating system through the CLI, which interprets the command lines received from the console handler via the terminal device driver and attempts to execute them.

The console handler task is used to coordinate all input and output with the terminal device. The data entered at the terminal is directed, by default, to the CLI and the output is to appear at the terminal. By using escape sequences the console handler can redirect the input to any specified task and, in particular, the debug task.

The debug task provides an interactive debugging tool for examining and modifying task variables, monitoring CPU registers and memory, setting break points, program code disassembly and single stepping facility. The debug task can run in two modes, as a TRIPOS task when accessed through the console system by typing the escape sequence, or in a stand alone mode. The later mode is entered following the execution of an exception routine, which signal hardware failure, or a *TRAP* instruction. While the debug task is in the stand alone mode, it is impossible for the operating system to continue.

When a disc is mounted for either writing or reading, a restart task is created. The task is responsible for checking for the validity of the disc structure. Until the validity check is completed, the disc is write protected.

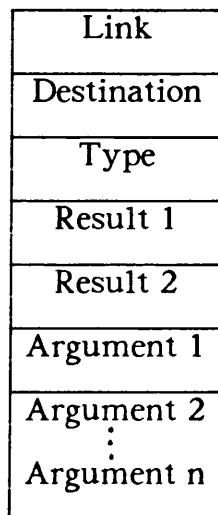
As each TRIPOS task represents a particular function of the operating system, there is allocated a priority for each task. The scheduler is responsible for organising task execution according to priority levels. Only one task is allowed to run at a time, while other tasks could be either waiting for something to occur, such as line printer acknowledgement or data to arrive from disc, or have been

interrupted and waiting to continue execution.

When a task is created by the *CREATTASK* primitive, a unique positive number, known as the task number, is assigned to it. The task number is used to index the task table where a pointer to the Task Control Block (TCB) can be found. Each TCB is associated with a particular task. The TCB contains all the information, such as a priority level and linked list of packets, which is required by the system to schedule and control the task.

5.2.3 Inter-task communication

An integral feature of TRIPOS is its message passing system. It is a mechanism where the communication between two tasks (or a task and a device) is achieved by sending packets. The Kernel manages the transfer of packets between tasks and devices by using the primitive calls *QPKT()*, which queues a packet, and *TASKWAIT()*. The field structure of a TRIPOS packet is shown below :



Triplos Packet Structure

Packets can be linked together on a work queue by pointing the link field of one packet to the link field of the next. The destination field of the packet contains an integer number which is used to identify the destination of the packet. If the integer is positive, the destination where the packet is to be send is a task. If it is negative, the destination is a device driver. The type field contains a number which indicates the type of action required by the receiving task or device. The result fields are reserved for values which will be returned to the packet originator. These values concern the completion or failure of the requested action. The argument fields represent any extra messages which might be needed by the task or the device to which the packet is sent.

5.2.4 TRIPOS device drivers

Unlike the UNIX operating system, TRIPOS is a simple system to add devices to.

Each device driver requires five assembly code routines. An *INIT* routine is used to initialise the device in order to be ready to receive packets. The *INIT* routine is called either when the device is created or during the initialisation of the operating system. An *UNINIT* routine is required when the device driver is to be removed from the operating system. A *START* routine is called to initiate any new packets which are sent to the driver. A *STOP* routine is called to cancel the processing of packets. Finally, an *INT* routine is used to provide any necessary action which is required to service a packet or

an interrupt from the physical device.

Similar to tasks, each created device driver is assigned a device number (negative integer) which is used to index the device table in order to allocate a pointer to the Device Control Block (DCB).

5.3 The BCPL programming language

The Algol report^[36] which was published in 1963 made enormous progress in the design of programming languages, especially the implementation of block structure and the stack mechanism. Since then, several new languages have been developed which have adopted some ideas from the Algol report including the block structure technique. One of these languages was CPL (Combined Programming Language) developed at London and Cambridge Universities. A detailed description of the CPL language can be found in reference [37].

CPL had led to the invention of a family of languages which include BCPL, B and C. They have been proved to be of a suitable use in compiler-writing and system programming^[38].

BCPL (Basic CPL) was designed by M.Richards in 1967 [28]. It was designed as a simplified CPL. The most important simplification of the language is its single data type, unlike other programming languages where data variables have to be declared (e.g integer, real, character). The user is free to store any type of data in the variables of his program.

BCPL is a block structured system language, where each source program consists of one or more compiled modules. Modules communicate through the use of a stack and a global vector. The global vector contains all the declared global variables and the pointers to all global functions and routines. This arrangement makes the linking of the compiled modules very fast and also eliminates any need for *GOTO* statements.

BCPL provides standard control flow primitives such as *SWITCHON* and *IF* for selection and *WHILE* and *FOR* for iteration. It has a well defined library of useful functions and routines. The functions are immediately available as BCPL function calls in the standard library.

BCPL is a powerful language under TRIPOS, since the majority of the operating system is written in this language. Further detailed information about the language can be found in reference [39].

5.4 The UNIX environment

UNIX describes a family of computer operating systems developed at the Bell Laboratories, and it a registered trade mark of AT&T.

5.4.1 The development of UNIX

UNIX was originally developed at Bell Laboratories, in 1969, by members of a research group led by K.Thompson to provide a flexible

and powerful environment for software development^[31].

The original UNIX was produced for the Digital Equipment Corporation PDP-7 minicomputer and was written in assembly language. One of Thompson's colleagues, D.Ritchie, designed a high level language called C in 1973, and, as the C language evolved and became suitable, UNIX was rewritten in C and implemented on the PDP-11/40 computer system^[31]. Since that time, UNIX and the great majority of software developed for use with UNIX has maintained use of the C language.

By writing the majority of the system in a high level language, the operating system becomes easy to read, understand, change, and move to other machines thus makes the problem of implementing it on a new host machine much less time consuming than if the whole system were written in assembly code. The portability of the system together with the advent of the powerful 16-bit microprocessors, have led to the popularity of UNIX operating system among a wide range of mini and micro based computer systems.

In 1973, the UNIX system and its utilities became only available to educational and research institutes for a nominal fee, which had assisted in the growing popularity and enhancement of the system in later years.

In 1981, AT&T released UNIX system III as their first commercially supported version. And in 1983, UNIX system V was

released and the latest version to be released at the time of writing is UNIX system V.3.

There are several versions of the UNIX operating system currently in use and supported worldwide. The most notable of these are, UNIX 4.2 developed at Berkeley, UNIX system V and V.3 developed by AT&T and XENIX developed by Microsoft.

The supportive system used in this project is implemented with UNIX version V.

5.4.2 The Structure of the UNIX operating system

To implement UNIX on a new machine, the minimum hardware configuration would include a processor of at least, a minimum of 256K bytes of main storage, a simple memory management unit, a high speed disc drive and a terminal connected to a serial interface.

UNIX is a multi-tasking, multi-user time-sharing, operating system. It consists of a Kernel and commands.

5.4.2.1 The UNIX Kernel

The Kernel is the program at the heart of the operating system which manages the system resources by providing a hierarchical file system, handling interrupts, allocating main memory for an executing process, controlling input and output, scheduling processes for execution on the CPU, and many other functions.

The Kernel permanently resides in primary memory and occupies the lowest memory locations.

There are two levels of execution modes supported by UNIX system, user and Kernel. The UNIX kernel routines are always executed in the privileged Kernel mode, where they can have access to all device registers, system and user addresses and can execute any instruction. All other program code can be executed in user mode with the exception of privileged instructions and certain accesses as direct input or output. The user program can perform input/output accesses by calling the Kernel via a trap instruction, which when executed, changes the execution mode to Kernel.

5.4.2.2 The UNIX process

A process is defined in the Unix literature as a task in various different states of execution. Many processes can appear to be executing simultaneously as the Kernel schedules them for execution. Each process is allowed to read or write its data, but it can not read or write to other processes. Processes can communicate with each other and with peripheral devices by using system calls, which will enable the user processes to access the Kernel facilities in a controlled manner.

In the UNIX system, the Kernel allocates the following four memory segments for every process. i) The process header, which is not directly addressable by the user process, and contains information which describe the attributes of the process. ii) The text segment

which contains the re-entrant executable machine code for the process. iii) The data segment which contains both the initialised and the uninitialised data. iv) The stack segment which contains the stack of the process when it is running in the user mode.

The Kernel identifies each process by its unique process number, called the process identification number or *PID*. And the Kernel contains a process table with an entry which describes the state of every active process in the system.

New processes can be created by using the *fork()* system call. This call requests the operating system to make an identical copy of the process invoking the fork system call. The process that executed the fork system call is the parent process, where the new created process is the child process. Every process has one parent, but it can have many child processes.

A process can be in one of the following states, where each state has its own characteristics which describe the process.

- i. The process running in user mode.
- ii. The process running in Kernel mode.
- iii. The process is in ready to run state. Processes in this state are waiting for the scheduler to determine which process to run next.

iv. The process is in sleeping state.

A sleeping process is a process which is waiting for an event to occur, such as data from a slow device, waiting for I/O to complete from a peripheral device or waiting for a process to exit. The code and data of a sleeping process can either be resident in memory or swapped out to disk in order to provide more space in memory for other processes. This technique allows many processes to run on a system with limited main memory resources.

Processes on a UNIX system are terminated by executing the *exit()* system call.

5.4.2.3 Interrupts and Exceptions

An exception condition refers to an unexpected software interrupt which causes a break in the normal execution of a process, and control is transferred to an exception handler. Exceptions are different from interrupts, which are caused by asynchronous events that are external to a process.

When an exception occurs, the Kernel checks the validity of the process, saves an image of the state of the current process and transfers control to the exception handler. After the handler completes its service, the Kernel restores the state of the current process and proceeds with the process execution. The UNIX system

uses the same procedure to handle exceptions and interrupts.

5.4.2.4 Inter-process communication

For many multi-tasking operating systems, such as TRIPOS, inter-task communication is provided mainly by two mechanisms, message queues and system data areas.

In the message queues mechanism, the Kernel stores messages on a linked list (queue) for tasks to communicate with each other. This mechanism enables tasks to suspend execution on a queue to wait for other tasks to read or write from the queue. The system data area is the other mechanism where tasks communicate with each other via global area which is accessible to all tasks. This mechanism allows large quantity of information to be shared between tasks.

Communication between processes in the UNIX system V is achieved by the use of signals, pipes, shared memory segments, inter-process messages and semaphores.

Signals are used to interrupt the execution of a running process and to synchronise a process execution with other events. Processes may send each other signals by using the system call *kill()*, or the Kernel may send signals to processes on detection of an abnormal exception such as an illegal instruction. For the UNIX System V, there are nineteen signals. Some are associated with process memory violations and others are used to inform the occurrence of events within the Kernel, such as when the user hangs up a terminal.

A UNIX pipe is a file which allows the transfer of a stream of data between processes in a first-in, first-out manner. Pipes also allow the synchronisation of process execution. Processes can redirect their standard output to a pipe to be read by other processes. Users can communicate with the pipe communication channel by the use of system calls for files, such as *read()* and *write()*. The synchronisation between reading and writing processes is maintained by the Kernel.

There is another kind of pipes, which is supported by UNIX system V, called named pipes. Named pipes are identical to the pipes mentioned above, except in the way that a process initially accesses them.

Shared memory segments represent a mechanism which allows processes to communicate directly with each other via a common memory. Each segment is mapped into the data space of the process which is linked to it and is accessed as a data segment.

Inter-process message queues represent another mechanism which allows communication between processes via the use of queues (linked lists) which are maintained by the Kernel.

The inter-process semaphore facility provides semaphore system calls to allow processes to synchronise execution. An implementation of semaphores is described by the Dijkstra Algorithm^[33].

5.4.2.5 The UNIX I/O System

In the UNIX system, peripheral devices are presented to the user through a uniform interface. This interface is known as a device driver. Device drivers are self contained pieces of code to allow a process or the Kernel to communicate with peripheral devices, such as disks and terminals. The Kernel manages these devices by dividing them into two types, block devices and character devices.

Block devices are associated with disks and magnetic tape type devices, where input and output transfer is performed in structured fixed size blocks of data.

Character device interface is used by devices which use unstructured input and output transfer such as terminals. Disk and tape drivers can also be referenced as character devices.

Under the UNIX system, device drivers are treated as files. The interface between the Kernel and the device driver is achieved by the use of the following five system calls: *open()*, *close()*, *read()*, *write()* and *seek()*.

The open system call is the first step a process must take to access a file. The notation for the open system call is as follow:

$$fd = open(filename, flags, modes);$$

Flags indicate whether reading, writing, or both are to be performed. Modes gives the file permissions if the file is being

created. The open system call returns an integer known as a user file descriptor (*fd*) which will be used in references to the file. The Kernel follows the same procedure for opening a device as it does for opening files.

The close system call is used by a process when it no longer needs to access an open file. The syntax for the close system call is:

```
close(fd);
```

where *fd* is the file descriptor for the open file.

Reading from a file or writing to it can be accomplished using the following system calls:

```
number = read( fd, buffer, count );
```

and

```
number = write( fd, buffer, count );
```

Buffer is the location of data in the user process into which the input will be placed, count is the number of bytes the user requires to read, and number is the actual number of bytes read.

As files consist of a sequence of characters, reading from a file or writing to it is often sequential. However, the system call *seek()* allow processes to access a file in a non-sequential manner by adjusting the offset within the file. The notation of the seek system call is as follow:

position = seek(fd, offset, reference);

Offset is a byte offset, reference indicates from which position the offset should be considered, and position is the returned byte offset which where the next read or write will start.

5.4.2.6 The UNIX file system

The UNIX file structure is hierarchical (tree structured), where directories can contain other directories as well as ordinary files. The top directory of the tree structure is called the root directory. From the root directory (node), the user can reference any other node in the filing system.

A UNIX file system on disk consists of a sequence of logical blocks, each containing 512 bytes. The first block on the device is called the boot block and is reserved for the system bootstrap program which is read to boot and initialise the operating system. The second block is called the super block. It contains all the information about the block structure of the device such as the size of the disk, file system name and list of free blocks. The third block in the file system contains the '*i-node*' list. Each *i-node* represents one file or directory and contains the following information concerning the state of the file or directory:

- a. Time of creation, time of modification, time of last access.
- b. Size of file in bytes.
- c. User and group that the file belong to.
- d. Number of links to the file.
- e. Type of entry: file, directory, a block or character device.
- f. Nine permission bits which are used by the operating system to provide security of file information on UNIX.

The remaining blocks in the file system are free storage area, and are used for file data.

5.4.2.7 Directory structure

The root directory in the UNIX file system is referred to as is a directory of files. Files at the leaf node of the tree are either directories, regular files, or special device files. A name of a file is given by a path name that describes how to locate the file in the file system hierarchy.

5.4.2.8 The UNIX shell

The shell is the UNIX command line interpreter (CLI) mechanism for communication between users and the system. The shell program is usually executed by users after logging into the system. The shell program is not part of the Kernel. It is not permanently a resident in main memory, it can be swapped as

required, and can be modified to a particular environment.

The shell provides each program it executes with three open files, input file, output file, and error output file. These files are usually assigned by default to terminal devices, but they can be redirected to any file or device as needed.

The shell is both a command line interpreter and a command programming language. It provides many features, such as input and output redirection and pipes. The redirection of input and output is achieved using the following command:

```
ls > newfile
```

where *ls* is a command for printing a list of the file names in the current directory. The '>' instructs the shell to close the standard output and open the file 'newfile'. All the output generated will be redirected to the 'newfile'.

Pipes are another feature of the shell program, where the output of one program can be connected to the input of another.

The hierarchical file system structure and the shell command interpreter are two major advantages that UNIX provides over most microcomputer operating systems. And the many features of the shell, had also contributed to the popularity and flexibility of the UNIX operating system.

5.4.2.9 System boot

The bootstrapping process can be defined as follow: loading the Kernel into main memory, initialising the system and starting execution.

The bootstrapping procedure goes through a series of stages in order to get a copy of the Kernel into the main memory. First the bootstrap procedure reads the boot block of a disk, then loads it into the memory for execution. As a result, a copy of the Kernel will be loaded into the memory and the Kernel takes full control. When the Kernel start running, it begins an initialisation phase which includes clock, memory, drivers and system tables.

After initialisation, the Kernel mounts the root file system onto root directory, and spawns a single process from a file called '*init*'. When *init* is executed, it connects its standard input and output to the default console terminal for reading and writing, and it forks to create a shell for this console device. The shell, which acts as a command interpreter, allows the communication between the console terminal, operated by a user, and the operating system. As the console terminal is the only active device in the system at this stage, the system is said to be running in a single-user mode.

To bring the system into multi-user, *init* is informed to create a *getty* process for each terminal device in the system which is going to be active. When the user *ID* is entered, the *getty* process executes a

process called login. Login prompts for user password and, if it is correct, a shell is executed. The system remains in multi-user mode until it is instructed to enter single-user mode when receiving a 'handgrip' signal from another process.

5.4.2.10 UNIX utilities

The UNIX utility environment contains a wide range of software tools, including a program checker (*lint*) and a source code management utility (*make*). The make command is a very useful tool that allows the software developer to build new versions, or re-create old versions, of a complex software application in a semi-automatic fashion.

The UNIX environment provides a set of programs called the Source Code Control System (SCCS) whose main function is to reconstruct, update and retrieve any previously released version of a program.

Another utility which is currently supported under UNIX at Bath is a program called Omnia, which was originally developed for the POLESTAR system. Omnia is a universal two-pass assembler, it currently provides assemblers for several microprocessors MC68000, MC6800, Z80, Intel 8086 and 6502.

5.5 The C programming language

As stated in section 5.3, C is a general purpose programming language originally developed for the PDP-11 under UNIX. One of its first uses was to rewrite UNIX operating system which was previously written in PDP-11 assembly code. The C language is a portable machine-independent, very productive software development, high level language.

In contrast to BCPL (which is a typless language that supports only one object, the machine word) C is a typed language that provides different basic data objects such as integers, characters and floating point numbers. Other derived types include pointers, unions and structures.

One of the important features of the language is its support of pointers to other data such as variables and functions. Pointers are variables which contain addresses of other variables. C also provides pointer arithmetic and type conversion on pointer assignment.

Under the UNIX operating system, C has a rich software utility environment, which include lint and make. The C language would be less successful if it was used under other operating systems, such as CP/M or MS-DOS, that lack such facilities.

A detailed information about the C programming language can be found in the book by Kernighan & Ritchie^[40].

6. THE EDUCATIONAL INTERFACE BOARD

This chapter begins with a discussion of some methods used for dual-processor communication followed by description of the Educational Interface Board (EIB) specification and design.

The interface design for dual-processor communication is based on the concept that the available microprocessors have a mechanism of releasing control of the bus to an external device to perform direct memory access operations.

There are several methods by which processors can communicate with each other. For example, it is possible to interface processors by a serial link, as shown in Figure 6.1, or parallel bus for direct communication. Both types of communication could be straightforward and easily implemented, but have several disadvantages. If the two processors vary in their processor speed, then the fast processor can over run the slower processor thereby resulting in a delay or loss of data [41]. Also, parallel communication requires complex synchronization procedures, and the cost of implementing such protocol is high [42]. The communication between the processors in this type of interface is not based on the release of bus control by one processor in order for the other to perform direct memory accesses, and neither of the processors can control the operation of the other. Such a scheme is not suitable for this study where the supportive processor is required to evaluate and examine

target processors and to directly retrieve data from the target memory without assistance.

The straightforward scheme that satisfies the concept that each processor has a mechanism for releasing bus control to an external device is shown in Figure 6.2. In this scheme each device is capable of signalling for bus control. When the request is granted, the requested processor can directly take control over the bus. The two processors share a common bus so that each processor may access the memory of the other. If the two processors used are of different type, then control signal conversion would be essential. The disadvantage of such scheme is that no buffers are employed which will limit the execution to only one processor at a time. Bus conflict between the two buses can occur as a result of directly connecting the buses.

To prevent bus conflict and to allow for simultaneously independent operation each system bus must be buffered. Such a scheme is shown in Figure 6.3. Although the two processors can execute programs simultaneously and they share common resources, neither of the processors can access the local memory of the other. This facility is important if the supportive processor is to evaluate and examine the target processor.

The scheme adapted in this study is shown in Figure 6.4. It is similar to the scheme suggested by Whitworth [13] for eight bit supportive processor.

The design of each target system should be as simple as possible with enough random access memory on board for independent

operations.

Since the function of the EIB is to allow the supervisory system to evaluate and communicate with the target system and to control and monitor its interrupt, \overline{HALT} and \overline{RESET} lines, the target system is not required to perform direct memory access to the supportive system. On the other hand, the target system can communicate with the supervisory system through the common communication area, that of shared memory. The one way direct memory access operation simplifies the interface design and prevents target processors from slowing down the supervisory system.

The use of shared memory in a multi-processor system is useful for passing large blocks of data and for providing hold and work with shared data.

The EIB is designed to be universal in order to adapt to any currently available microprocessor based system. When plugged into the supportive system, the EIB will allow users to evaluate a variety of microprocessor family based systems. This approach will provide the needed hardware to serve as an economical evaluation tool for target systems and will demonstrate a performance-to-cost ratio which is very favourable to educational institutes.

6.1 Interface specification

It is necessary that the interface board support and provide the following hardware facilities :

- a. Direct memory access into the target memory and I/O locations by the supervisor processor.
- b. Each processor must perform its own operations and both processors may run simultaneously.
- c. A communication area, shared memory, accessible by both processors on first come first served basis must be available. Access will be delayed only if both processors attempt to access the shared memory simultaneously.
- d. An arbitration circuit must be used to prevent bus collision during shared memory accesses and to grant access to the processor with higher priority.
- e. Wait-state generation logic must be available for each processor. If the shared memory is in use by one processor, the wait generation logic is responsible for suspending the other processor from accessing the shared memory until it is free.
- f. A parallel input/output controller is required to allow the supervisor processor to examine and control the target system interrupt, halt and rest lines.
- g. A facility is needed to demultiplex and multiplex the target bus as required.

- h. DTACK generation circuit responsible for generating DTACK signal to suit the access times of different target systems must be included.
- i. Address decoding logic to generate the required master and target request signals is necessary.

A schematic arrangement of the supportive and target systems is shown in Figure 6.5. Each type of target processor requires a unique personality module card (PMC) which plugs into the interface board. The PMC is responsible for any control signal transformation required by the particular target processor. The design and implementation of some personality module cards will be discussed in chapter 7.

A detailed block diagram of the EIB is shown in Figure 6.6, and the complete circuit diagram is given in Appendix E.

To allow for individual operations of each system and to prevent bus conflict, all data, address and control lines for both systems are buffered as they enter or leave the interface board. All the buffers are activated or deactivated as required by the accessing processor. The control signals that enable/disable data and address buffers (master side) is shown in Figure 6.7. The data direction buffers (master side) are controlled by the (R/\bar{W}) signal of the MC68000 processor. Figure 6.8 shows the control signals needed to drive the data buffers (target side).

6.2 Hardware design

6.2.1 Address decoding logic

The addressing capability of the supportive processor enables it to access any target memory or I/O location. For the purpose of this study, a free area of 128 k-bytes of the supportive memory map is chosen to handle the interface activities. This area corresponds to the hexadecimal addresses 860000_{16} to $87FFFF_{16}$.

Using DIL switches and the 2521 comparator, the interface board will respond when the selected master addresses are accessed. As shown in Figure 6.9, when any of the master addresses $86xxxx_{16}$ is decoded, a 'match' signal will be generated from the 2521 comparator which will enable the 74LS138 decoder.

The function of the decoded master addresses are as shown in the following table :

ADDRESS	FUNCTION
(860000–860xxx) ₁₆	Master Shared Memory Request (\overline{MSMR})
(862000–862xxx) ₁₆	Master Target I/O Request ($\overline{TI/O}$)
(864000–864xxx) ₁₆	Target Access latch (\overline{TACC})
(866000–866xxx) ₁₆	PIA Enable (\overline{PIAEN})
(868000–868xxx) ₁₆	Vector Latch (\overline{VECL})
(86A 000–86Axxx) ₁₆	Extra Byte Address Latch (\overline{EBAL})
(86C 000–86Cxxx) ₁₆	PIA interrupt input port A ($\overline{PIACA 1}$)
(86E 000–86Exxx) ₁₆	PIA interrupt input port B ($\overline{PIACB 1}$)
(870000–87FFFF) ₁₆	Master Target Memory Request (\overline{MTMR})

Table 6.1

– When the master requests direct memory access to the target memory, a Master Target Memory Request (\overline{MTMR}) signal is asserted. This signal is routed through the personality module card to assert the Target Bus Request (\overline{TBR}) signal. The Target Bus Grant (\overline{TBG}) signal will be asserted according to the target processor bus request cycle protocol.

Each target system has a reserved area in its memory map for shared memory accesses. As shown in Figure 6.10 two latches and two comparators are used to decode the target address lines (TA_{11} – TA_{23}) to generate the Target Shared Memory Request (\overline{TSMR}) signal.

The \overline{TSMR} signal is also routed to the PMC to immediately assert the target wait line signal \overline{TWAIT} irrespective of whether the master is using the shared memory or not. The \overline{TWAIT} signal will be active

for 500 nanoseconds in order to prevent the target from requesting another access to the shared memory before the previous request is arbitrated.

6.2.2 Arbitration logic

The common memory is accessible by the two processors on first come first serve basis. The arbitration circuitry will allow the higher priority request, MSMR or TSMR, to access the shared memory. Each shared memory access request made by either processor will be granted if the memory is not in use. If the shared memory is in use by one processor, the other processor requesting shared memory access is required to wait until the access by the other processor is complete. An arbitrated signal (\overline{MSMRA} or \overline{TSMRA}) is asserted for the processor permitted to use the shared memory. Since accessing of the shared memory has to take as little time as possible, the arbitration circuit is driven by high speed clock of 16 MHz. The arbitration circuitry is shown in Figure 6.11.

A separate circuit is used to generate the correct timing for the shared memory RAM enable and read/write signals.

6.2.3 DTACK generation circuit

When the supervisory processor addresses any valid memory location within the interface memory range, the \overline{DTACK} (Data Transfer ACKnowledge) signal is expected to be asserted within 50

microseconds, otherwise a bus error condition will be signalled to the processor. The \overline{DTACK} signal is used to allow the supervisory processor to be interfaced to slow memory devices. As shown in Figure 6.12, the length of the delay can be set using the DIP switch.

6.2.4 I/O controller

As the MC68000 processor is a hardware compatible with its predecessor the M6800 family, the M6821 Peripheral Interface Adapter (PIA) is used as the parallel input/output controller to monitor and control target interrupt, halt and reset lines.

The PIA device contains two 8-bit ports, port A and port B. Each of the 16 lines can be programmed to be input or output. Each port consists of three programmable internal registers, output register, data direction register and control register. On the supportive addressing range they appear as the low order bytes of two adjacent 16-bit words.

The master address lines A_{13} and A_{14} are connected to the PIA chip select lines. Two other address lines A_1 and A_2 are also connected to the PIA to select the internal registers.

The PIA is connected to the master interrupt daisy-chain circuitry. When either of the PIA interrupt request lines is asserted and the processor Interrupt Acknowledge IN (\overline{IAIN}) is active, the interrupt daisy chain state machine will generate a local \overline{IACK} signal which will assert the \overline{VPA} line (Valid Peripheral Address) as shown in Figure 6.13. Active \overline{VPA} line alerts the M68000 processor that a M6800 peripheral (PIA) requires its attention and that it must

synchronise it self with the clock signal (E). On enabling \overline{VMA} (Valid Memory Address), the M68000 addresses the PIA and indicates that it is ready to interact in synchronisation with clock E. M6800 peripherals in general do not generate vector numbers. The M68000, therefore, uses the autovector procedure which allows it to access the seven autovectors of the exception table.

The PIA port A is programmed to be an output port, and PIA port B as input port. Two of the PIA outputs are used on the main board and the rest are routed to the personality module and therefore have functions particular to the target processor being used. The two PIA outputs used on the main board are PA0, PA1. Output PA0 informs the main board whether the target is an eight bit or a sixteen bit processor. Output PA1 enable/disable target shared memory accesses.

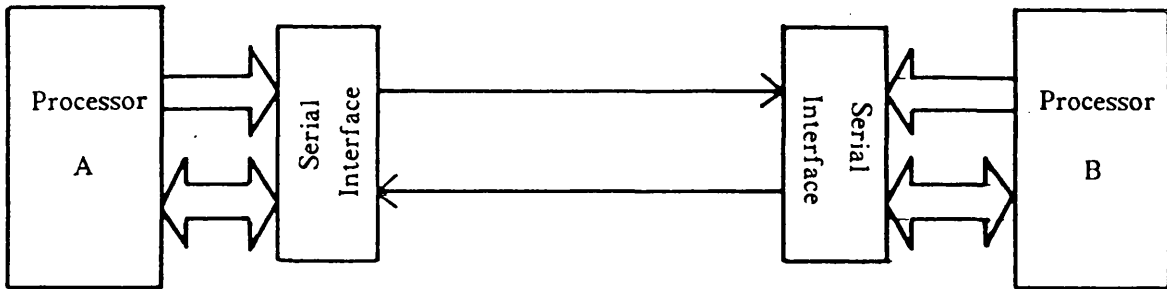


Figure 6.1 Connection of two Processors using Serial Interface

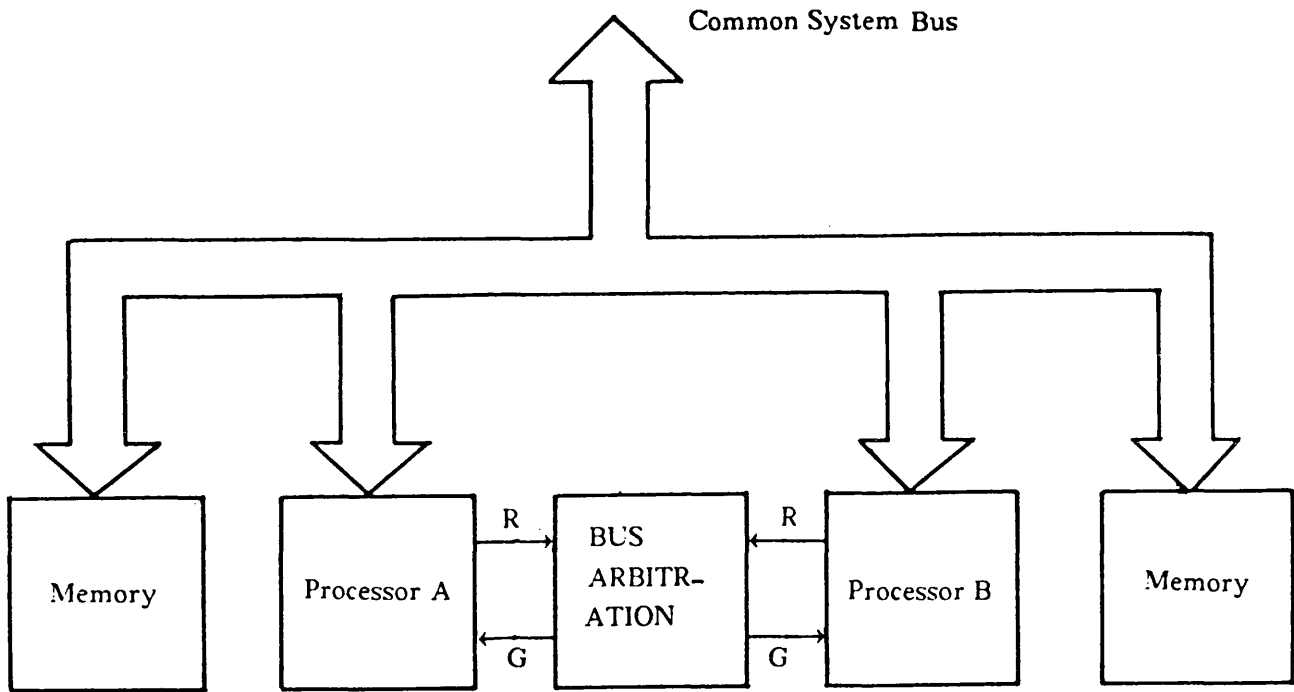


Figure 6.2 Connection of two Processors to Enable Access to Shared Resources

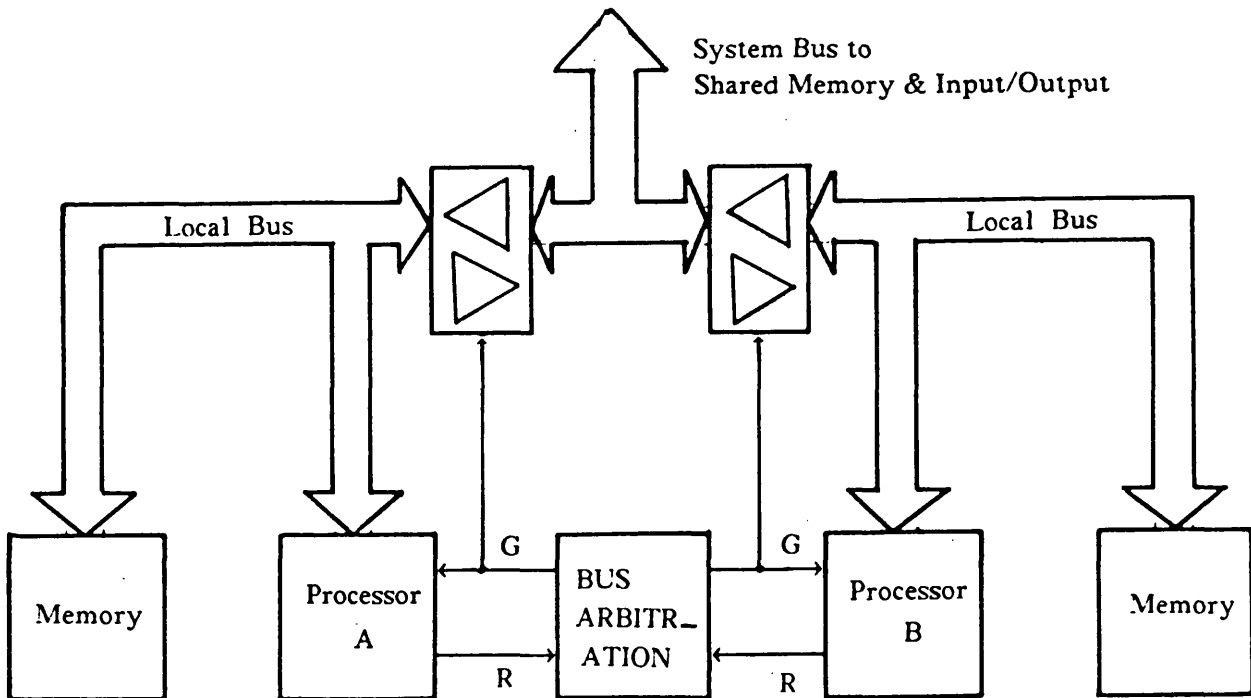


Figure 6.3 Connection of two Processors, each with Local Bus and Resources, to a Common Bus with Shared Resources

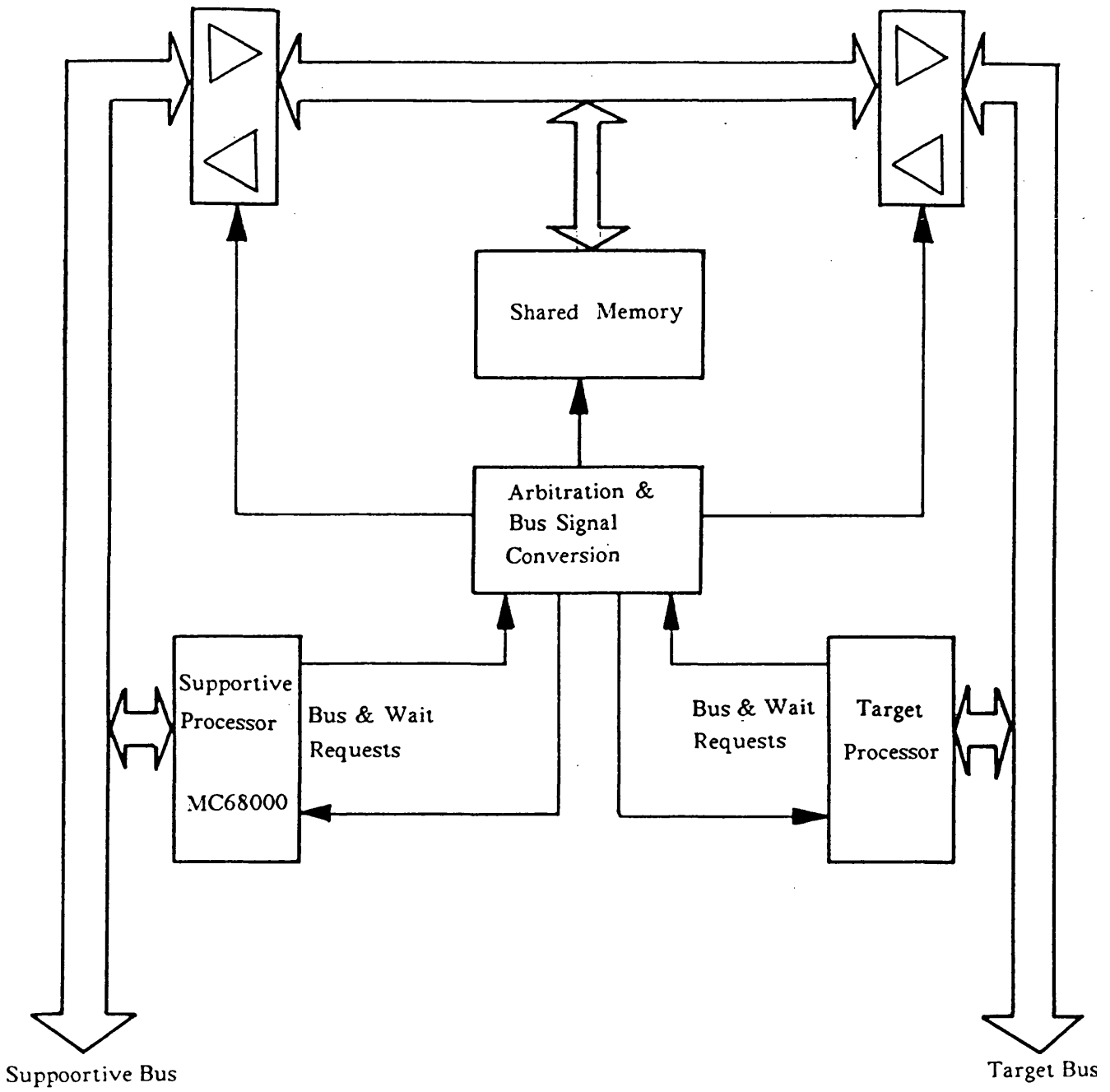


Figure 6.4 Adopted Connection Scheme of Supportive/Target Interface

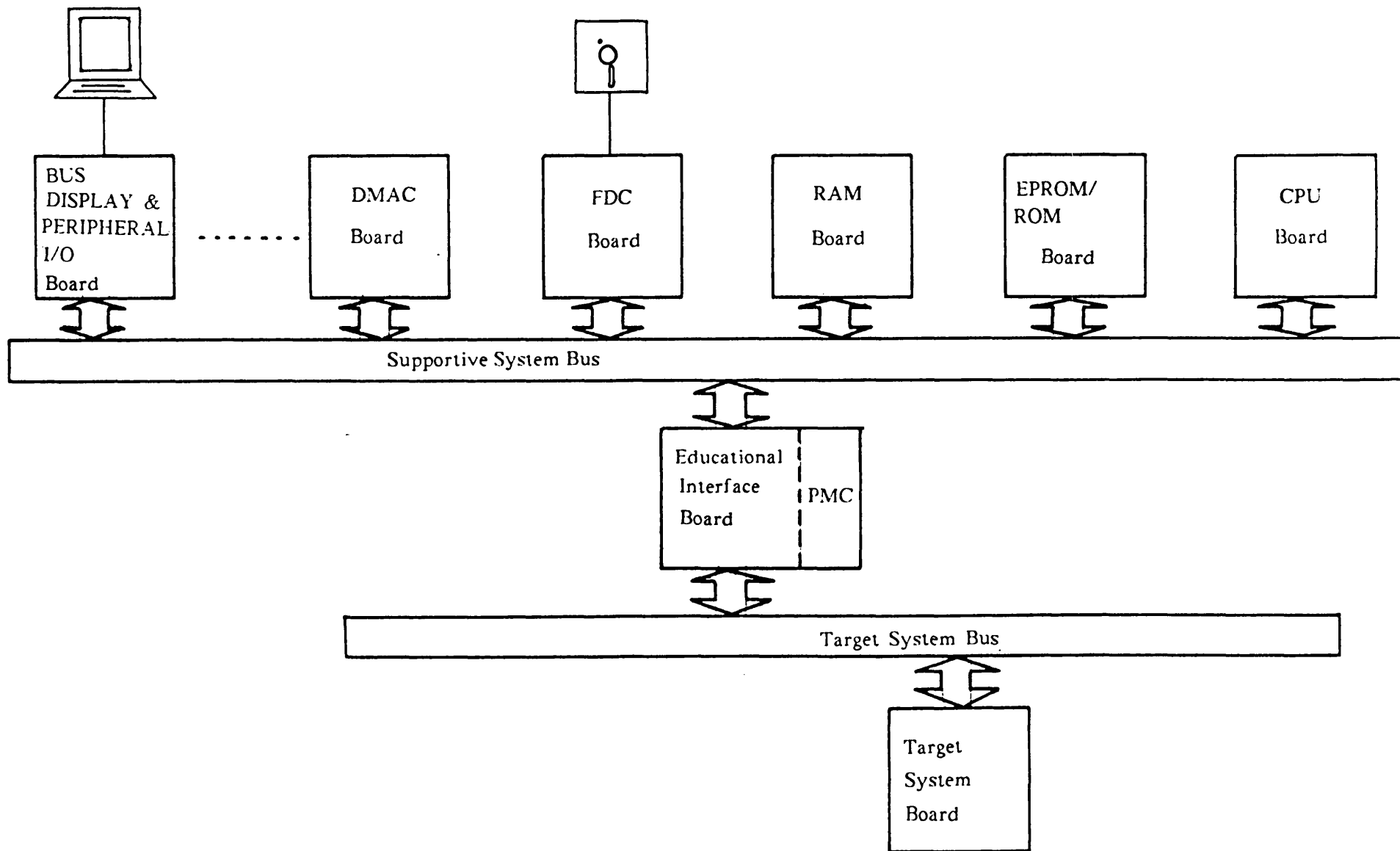


Figure 6.5 Supportive Target Interface Architecture

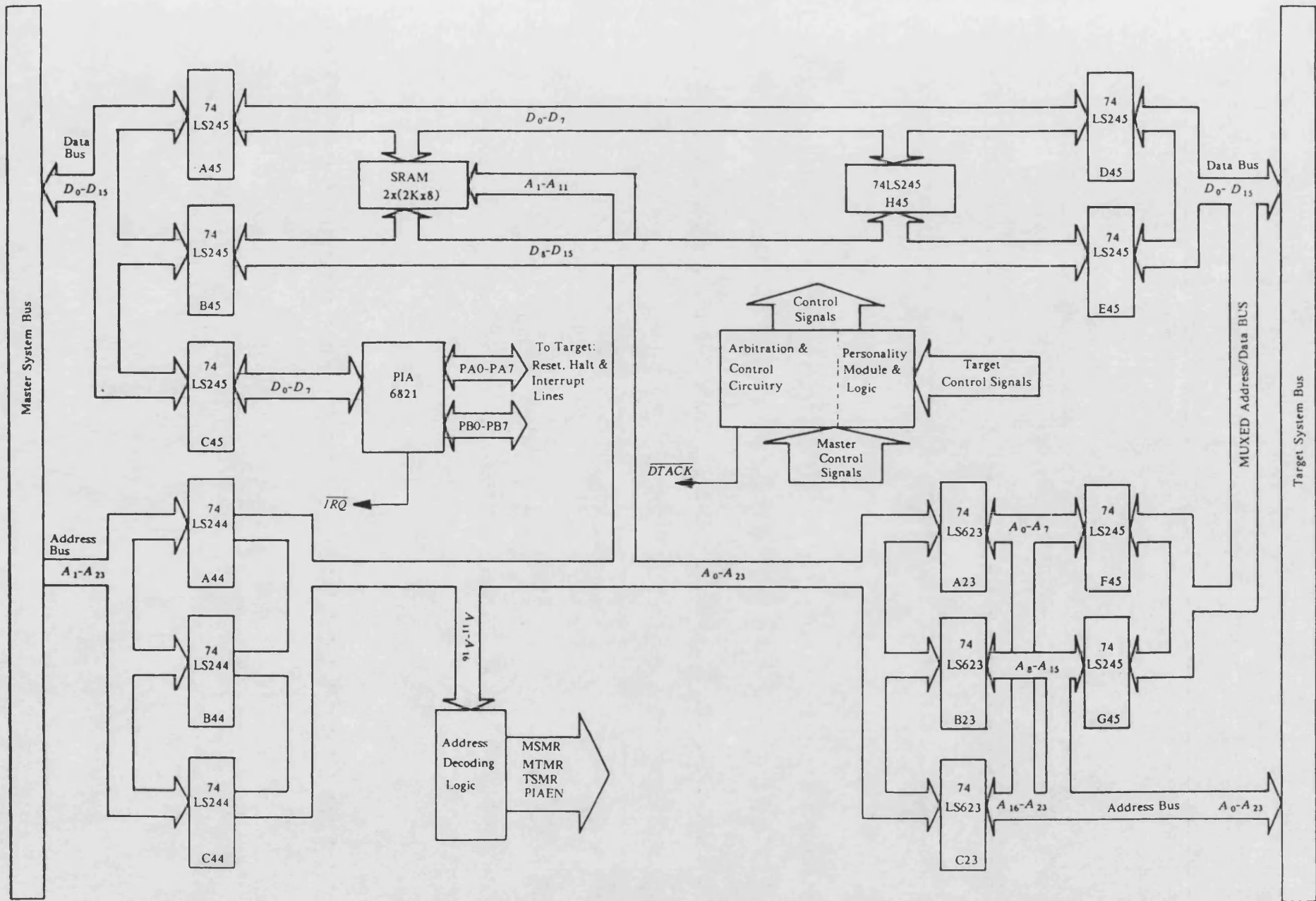


Figure 6.6 Educational Interface Board Architecture

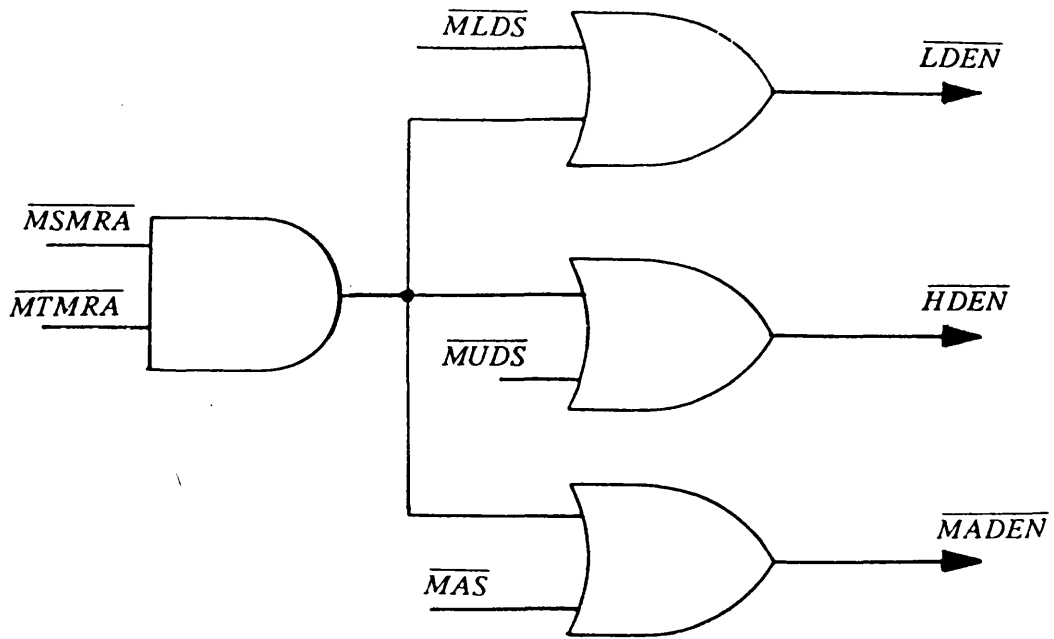


Figure 6.7 Logic Circuit to Generate Data and Address Enable Signals (Master side)

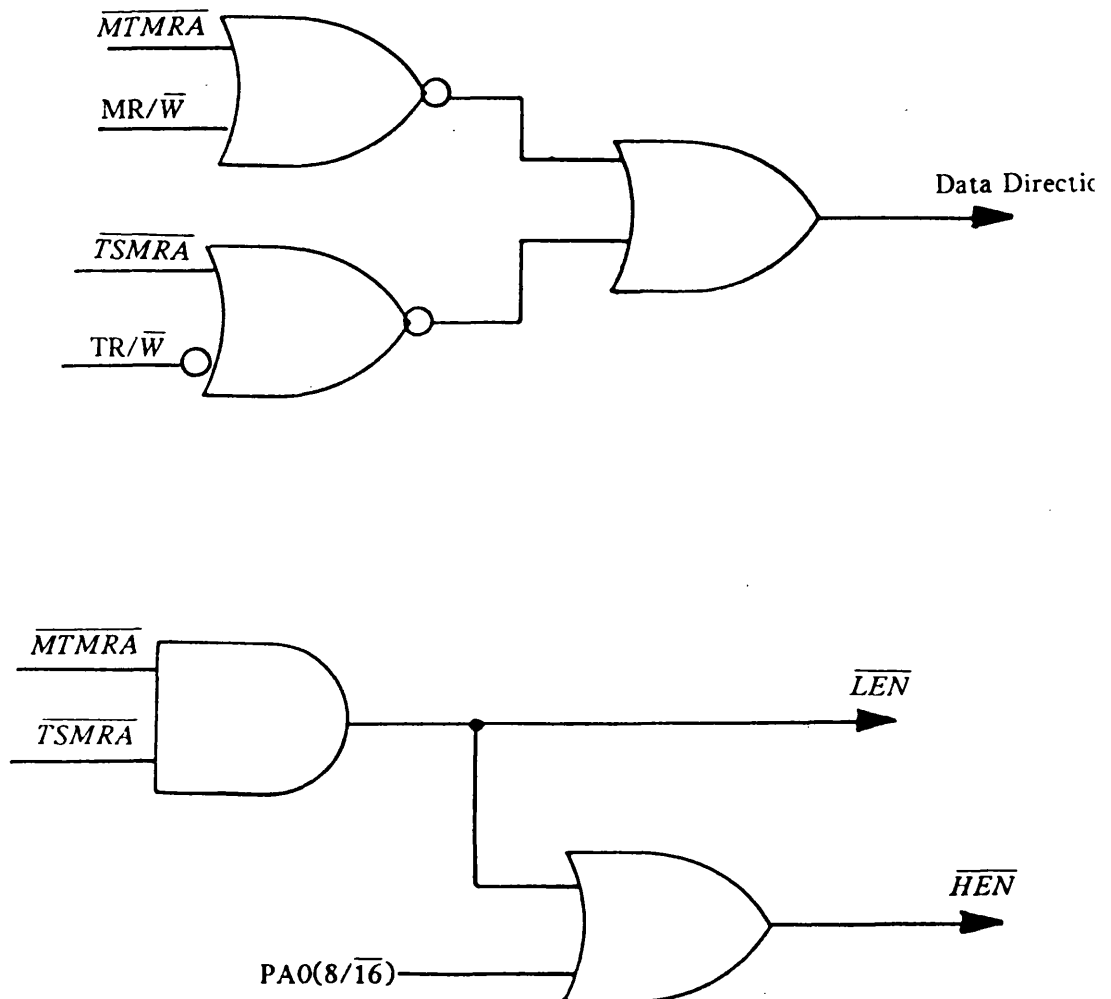


Figure 6.8 Data and Direction Enable (Target side)

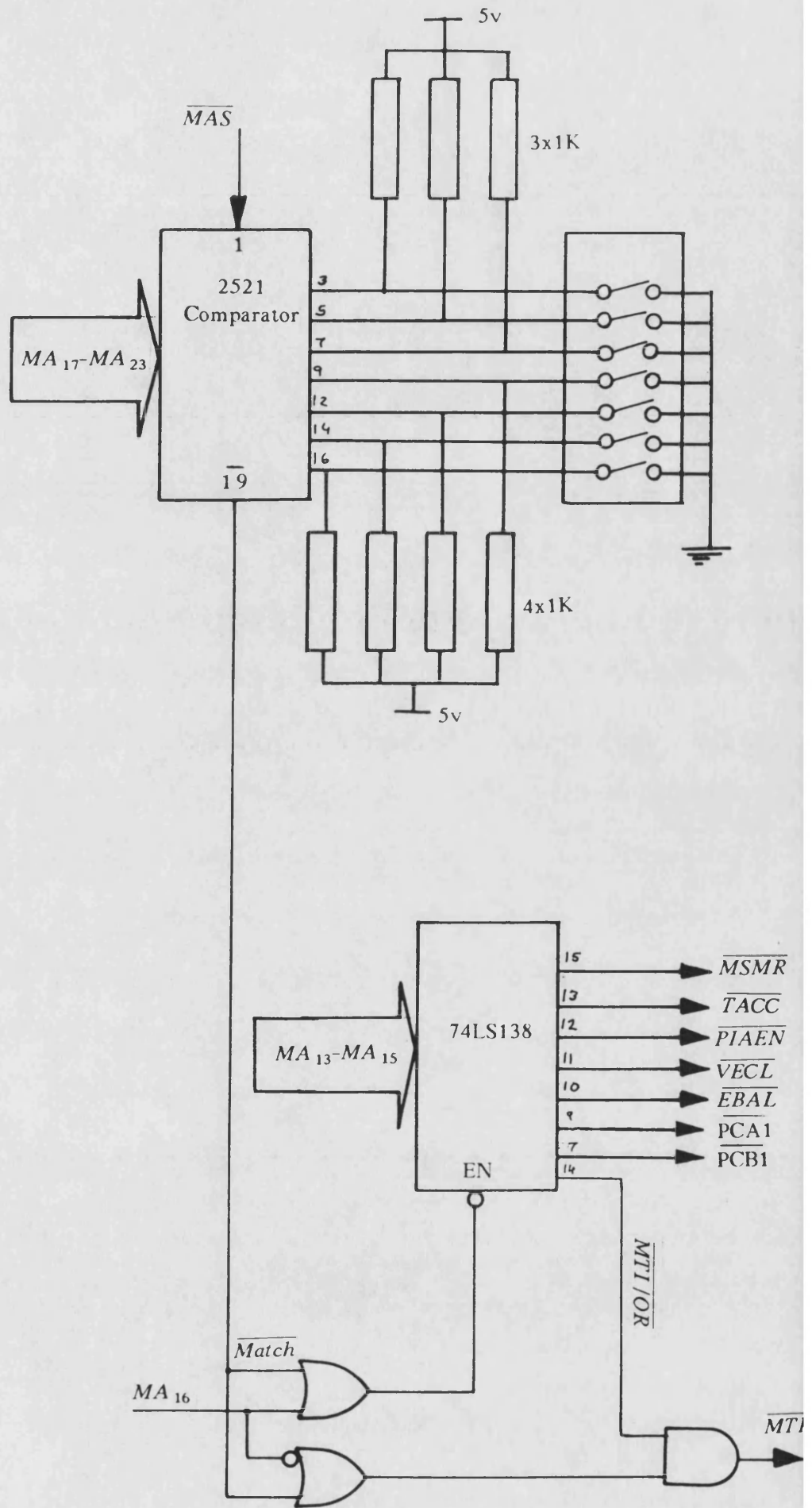


Figure 6.9 Address Decoding Circuit

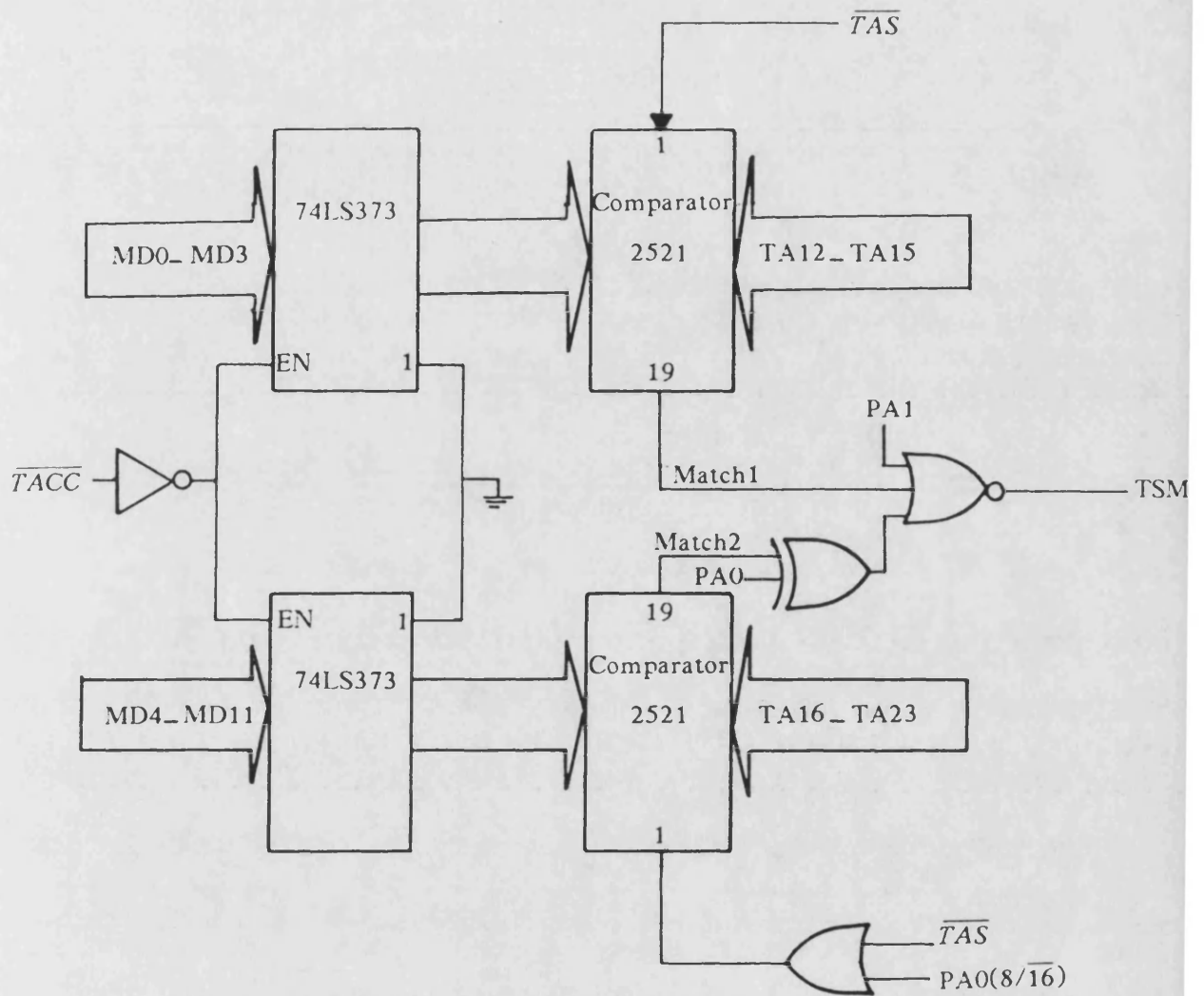


Figure 6.10 Target Shared Memory Request Circuit

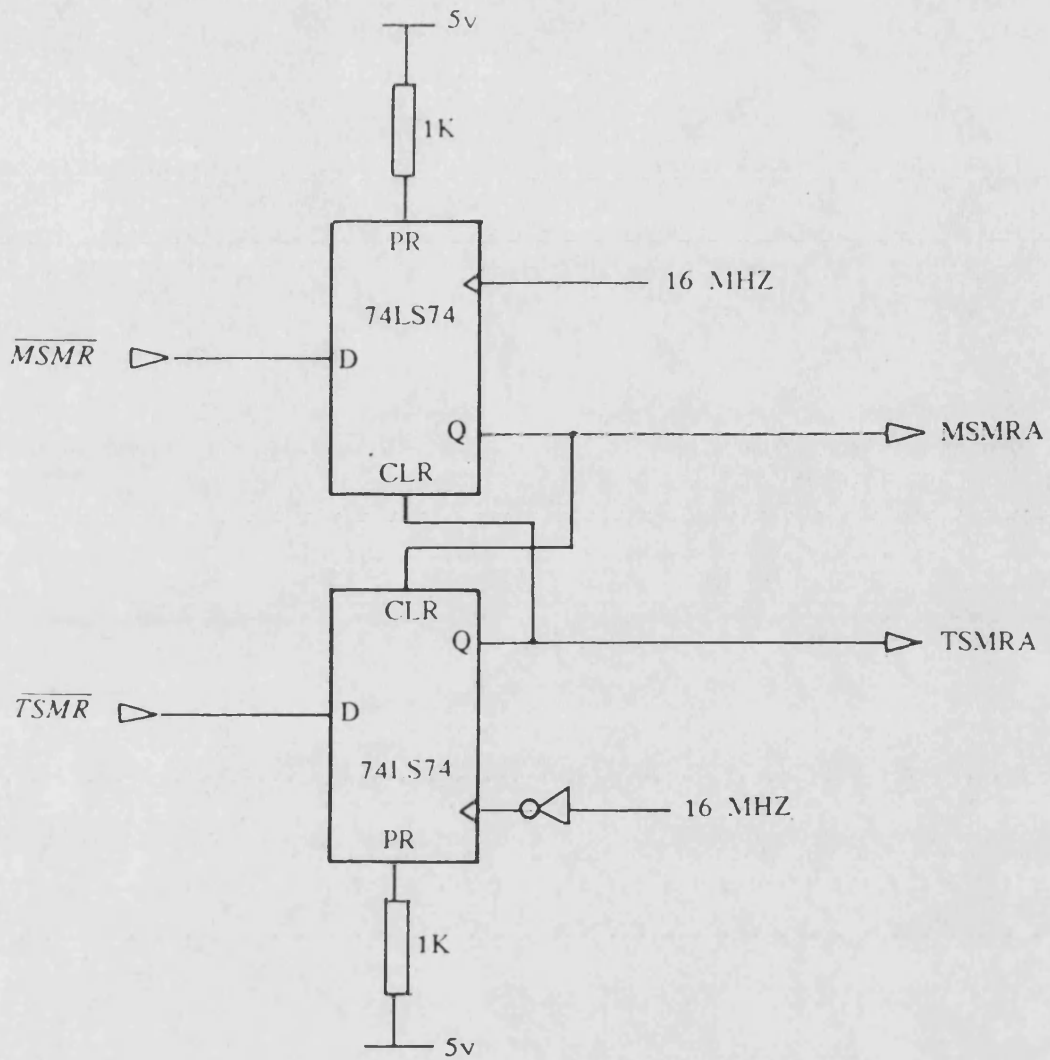


Figure 6.11 Master/Target Shared Memory Arbitration Circuit

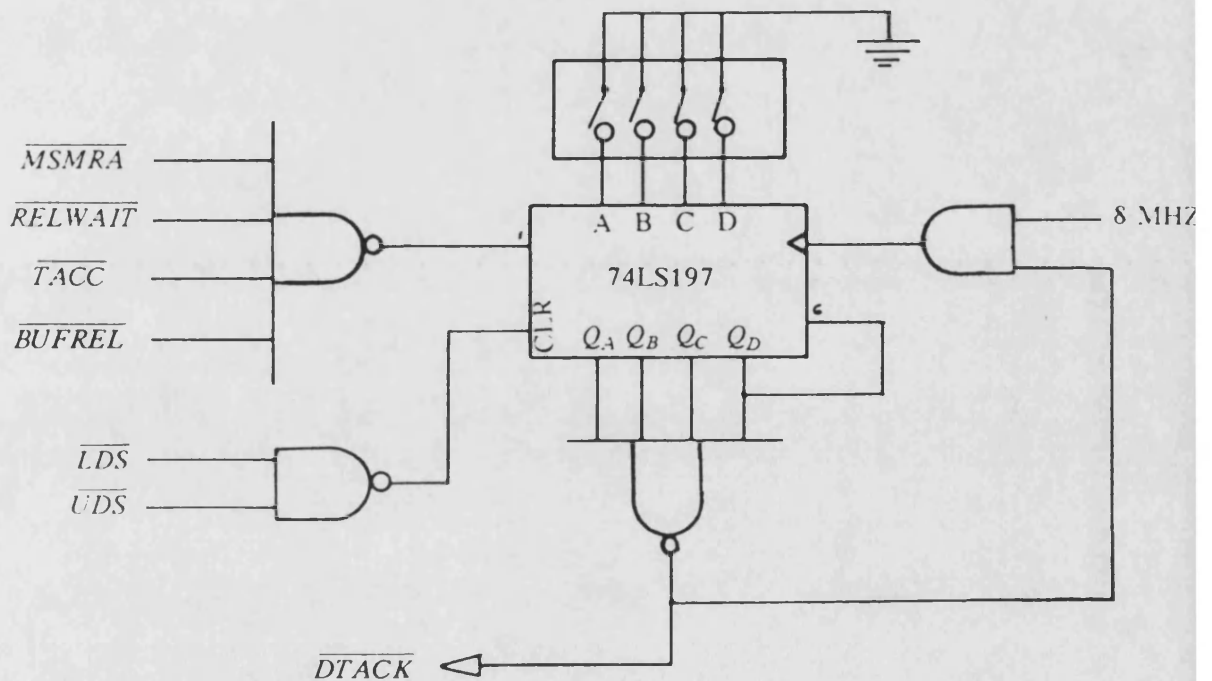


Figure 6.12 \overline{DTACK} Generation Circuit

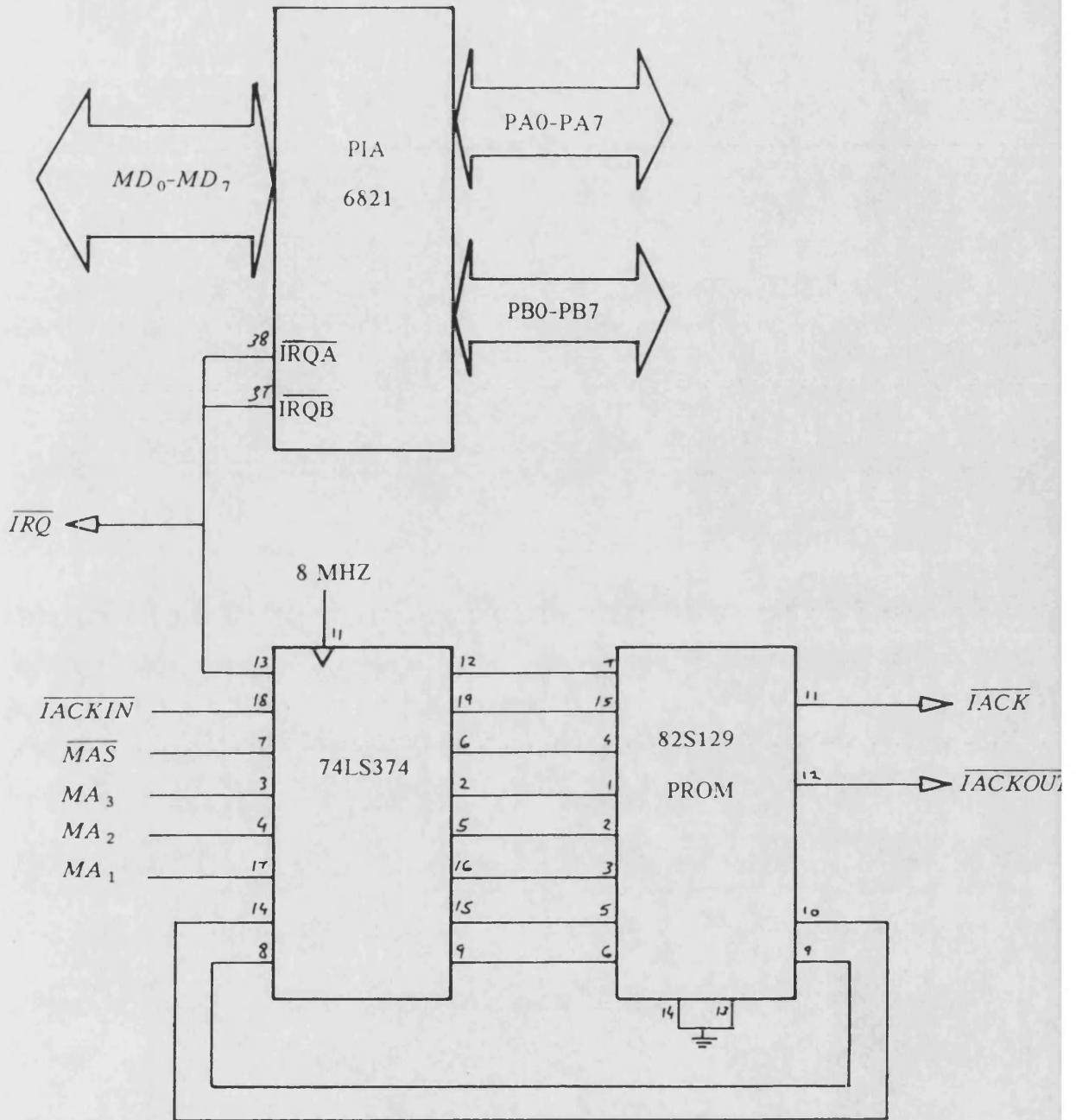


Figure 6.13 Interrupt Daisy Chain Logic

7. Target systems

7.1 Target system specification

As the complexity moves toward the supportive system, the target system should be as simple as possible with minimum facilities on board. The simple target system should be based on a CPU, memory and basic I/O structure. The function of the supportive machine is to assist in the interpretation and control of such systems.

The target systems should be provided with an EPROM facility in order for the target board to run programs in stand alone mode. Random access memory must be present at the reset vector space of the target microprocessor, and for stand alone operation the EPROM must occupy the vector space. Therefore, it is important that the board should provide a facility in order to be able to manipulate the reset vector between the two types of memory.

For demonstration purposes each target board should have parallel and serial I/O devices to enable the user to add his specific hardware application.

The target board should be divided into two distinct halves as shown in Figure 7.1. These are the CPU with its associated clock and buffers and the memory and I/O devices. Thus, the functionality of the board can be described as follows : The target CPU signals are to be buffered and applied to the J1 connector of the board. The J1 connector supplies all the signals required by the memory buffers and

decode logic circuitry. This means that the backplane always contains valid target signals irrespective of the device being accessed. As the target memory is always decoded from the J1 address bus, a direct memory access cycle has to acquire the backplane from the target processor and control the backplane as a normal target bus master. This separation between CPU and memory also allows the target memory or I/O to be completely disabled to enable the processor to work with user installed memory and I/O devices.

Two different target system boards are used in this work, one system is based on an eight bit microprocessor, the Z80, while the other board is based on a sixteen bit processor, the MC68000. The use of the two different target processors will demonstrate the versatility of the interface board. Two different personality module cards are designed to accompany the target boards.

7.2 The Z80 target system

The Z80 target board is a complete microprocessor system with the minimum basic devices on board. They include the CPU, memory and basic input/output device. When the target board is plugged into the supportive development system backplane together with the interface board, a new environment is created to allow the user to study and evaluate the target processor. Any user special hardware application can be added to work with the target system.

7.2.1 Circuit description

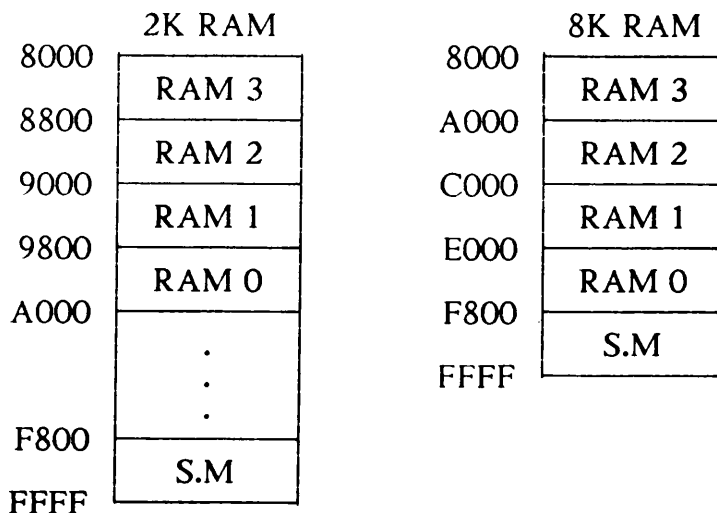
The Z80 target board contains a fully buffered Z80A processor running at 4 MHz clock. The clock generation circuit is built around a 16 MHz oscillator which is then divided down to 4 MHz signal.

As shown in Figure 7.2, the Z80 address and data lines leaving the processor are buffered, the address bus by the 74LS244 devices (ic 18,20), the data lines by the bi-directional buffer 75LS245 (ic22). These devices are enabled while the Z80 has control of the backplane, and disabled when a direct memory access operation has been requested and granted by the BUS ACKnowledge (\overline{BUSACK}) signal. The direction of the data buffer is controlled by the DBUF signal. This signal is activated (i.e data transfer toward the CPU) when ever the processor is performing a read operation or receiving an interrupt vector during an interrupt cycle. The processor control lines are buffered using the 74LS244 device. The buses are buffered on the edge of the board to protect the on board devices from noise induced on the backplane.

The memory decoding logic can access up to 32 Kbytes of static RAM and 32 Kbytes of EPROM. Referring also to Figure 7.2, the address data and control lines for both the memory and I/O device are buffered from the backplane (ic 17,19,21). The address and control buffers are continuously enabled while the data buffer is enabled by (EN) signal and the direction control is by the (DIR) signal. The EN signal is activated for all memory or I/O read and write cycles. The direction control signal DIR drives toward the

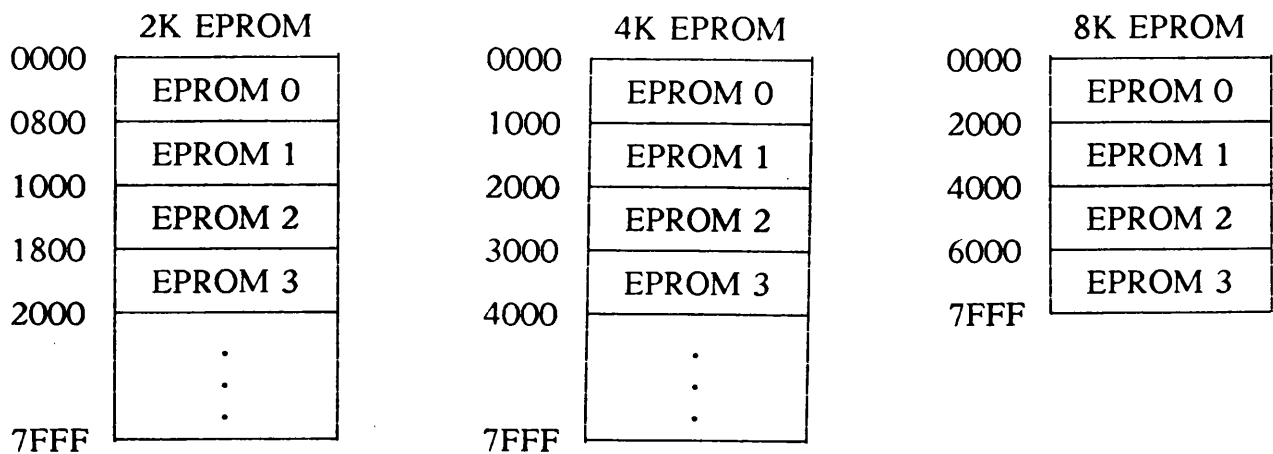
memory devices for all onboard memory write cycles, and toward the J1 connector (backplane) for read cycles from off board memory. The memory decoding logic circuit is shown in Figure 7.3. The decoding is achieved by the use of prom and 3 to 8 decoder. Address lines A_{11} - A_{15} and switches (S0-S2) are applied to the prom. The switches are to inform the prom of the size of memory device currently in use, while the address lines allow the prom to recognise 2K pages of memory. Three outputs signals from the prom are decoded by the 74LS138 to select the chip enable of the memory device requested. The fourth output signal (\overline{MEM}) generated from the prom is also fed to the decoder. This signal, when in a low state, indicates that an onboard memory is being accessed. The decode logic produces signals to enable four RAM and four EPROM devices. Using the RAM/EPROM swap switches (S3-S6) together with the chip select signals enable the RAM and EPROM devices to swap positions in the memory map in order to locate the reset vector address. The RAM sockets can support 2K or 8K devices (e.g 6116 and 6264), while the EPROM can support 2K, 4K and 8K devices (e.g 2716, 2732 and 2764).

As mentioned previously, the size of the memory device is defined by the switches S0-S2. The RAM maps are as follows :



An area of 2K bytes ($F800-FFFF$)₁₆ is reserved on the target memory map for shared memory activities.

The EPROM decode is achieved in a similar manner with the switches S0-S2 to select the size of the EPROM device. The memory maps of the EPROM are shown as follows :



7.2.1.1 The target I/O facility

The Z80 target board also contains two Z80 PIO devices, Dual-channel Asynchronous Receiver/Transmitter (DART) and two Z80 counter timers (CTC). One of the timers is committed to generate the baud rates for the DART serial conversion. A visual display output comprising of an array of eight LEDs is also provided on board the target card.

An area in the Z80 memory map is reserved to address the I/O devices, it is decoded as follows :

0090-0093	CTC1	the first counter timer chip
0094-0097	DART	the dual channel serial device
0098-009B	PIO1	the first parallel I/O device
009C-009F	PIO2	the second parallel I/O device
00A0-00A3	CTC2	the second counter timer chip
00A4-00A7	LEDs	these four locations provide write only access to the 8 LEDs on the board edge.
00A8-00AB	DIL	these four locations provide read only access to the 8 way DIL switch on the board edge.

The Z80 input/output devices are connected together using daisy chain interrupt priority system. The daisy chain is then routed to the backplane to allow other devices to be connected. The input/output lines generated by the two Z80 PIOs are applied to the bottom connector (J2) of the double eurocard to enable external devices to be controlled if any external experiment is to take place. AS one of the counter timers is used to provide baud rates for the DART device, the second counter timer is applied to the J2 connector. The control and data lines of the Z80 DART are translated to RS232 levels and passed to J2 connector. The pinout of J1 connector is given in Appendix B. If a terminal is to be connected to the Z80 DART via J2, then it can be

used to examine and alter the memory and registers of the Z80 target. This action is only possible under the control of a monitor program that can reside in one of the EPROM sockets, and if desired the target system can run completely independent of the supervisory system.

The open collector lines, interrupt request, non-maskable interrupt, bus request and halt are provided with pull up resistors and are available on the backplane and hence can be monitored by the supervisory processor via the interface card. The interrupt request line (\overline{INT}) is taken to all the target I/O devices so that these devices may interrupt the processor if required.

A power up reset facility is provided on board the target system by the use of the 555 timer. The reset signal generated is taken to the backplane reset signal via an open collector buffer. The reset signal is buffered from the backplane and applied to the CPU, DART and CTC reset lines. This signal is also applied to a red LED, via open collector buffer which, when illuminated indicates that the Z80 target system is in a reset state. The board can be reset manually using a toggle switch which when set, resets the 555 timer. If the switch is permanently positioned toward reset state, then the supportive system would be unable to gain control of the target system. Similarly, the halt line is taken to another red LED which, when illuminated, indicates that that the processor is in halt state.

7.3 The Z80 personality module card

As the educational interface board is designed to be universal to

easily adapt to any microprocessor system, a dedicated Personality Module Card (PMC) for each type of target processor is, therefore, required. The PMC is small in size and can be plugged on top of the interface board. Each PMC is responsible for the control signal conversion required by the target processor in order to establish communication between the two systems. The complete circuit diagram of the Z80 PMC is shown in Figure 7.9. The circuit description of the PMC will be included in the discussion of the Z80 target interface section.

7.4 The Z80 target interface

The data bus of the master processor, the MC68000, is sixteen bit wide, while the Z80 target is eight bit wide. The master processor is capable of communicating using either the entire data bus to transfer words of data or using upper or lower data bus for byte transfer.

As shown in the general layout of the interface board Figure 6.6, the function of the bi-directional buffer (ic H45) is to allow the master processor to communicate with eight bit target systems using both upper and lower data bus. This buffer will also enable the eight bit target processors to access the high order byte of the shared memory. The control circuit to drive this buffer is shown in Figure 7.4. The target address line A0 is passed to the interface board and used as the target upper byte request signal (\overline{TUBR}). When inverted it is used as the target lower byte request (TLBR).

All the major Z80 control signals are buffered as they enter the personality card. The extra address ($TA_{16}-TA_{23}$) and data (TD_8-TD_{15}) lines which are not used on board the interface card, in the case of an eight bit target processor, are set to low state on board the personality card.

7.4.1 Master to Z80 target memory access

The master processor can request direct memory access to the target system by accessing any of the target memory request hex addresses ($870000-87FFFF$)₁₆. When any of these addresses is accessed, a master Target Memory Request (\overline{MTMR}) signal is generated by the interface address decoding logic. This signal is passed to the personality card and applied to the circuit shown in Figure 7.5a to generate the Z80 Target BUS REQuest signal ($\overline{TBUSREQ}$). The Z80 bus request signal has a high priority level and is always recognised by the processor at the end of the current machine cycle. When the processor detects that \overline{BUSREQ} is active, it forces the address bus, data bus and the control signals \overline{MREQ} , \overline{IORQ} , \overline{RD} and \overline{WR} into a high impedance state so that the supervisor processor can take control of the buses to start direct memory access operation.

The master processor is informed of the target bus mastership by the assertion of the Z80 BUS ACKnowledge (\overline{BUSACK}) signal. The BUSACK signal is passed to the interface board as Master Target Memory Request Arbitrated (\overline{MTMRA}) signal. This signal is used to enable/disable the data and address buffers on the interface board. It is also applied to a D-type flip flop to generate a $\overline{RELWAIT}$ within 500

nanoseconds. The $\overline{RELWAIT}$ signal is taken to the interface board to assert the \overline{DTACK} signal in order to terminate the master target memory access cycle. The delay generated by the flip flop is to ensure that the master does not try for another target access before the previous target access cycle is completed. It also allows enough time for the data bus to be stable.

When the master gains control over the target bus, it starts a normal Z80 read or write cycle, as requested, to the target memory by issuing target Memory REQuest (\overline{MREQ}) signal and the appropriate target read or write signal.

If a target I/O address is decoded on board the interface card, the Master Target I/O Request (\overline{MTIOR}) signal will be asserted together with the \overline{MTR} line. As the master gains control over the target buses as in the target memory access, it begins a Z80 I/O read or write cycle by asserting the target (\overline{IORQ}) line and the appropriate target read or write signal. The logic circuit to generate the major target control signals, \overline{MREQ} , \overline{IORQ} , \overline{RD} and \overline{WR} is shown in Figure 7.6.

In the master to a target access write cycle, the personality card must terminate the target cycle before it terminates the master cycle to ensure that the valid data is latched to the target memory before the master relinquishes the data bus. When in the master to target read cycle, the personality card is responsible for terminating the master cycle before it terminates the target cycle to ensure that the master has captured the correct data. Read/write cycle timing diagram is

shown in Figure 7.5b.

7.4.2 The Z80 target to shared memory access

As the Zilog Z80 processor accesses any of the shared memory addresses, a Target Shared Memory Request (\overline{TSMR}) signal is generated on board the interface card. The \overline{TSMR} and \overline{MSMR} signals are fed to the arbitration circuit to decide which system will have access to the shared memory. If the target request has a higher priority level than the master, an arbitrated \overline{TSMR} signal (\overline{TSMRA}) is generated. The \overline{TSMRA} signal together with the appropriate target Read/Write signal are used on board the interface card to select the required shared RAM. The arbitrated signal, \overline{TSMRA} , is also used to enable data and address buffers (target side).

The \overline{TSMR} and the \overline{TSMRA} signals are routed to the Z80 personality module card and applied to the circuit shown in Figure 7.7. When either or both of the signals are asserted, Z80 wait states are inserted for a period of 500 nanoseconds to ensure that the target processor does not request another shared memory access before the previous cycle is completed.

7.4.3 The Z80 target interrupts

The Z80 personality module card allows direct master control of target \overline{INT} , \overline{NMI} , \overline{BUSREQ} and \overline{RESET} lines using the interface PIA. These lines are activated by the master programming the PIA as shown in the following table :

PIA A7	PIA A6	PIA A5	Function
0	0	0	Homestate
0	0	1	\overline{INT} immediate
0	1	0	\overline{NMI} "
0	1	1	\overline{RESET} "
1	0	0	\overline{BUSREQ}

TABLE 7.1

The output lines PIA A3 and PIA A4 are programmed to enable or disable any possible target \overline{INT} or \overline{NMI} requested by the supportive system. The input lines (PIA B0-PIA B5) are programmed to monitor the following target signals:

PIA B0 The \overline{INT} Line
 PIA B1 " \overline{NMI} "
 PIA B2 " \overline{RESET} "
 PIA B3 " \overline{BUSACK} "
 PIA B4 " \overline{BUSREQ} "
 PIA B5 " \overline{HALT} "

In mode 2 of the Z80 maskable interrupt, the interrupting device supplies the starting address of the interrupt service routine by placing an eight bit vector on the data bus during the interrupt acknowledge (\overline{IACK}) cycle. Figure 7.8 shows the circuit used to supply the vector number during \overline{IACK} cycle.

7.5 The MC68000 target system

The 68000 target board is based on the MC68000L8 CPU running at 8 MHz clock. Similar to the Z80 target board, the overall structure of the M68000 target board is divided into two halves, the processor with its clock and buffers, and the memory and input/output devices.

The functionality of the M68000 target board is also similar to that of the Z80. The M68000 processor signals are buffered and applied to the J1 connector, and memory buffers and decode logic receive all the relevant information from the J1 connector. This makes the backplane always carries valid M68000 target signals.

The board carries two M68230 PI/T (Programmable Interface and Timer) devices which supply 40 lines of parallel I/O and two counter timers along with two M6850 serial ACIAs, fully buffered to RS232 levels, with baud rate clocks available from MC14411 baud rate generator. A simple I/O structure of 16 LEDs and 16 DIL switches complete the target peripheral facilities.

As the target processor is reset, it fetches eight bytes of data from memory locations 0-7 and uses these locations to load the program counter and stack pointer. In total, the first Kilobyte of memory locations is reserved for 255 vectors, each of thirty-two bits.

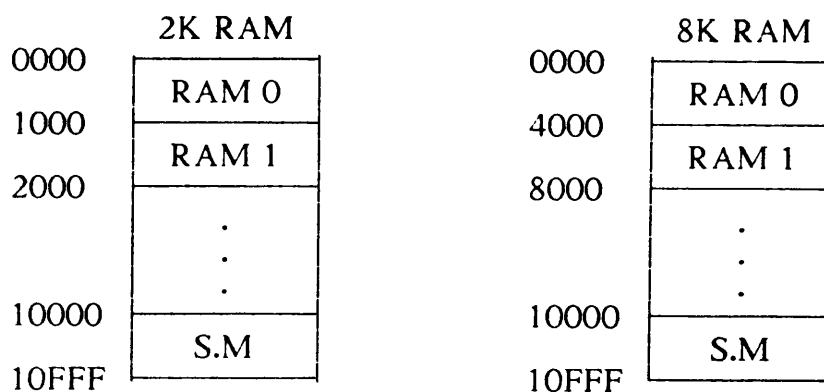
Buffering of the target buses is provided in a similar way to that described for the Zilog Z80 target system. The twenty three address lines are buffered as they leave the processor by three 74LS244

devices. The data bus is buffered by two 74LS245 devices. The address and data buffer devices are enabled while the M68000 target has control of the backplane and they are disabled when the target passes control to another bus master by asserting Bus Grant ACKnowledge (\overline{BGACK}) signal. The R/ \overline{W} target line is used to select the appropriate direction of the data buffer.

7.5.1 Memory maps and manipulation

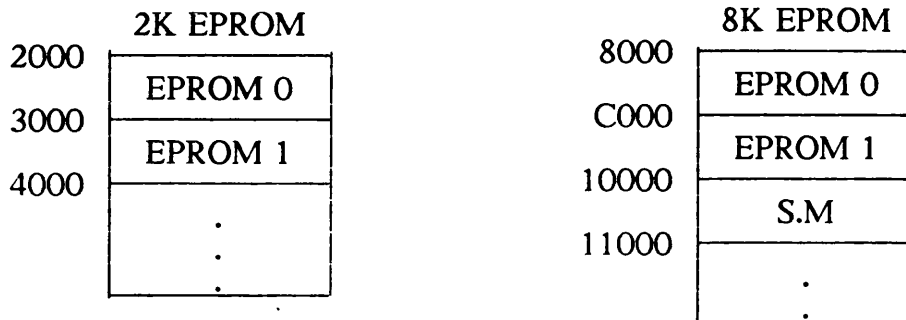
The memory decode logic provides chip enable signals for four RAM and four EPROM devices. The RAM sockets can support 2K or 8K devices (e.g. 6116 and 6264). The EPROM can also support 2K and 8K devices (e.g. 2716 and 2764). The memory maps for the EPROM and RAM area changes depending on the size of the installed devices. The size of the memory devices are defined by two switches S18 and S19 on board the target card.

The RAM maps are as follows :



An area of 4K bytes $(10000-10FFF)_{16}$ is reserved on the target memory map for shared memory activities.

The EPROM decode works in a similar manner, with the switch S19 to define the size of the EPROM device. The EPROM memory maps are as follows :



The reset vector and program can be manipulated to allow the target system to develop M68000 based applications. For the target to operate in a stand alone mode, the EPROM devices must occupy the program and vector space. Using switches S20 and S21 the facility of swapping the RAM and EPROM devices can be achieved.

Memory decoding as shown in Figure 7.10 is achieved by a 14L4 PAL (ic15), two 74LS138 decoders (ics 16,17) and a 10L8 PAL (ic18). Three output signals from ic15 are used to enable and select memory devices, a fourth enables I/O access. Address lines $A_{12}-A_{23}$ and switches S18 and S19 are the inputs to the PAL ic15. The \overline{MEM} signal from ic15 is the global memory enable line whilst outputs A and B form an encoded addressing inputs depending on the size selection switches S18 (for RAM devices) and S19 (for EPROM devices). The encoded chip select signals are decoded by ics 16,17 and

qualified by address strobe (\overline{AS}) signal. Upper and lower byte devices are selected with \overline{UDS} and \overline{LDS} lines respectively. The PAL equations used for the memory and I/O decode is given in Appendix D.

The I/O devices on the M68000 target board are memory mapped so they can be placed as desired. AS shown in Figure 7.10, the I/O devices are enabled by ic19 and a 74LS138 decoder ic20. The inputs to ic19 consists of a global I/O enable signal from ic15, address lines A_2-A_{11} and \overline{AS} . Three outputs A,B,C are applied to a 74LS138 decoder (ic20) to select the relevant I/O device. Three signals $\overline{M6800}$, $\overline{I/OPAGE}$ and $\overline{VPADRIVE}$ are generated by ic19. $\overline{M6800}$ and $\overline{I/OPAGE}$ are I/O enable lines for external I/O on separate boards available at the J2 connector. $\overline{M6800}$ is for M6800 type peripherals and $\overline{I/OPAGE}$ is for M68000 type peripherals. The output $\overline{VPADRIVE}$ is used to assert \overline{VPA} for all M6800 devices. This forces the M68000 target into a pseudo M6800 synchronous cycle and thus allows the interfacing of M6800 peripherals.

As stated previously, the one important difference between the M68000 and many other microprocessors is its ability to carry out asynchronous data transfer between itself and memory or peripheral devices. The asynchronous data transfer between the processor and other devices is controlled by five signals, \overline{AS} , \overline{UDS} , \overline{LDS} , R/\overline{W} and \overline{DTACK} .

The \overline{DTACK} generation circuit is shown in Figure 7.11. It consists of shift register ic25 (74LS273) driven by the 8 MHz clock

and enabled by the assertion of the \overline{UDS} and/or \overline{LDS} driving an open collector device ic52 (7403) qualified by an 'on-board' memory access signal \overline{MEM} via Link array LK21. This circuit allows the setting of \overline{DTACK} for 125 nanoseconds memory access time incremented by 125 ns to 1000 ns using LK21 in order to adapt to most situations. If the \overline{DTACK} signal is not asserted during a cycle, then the processor will theoretically wait indefinitely. Provision has been made to drive the processor into exception processing by asserting the Bus ERRor (\overline{BERR}) line some time after the \overline{DTACK} signal was expected. This will then allow the operation of the M68000 to be recovered.

The target interrupt structure possesses 192 usable vectors for peripherals that can provide a vector number, such as the MC68000 I/O devices, and seven autovectors allocated for devices that do not generate a vector number, such as the M6800 peripheral devices.

The hardware interrupts used by the on-board I/O devices are encoded by ic23 of Figure 7.12a, into three lines IPL 0,1,2 required by the target processor. There are seven levels of interrupts, six maskable interrupts (levels 1 to 6) and one non-maskable (level 7). Using the link array ic24, the interrupt outputs from the peripheral devices and backplane lines J2C2 and J2C3 can be set as required. The J1C2 is connected for level 7 to ensure that the master interface has priority over other interrupts. To ensure that devices requiring auto-vectoring addresses A_1 - A_3 and \overline{IACK} line are decoded by ic21, Figure 7.12b, and passed to link array ic22. During an IACK cycle these three address lines are coded with the interrupt level number that is

being processed. If the \overline{VPA} line of the processor is asserted at this time, then auto-vectoring begins. The pinout of J1 connector is given in Appendix C.

7.6 The M68000 Personality Module Card

As both the supportive and target processors are identical, the interface between the two systems is straightforward and of low complexity. The personality card is mainly consists of three state machines, one for initiating the master to target bus request, the second for initiating the target to shared memory read/write access and the third is responsible for generating the master to target interrupt, halt and reset signals. The complete circuit diagram of the M68000 PMC is shown in Figure 7.16.

7.7 Master to MC68000 target memory access

The master processor uses the same procedure to generate the master to target request signal (\overline{MTR}) as described for the Z80 target system.

The target memory request addresses $(870000-87FFFF)_{16}$ access 64 Kilobytes of memory, which is sufficient for eight bit target processors. For sixteen bit processors, such as the M68000 target processor, a latch (74LS374) is used on board the PMC to supply the highest address byte in order for the master system to be able to access the entire memory range of the target processor.

The generated \overline{MTR} signal, on board the interface card, is routed to the M68000 PMC and applied as one of the inputs to the state machine shown in Figure 7.13a. The \overline{MTR} signal forces the target Bus Request (\overline{BR}) line to be active. The M68000 target is at a lower bus priority level than the master, and will relinquish the bus after it has completed its current bus cycle. The target informs all other potential bus mastership devices that it will release bus control at the end of the current bus cycle by asserting the Bus Grant (\overline{BG}) signal. The target \overline{BG} , \overline{AS} , \overline{DTACK} and \overline{BGACK} signals are also applied as inputs to the state machine shown in Figure 7.13a. As soon as the \overline{TBG} is asserted and the \overline{TAS} , \overline{TDTACK} and \overline{TBGACK} are negated, the supportive processor takes over the control of the target bus by asserting the \overline{TBGACK} signal. The asserted \overline{TBGACK} signal is routed back to the interface board to enable the data and address buffers. This signal is also used in the master \overline{DTACK} generation circuit to terminate a master to target memory cycle by asserting the master \overline{DTACK} signal after the specified time of the generation circuit. This, in turn, will terminate bus mastership by the negation of the target \overline{BGACK} signal. Figure 7.13b shows the bus request cycle timing diagram.

When the master gains control over the target bus, it starts a normal M68000 read or write cycle, as requested by the master, to the target memory by issuing the appropriate control signals R/\overline{W} , \overline{AS} , \overline{LDS} and \overline{UDS} .

The active \overline{TBGACK} signal is used on board the PMC to enable

unidirectional buffer to pass master R/\overline{W} , \overline{LDS} , \overline{UDS} , \overline{AS} and \overline{DTACK} signals to their equivalent target lines.

7.8 M68000 target to shared memory access

The target system can request access to the shared memory by accessing any of the reserved S.M addresses. The target S.M address is compared, on board the interface board, with the previously latched S.M address. If it matches, a \overline{TSMR} signal is generated. The TSMR signal is arbitrated with MSMR line. If the target request is at a higher priority level than the master request, then an arbitrated target request signal (\overline{TSMRA}) is generated. The target arbitrated signal is used to enable the address buffers and used together with target \overline{LDS} and \overline{UDS} lines to activate the data buffers. It is also used to enable the shared memory during target read/write cycles.

The \overline{TSMR} , \overline{TSMRA} and TR/\overline{W} signals are applied to the circuit shown in Figure 7.14a to generate \overline{TWAIT} signal which is used to wait the target processor for approximately 200 nanoseconds before asserting target DTACK to signal to the termination of the target cycle. The delay is to ensure that the target system does not request another S.M access before the previous cycle is terminated. The target to shared memory read/write cycle timing diagram is shown in Figure 7.14b.

7.9 Target Interrupts

Similar to the Z80 PMC, the M68000 PMC provides direct master control and monitoring of the target interrupts, halt and reset lines by programming the PIA interface device. The PIA A5,A6 and target \overline{INTACK} signal are applied to the circuit shown in Figure 7.15a to generate target interrupt, halt, reset and vector latch signals. The vector latch signal is used, during an interrupt acknowledge cycle, to release the interrupting device supplied vector to the target processor. The timing diagram of this circuit is shown in Figure 7.15b. Port B of the PIA is programmed as inputs to monitor the target \overline{INT} , \overline{HALT} and \overline{RESET} lines.

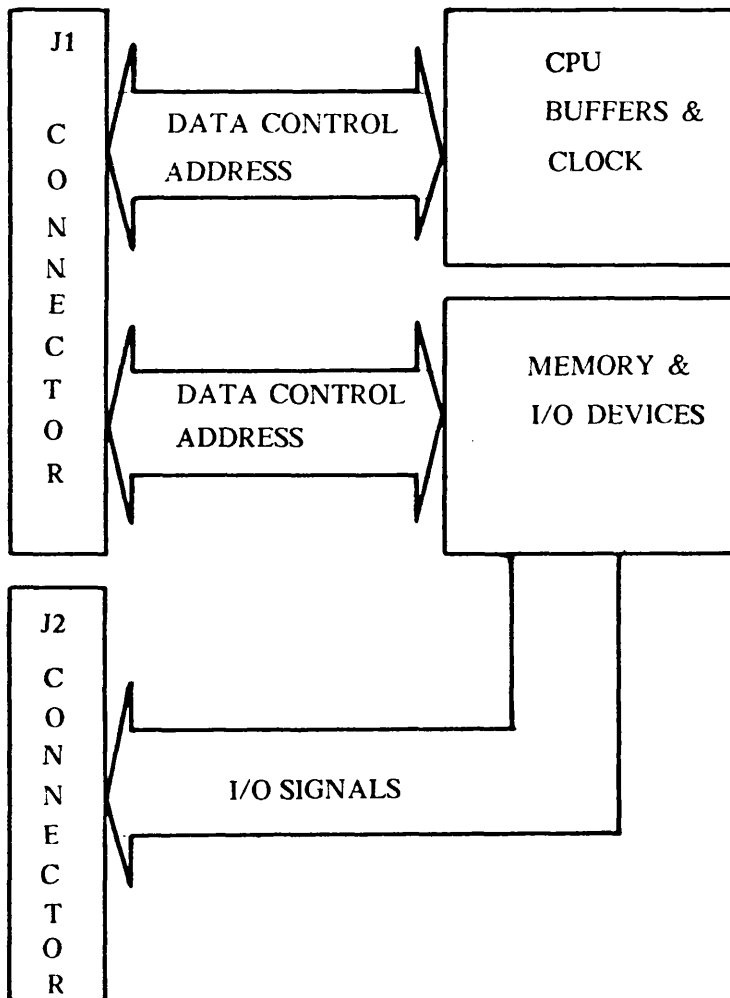


Figure 7.1 Target Board Architecture

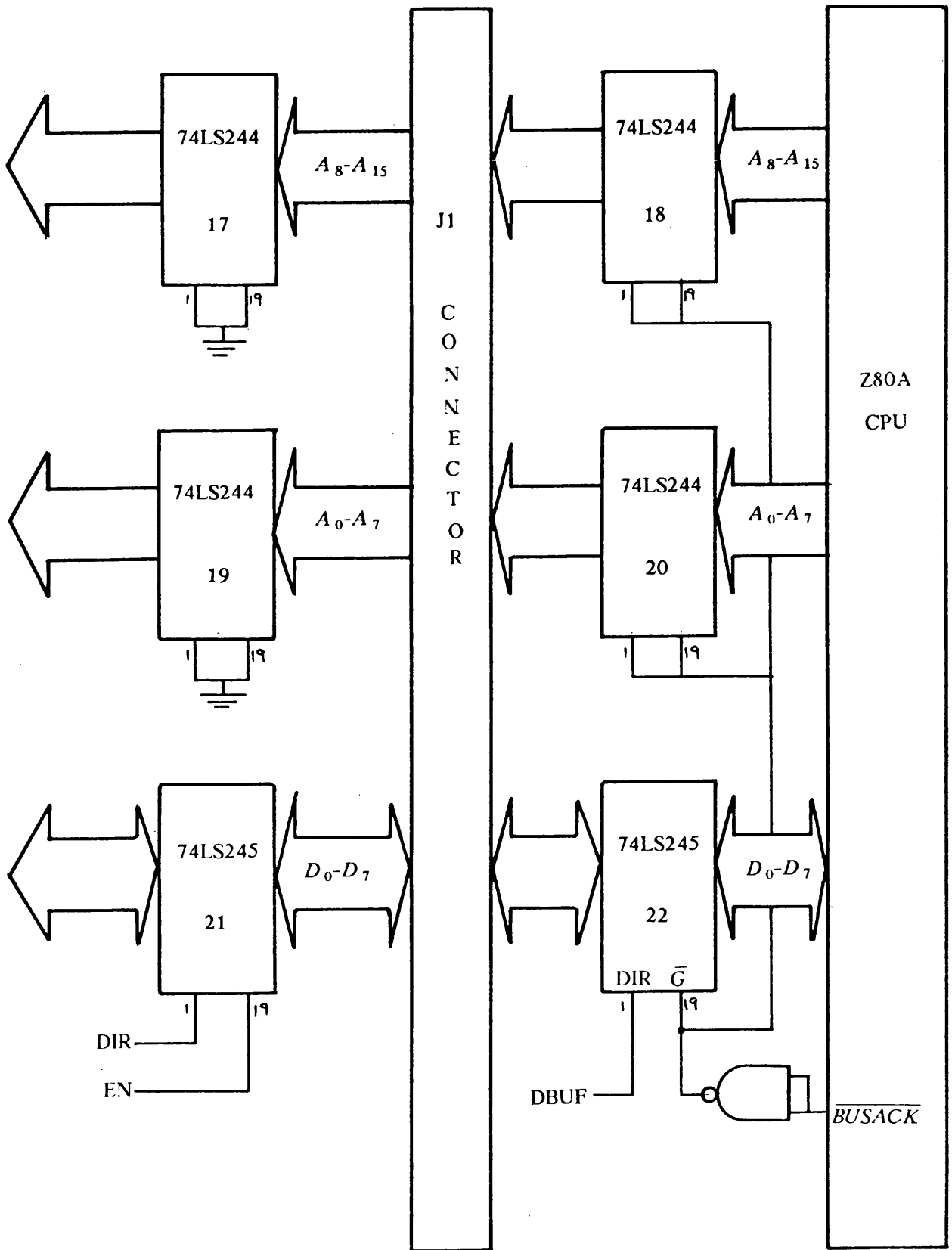


Figure 7.2 Z80 Target Address and Data Buffers

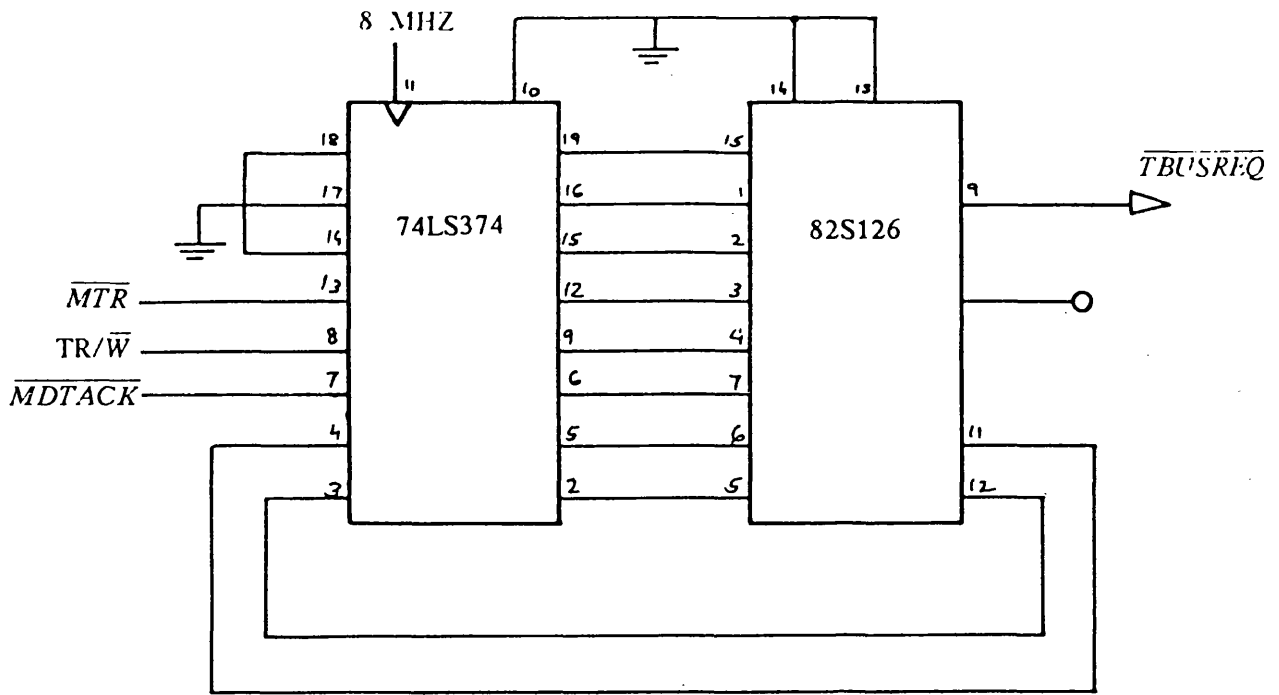


Figure 7.5 Z80 $\overline{TBUSREQ}$ Generation Circuit

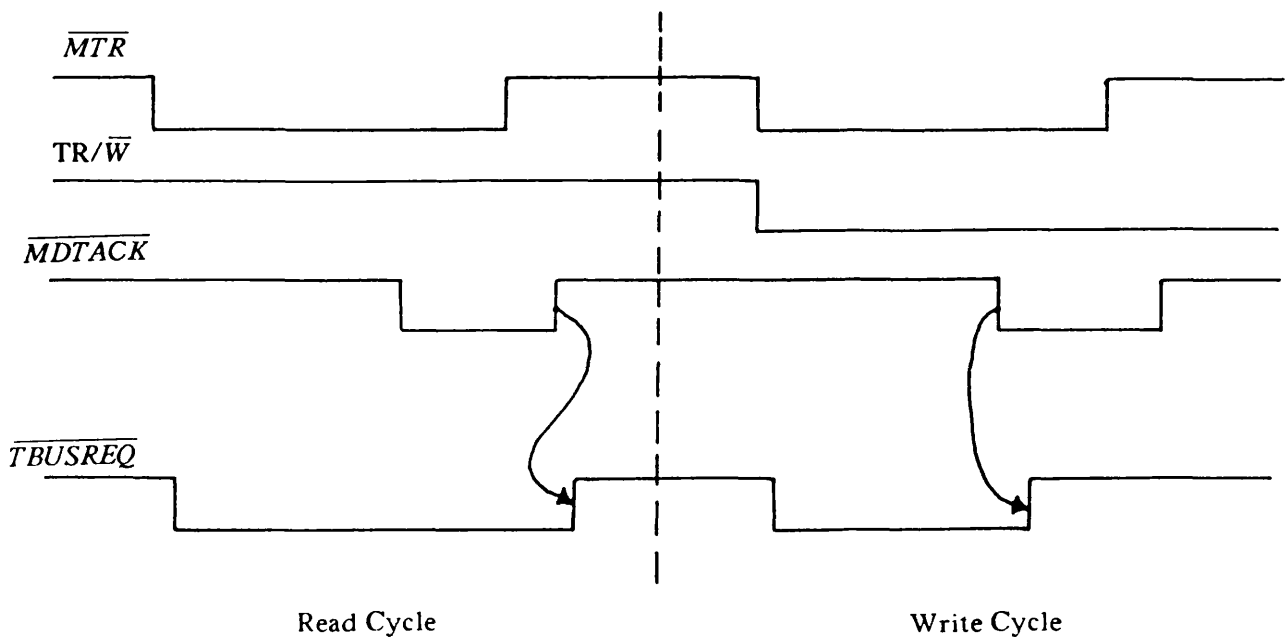


Figure 7.5b Master to Target (Z80) Read/Write Timing Diagram.

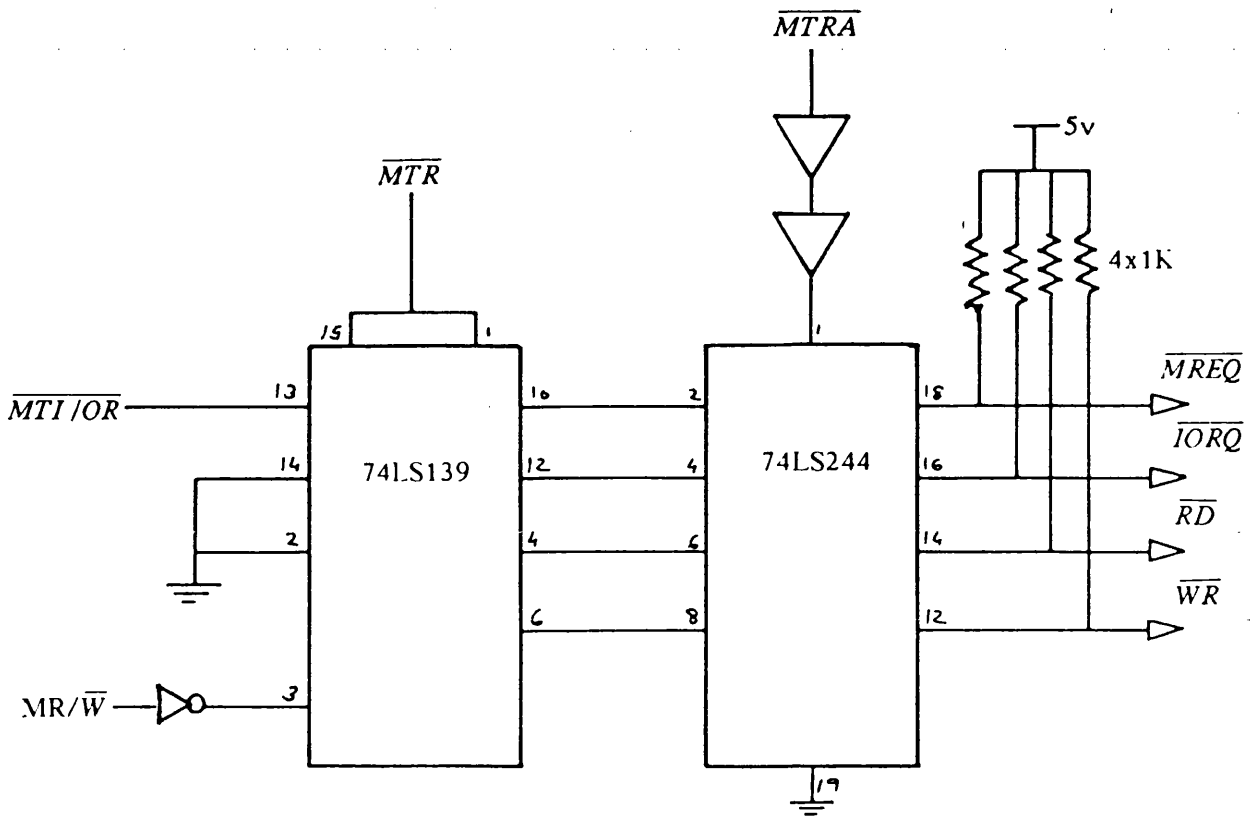


Figure 7.6 Z80 \overline{MREQ} , \overline{IORQ} , \overline{RD} , & \overline{WR} Generation Circuit

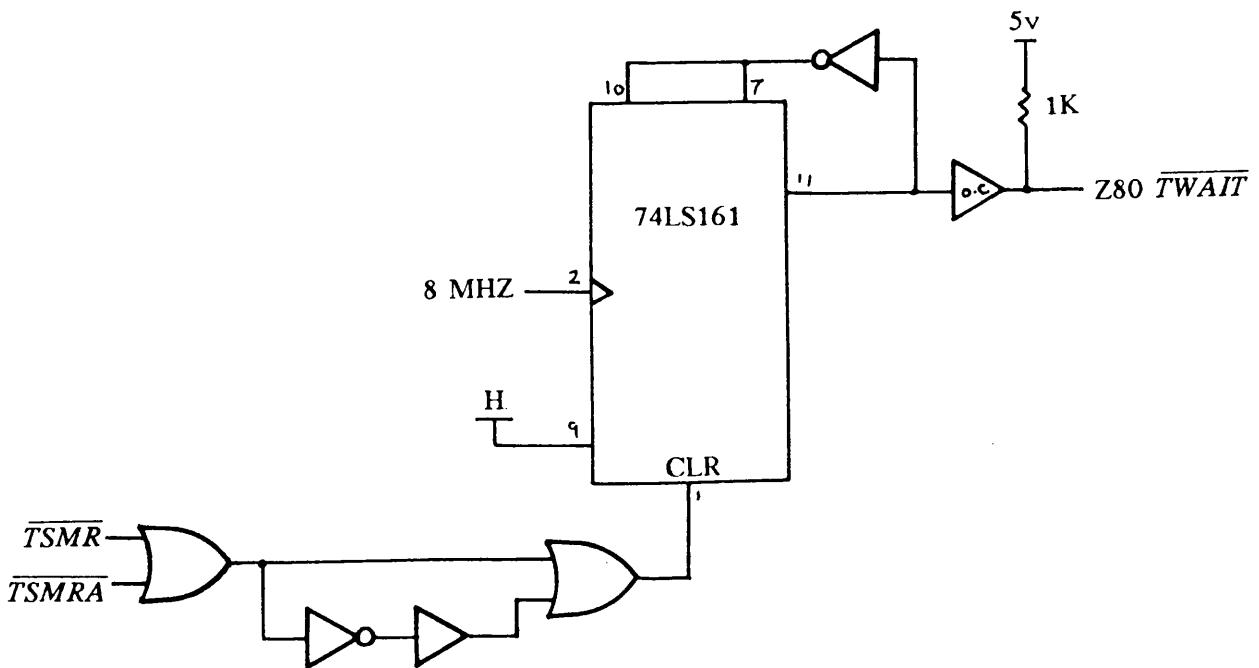


Figure 7.7 Target (Z80) Wait Generation Circuit.

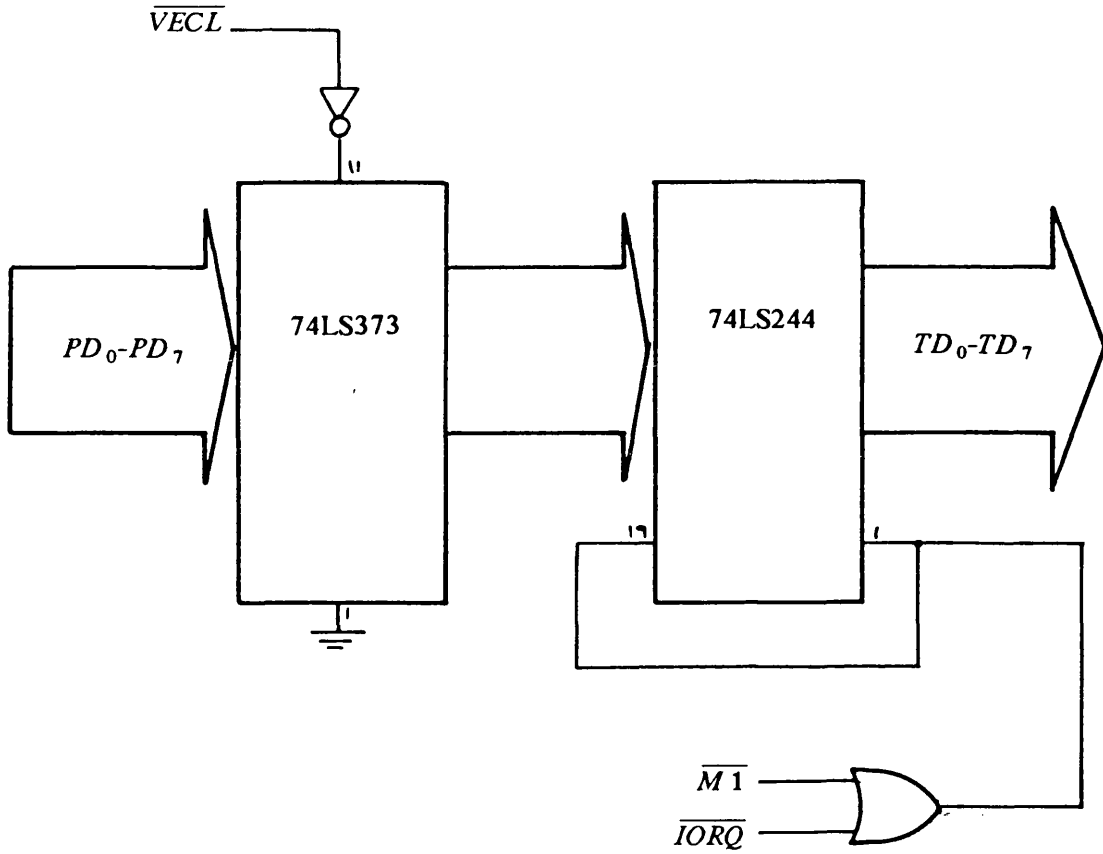


Figure 7.8 Circuit to supply Vector Number during an IACK cycle.

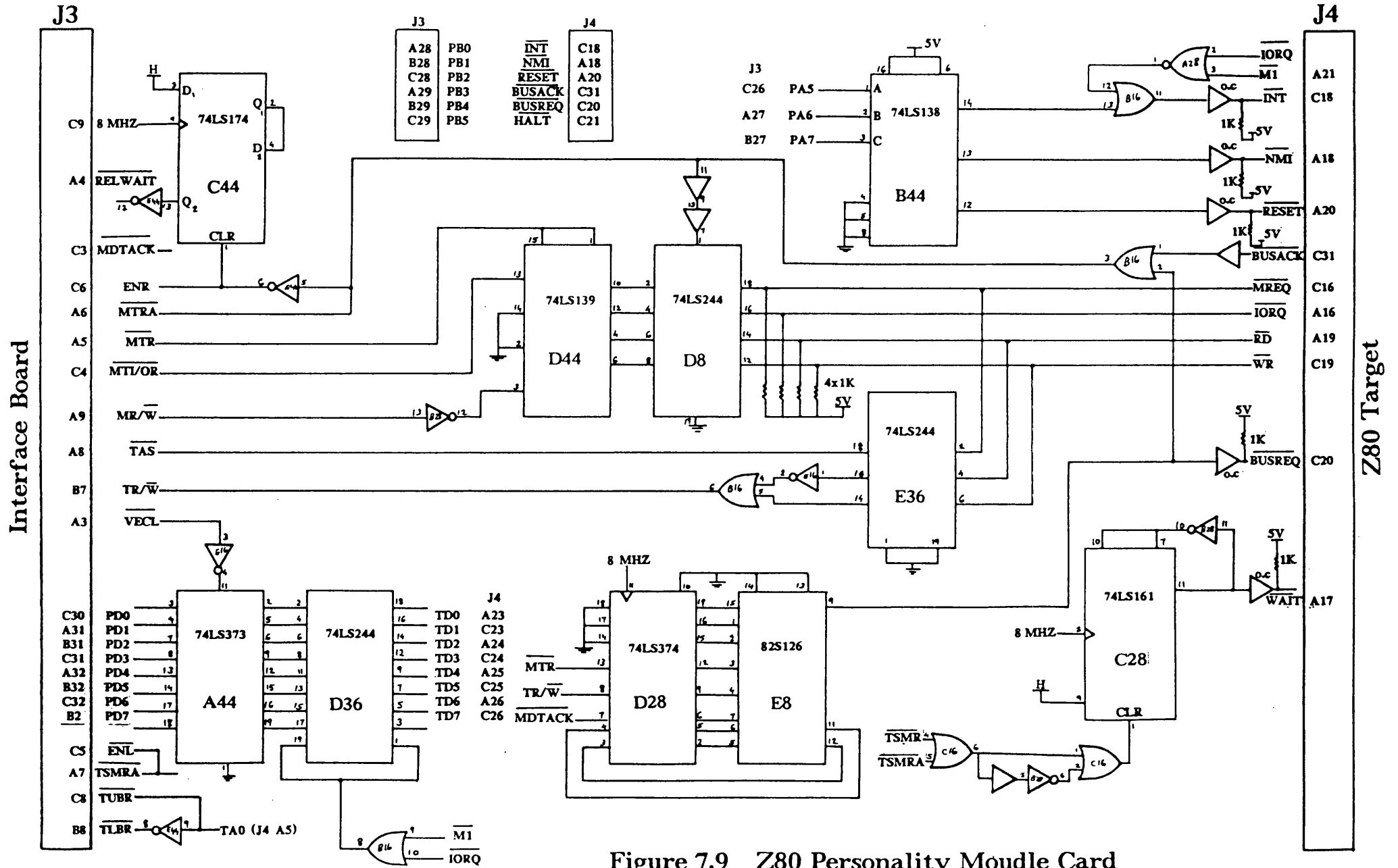


Figure 7.9 Z80 Personality Module Card

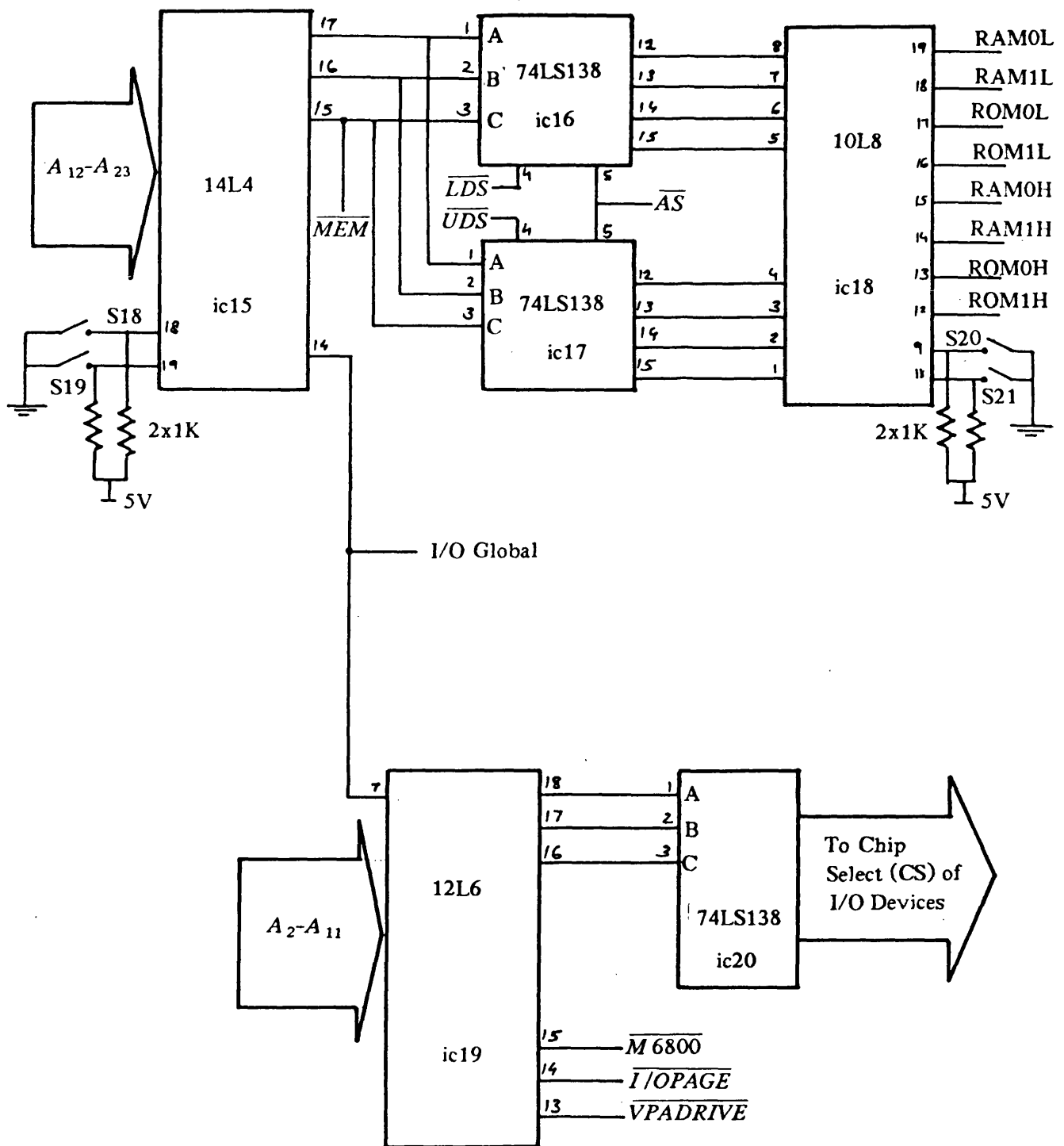


Figure 7.10 Target (MC68000) Memory & I/O Decoding Circuits

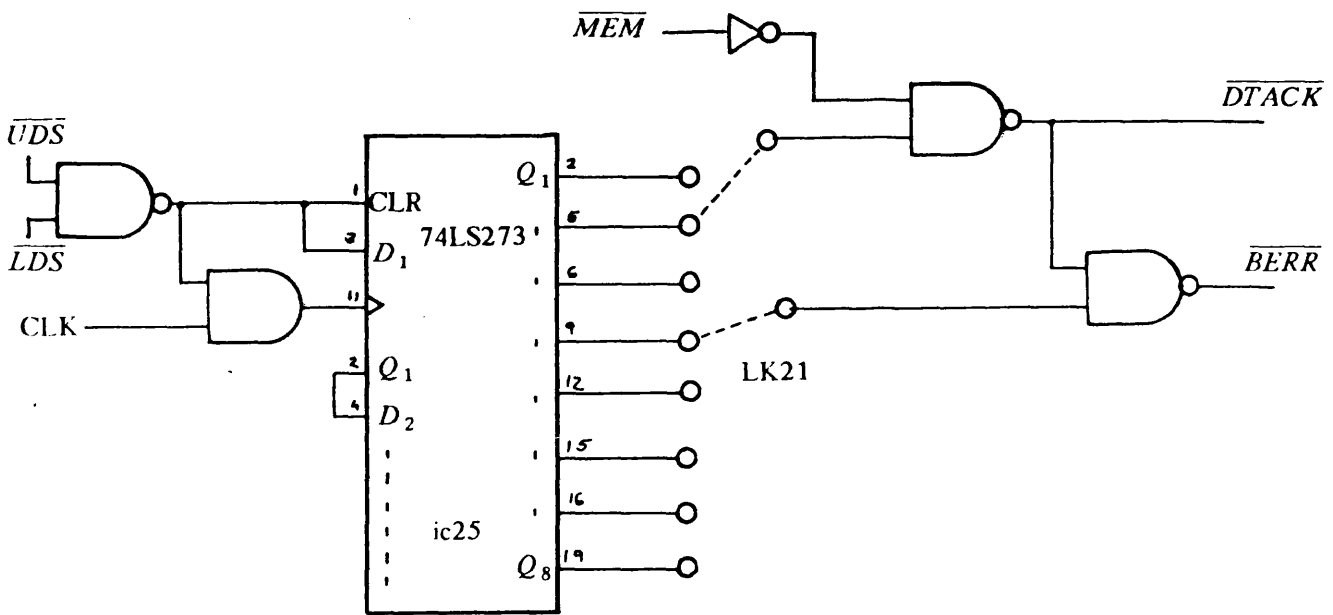


Figure 7.11 Target (MC68000) \overline{DTACK} Generation Circuit

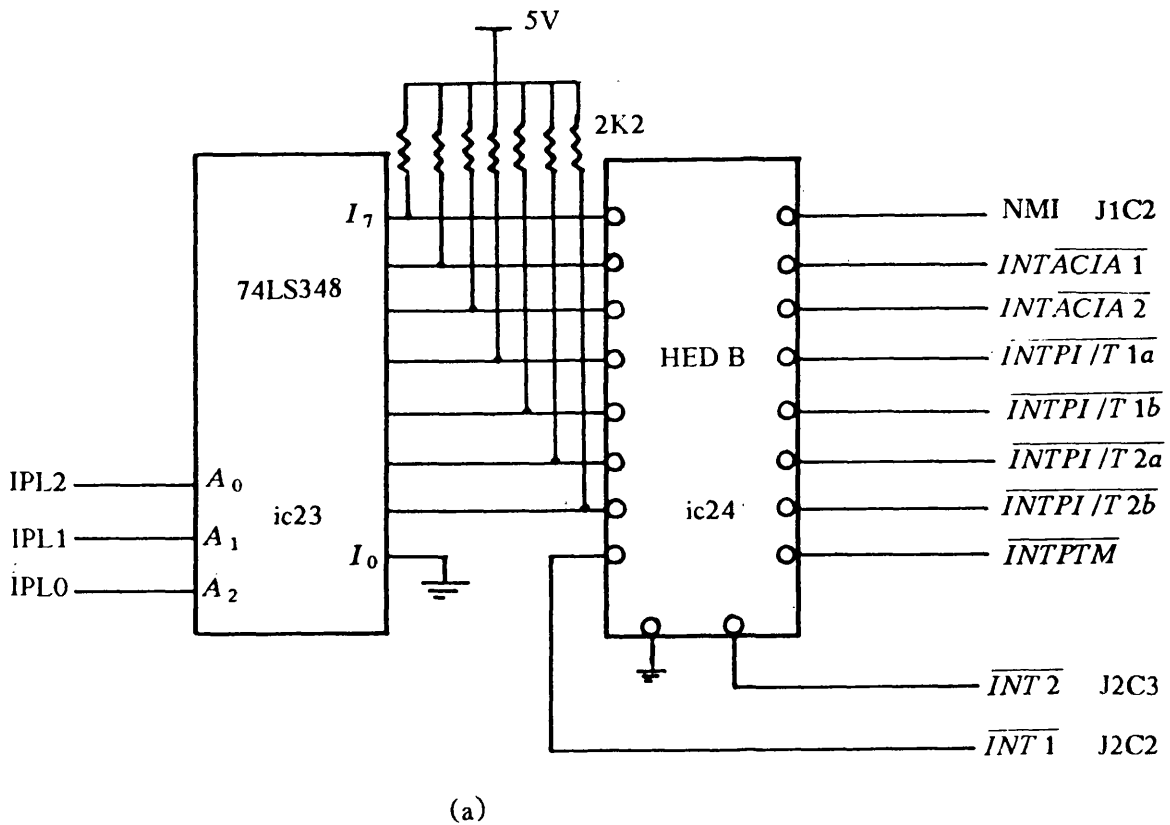
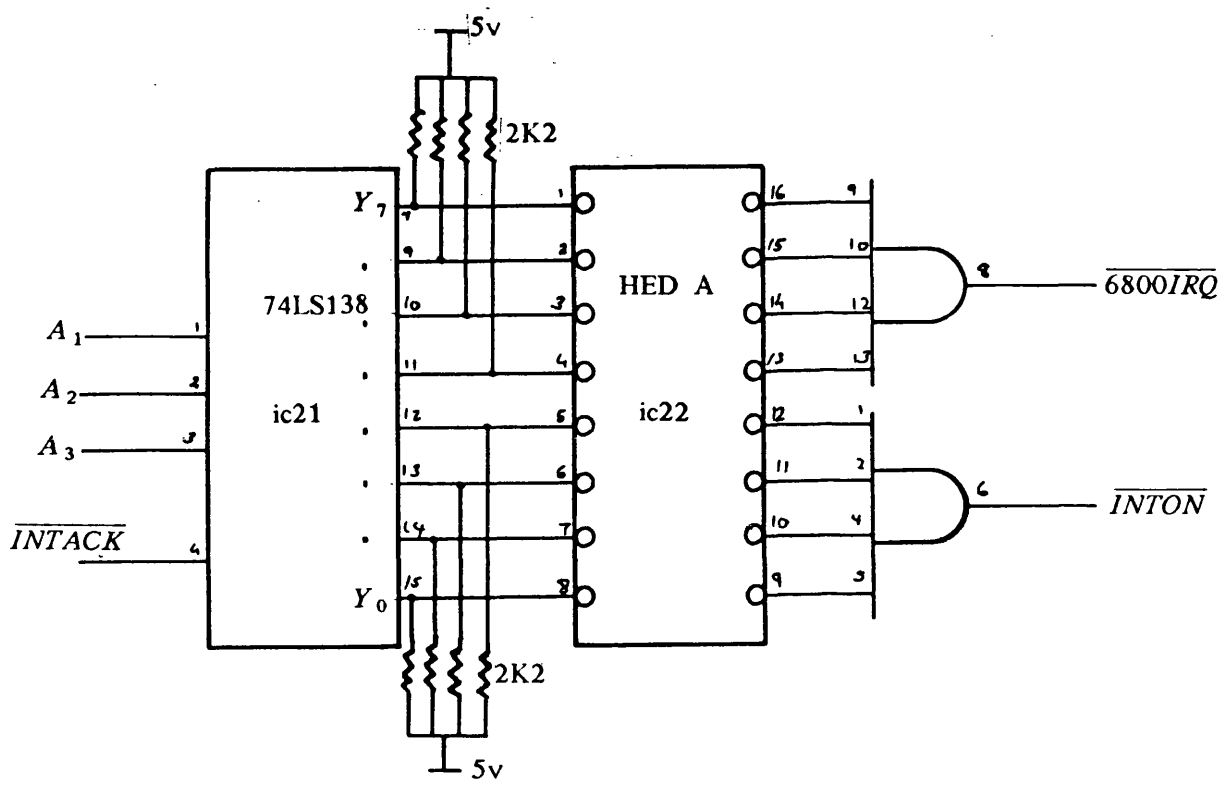


Figure 7.12 Target (MC68000) I/O Devices Interrupts



(b)

Figure 7.12 6800 Interrupt Request and on Board Interrupt Enable

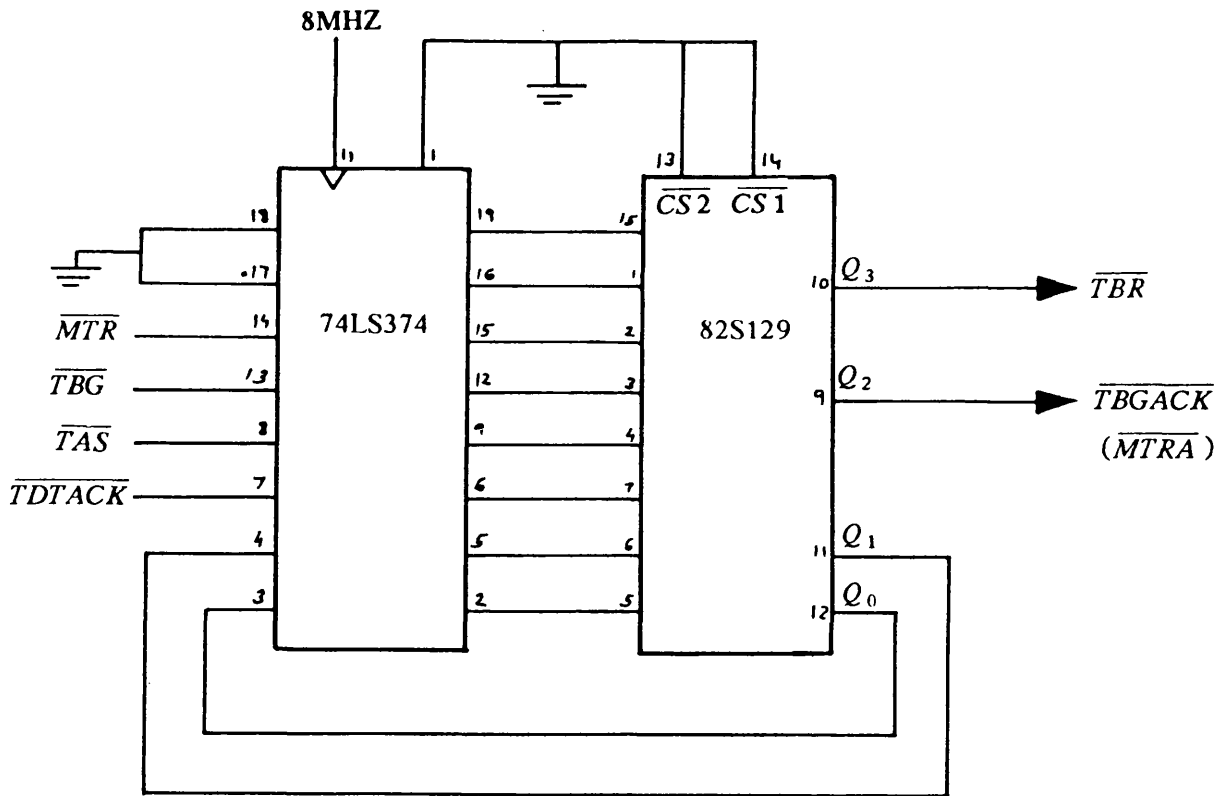


Figure 7.13a M68000 Bus Request Generation Circuit

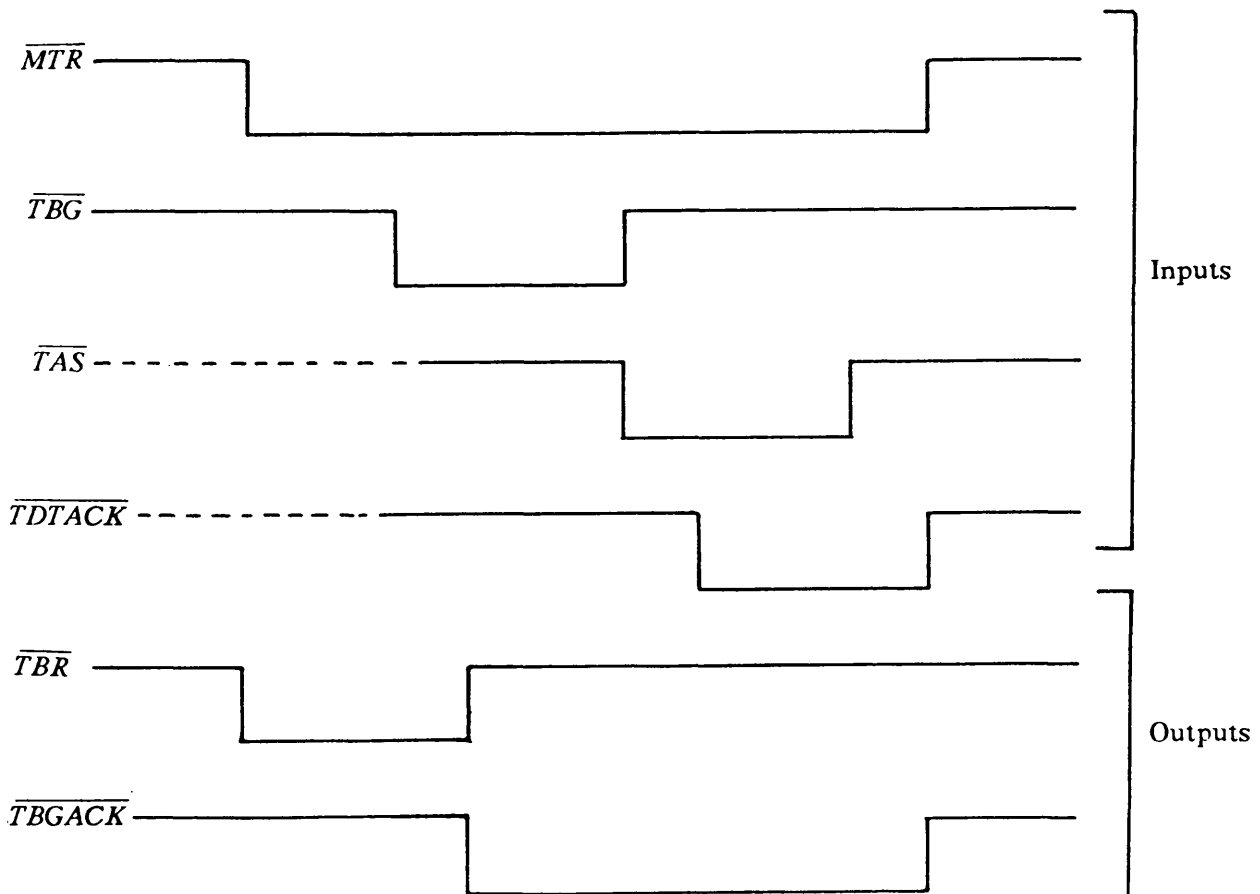


Figure 7.13b Bus Request Cycle Timing Diagram

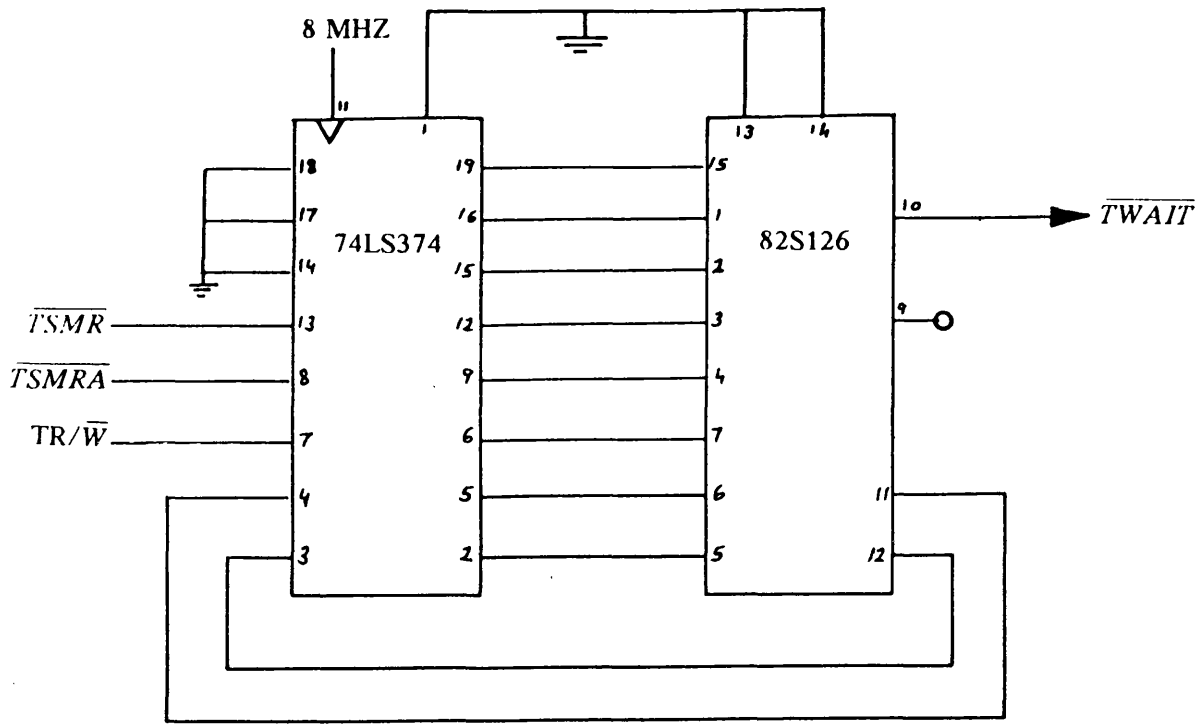


Figure 7.14 (a) TWAIT Generation Circuit During Target (M68000) to S.M Access

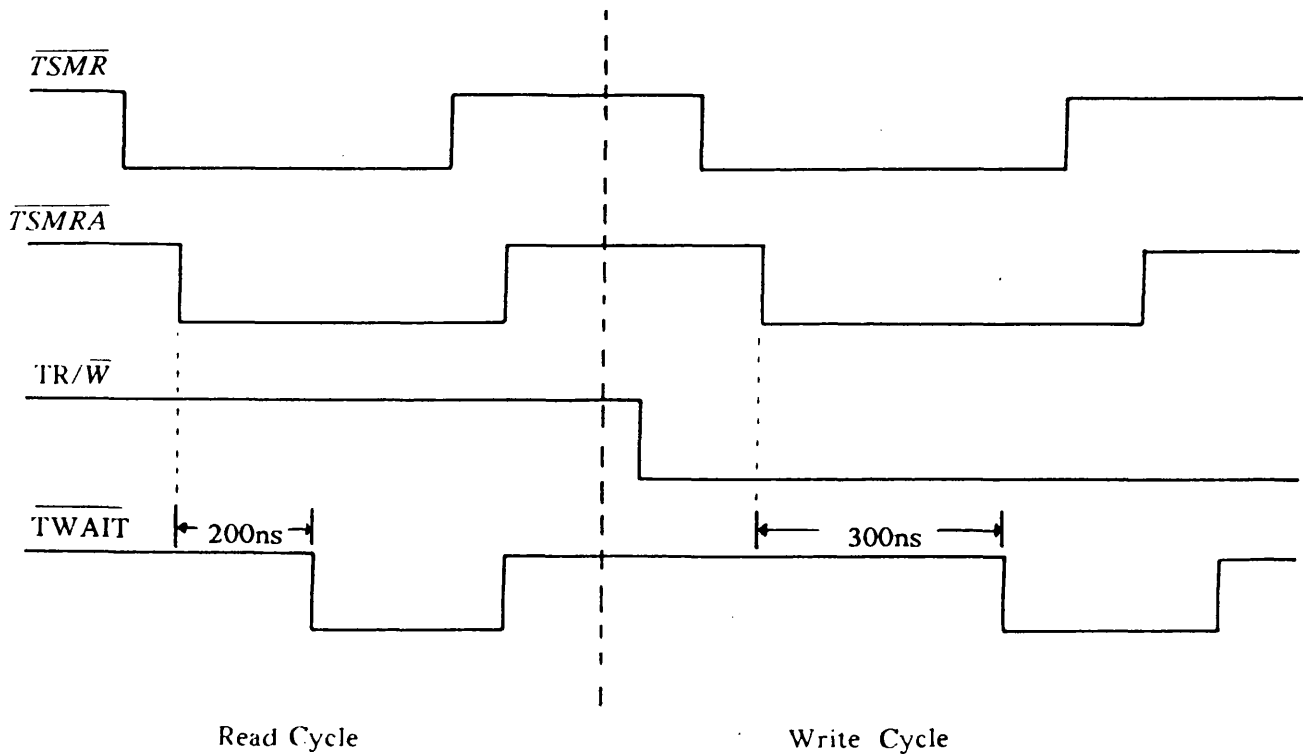


Figure 7.14 (b) TWAIT Timing Diagram

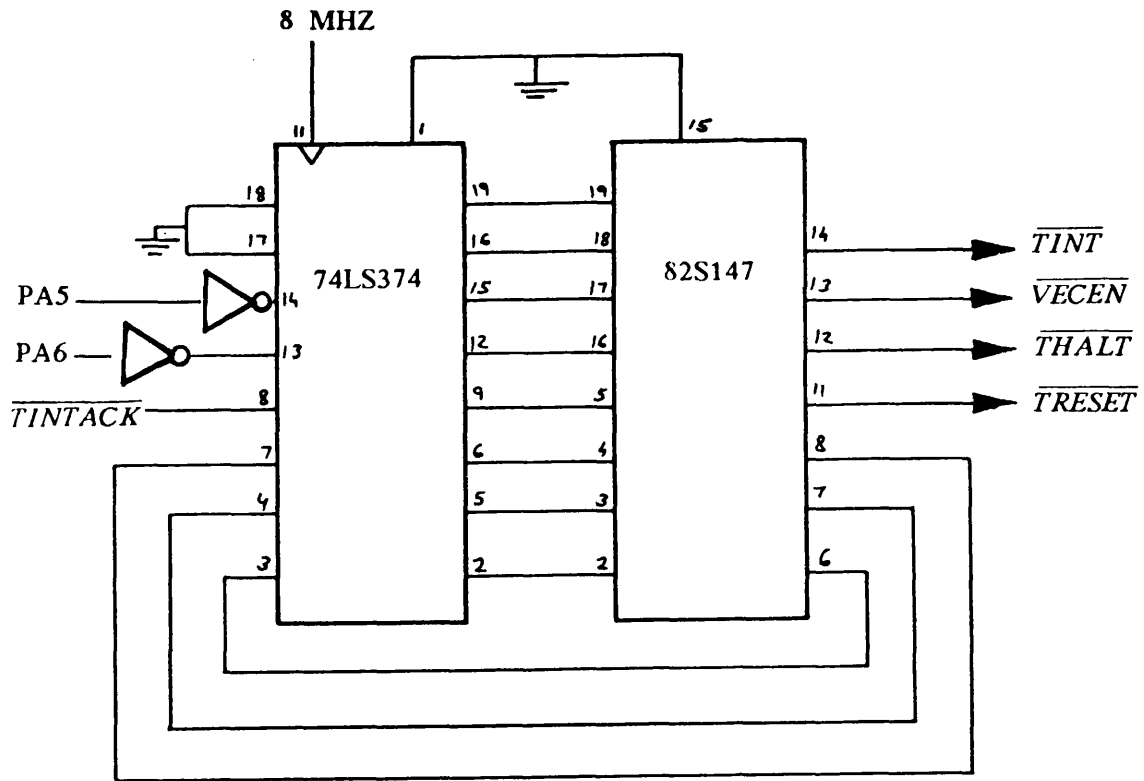


Figure 7.15a Supportive System to Target (M68000): Interrupt, Halt and Reset

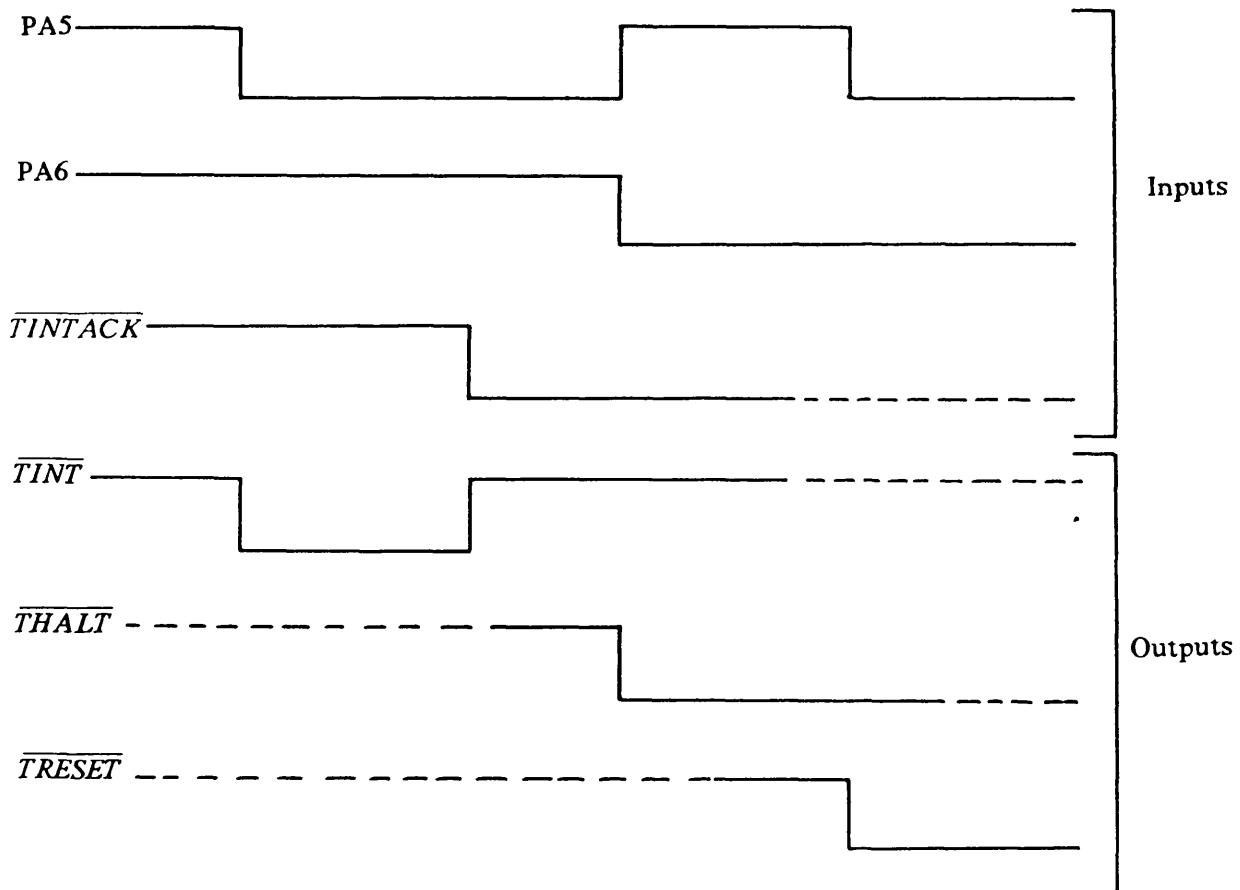


Figure 7.15b Supportive to Target \overline{INT} \overline{HALT} & \overline{RESET} Timing Diagram

Interface Board

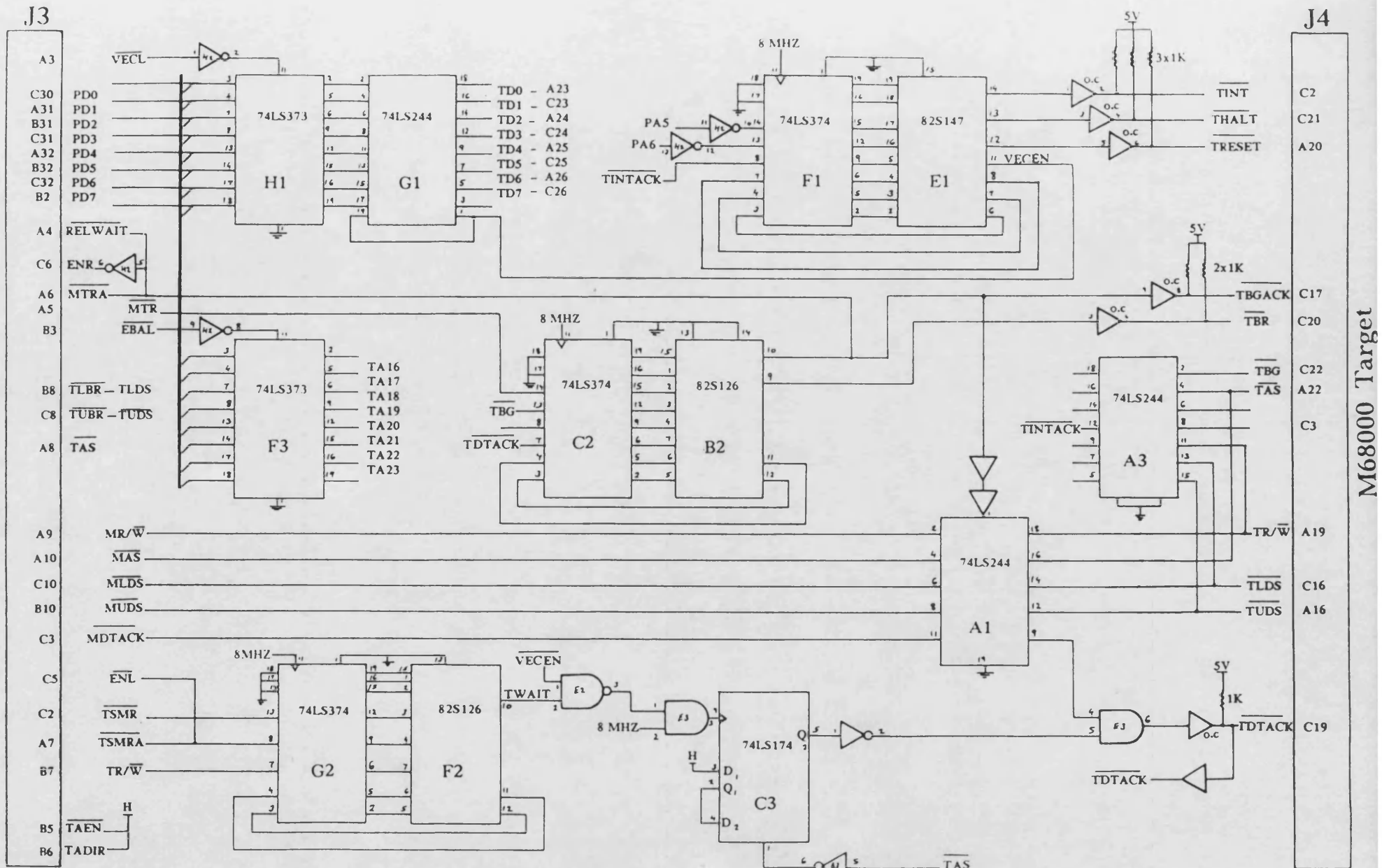


Fig. 7.16 M68000 Personality Module Card

8. SOFTWARE/HARDWARE INTEGRATION

This chapter brings the development of the Educational Interface Board for Multifamily Microprocessor Teaching to its logical conclusion by describing the final stages of system integration and testing under two different operating system environments, TRIPOS and UNIX.

8.1 The UNIX development software environment

In order for the Educational Interface Board to be implemented on the supervisory system supporting UNIX, a device driver is required to be written and build into the UNIX Kernel.

The device driver is the software interface between the peripheral device and the Kernel modules which control the device.

As stated in chapter five, the I/O system of UNIX is designed around two device models, block and character. The block interface is suitable for devices such as disks and tapes which look like a random access storage to the rest of the system and treat data in blocks. The character device interface is suitable for devices which use unstructured input and output transfer such as terminals and network media.

The structure of the UNIX operating system permits the user interface to a device to go through the file system, where all devices (including the educational interface board) are treated and accessed as

regular files. The device file differs from a regular file by the file type located in its inode table which specifies character or block interface.

The internal representation of a UNIX file is given by three system tables, inode table, file table and user file descriptor table. The inode table gives the attributes of the file such as file owner, access permissions and access times. The file table contains global Kernel information such as the byte offset in the file where the user's next read or write will start. The file descriptor table is allocated for every process, and contains information which identifies all open files for a process. A file descriptor is returned by the Kernel for *open()* and *creat()* system calls.

The device file interface is mainly consists of a few system calls which perform special operations. They include open, read, write and close a device. The algorithms which handle these functions are part of the Kernel. The Kernel first executes the *open()* system call, which opens the device file and sets up entries in the system tables, to allow a process to communicate and access data on the device file. The Kernel then returns the user file descriptor to the calling process. When executing *read()* or *write()* system calls, the Kernel uses the file descriptor as an index to access the three system tables, and from the inode table the Kernel locates the required data. For character devices such as the interface board driver, the input output control system call, *ioctl()*, is used to provide an interface which enable processes to control the device. The *ioctl()* system call has the following notation :

ioctl(fd, command, arg);

Where *fd* is the file descriptor returned previously by the *open()* system call. *Command* is a request passed by the user program to the driver to perform certain actions such as accessing shared memory or target memory. *Arg* is a parameter which points to a structure. When a process is no longer required to access an open device, the Kernel closes it by executing the system call *close()*. The Kernel manages the close operation by manipulating the file descriptor and the corresponding file table and inode table entries.

The development of the device driver goes through the following phases : i) developing the driver software on a UNIX machine which is provided with all parts of the Kernel. ii) building the UNIX Kernel. iii) Transferring the Kernel to any other UNIX development system. iv) testing and debugging. Once the driver is built into the system, the environment for software development and debugging becomes very restricted,

and any development or correction made to the driver would require going over the development phased mentioned above. This can result in frustrating development efforts. On the other hand, since the driver is written in C language, this can improve the development time and allow for higher level functions to be included in the driver.

For the MC68000 educational interface board, a UNIX special character device called *"/dev/mā"* has been created within the filing system.

The user program is responsible for feeding the driver with the required information to carry out the requested tasks. The user has to define in his program a buffer, where the data is to come from or go to, size of the buffer, which represent the number of characters to be transferred, and also has to set an address offset. Before the user can actually perform any read or write to the target memory a request must be made to the Kernel to open the specified device file. The arguments of the open call specify the device name and read write mode type. If the open is successful, it returns a valid file descriptor and the device is ready for action. An unsuccessful attempt will result in the return of -1. The user, is then required to use the *ioctl()* call to inform the driver of type of action to be taken. As the Kernel receives the request command, it compares it with the command options supported by the driver. If a successful match, then the requested routine will be called, else an input output error is reported to the user.

If a valid request has been made, then all the input and output communications is done using the two calls *read()* and *write()*. For both calls, the first argument is the file descriptor returned previously by the *open()* call. The second and third arguments are the buffer and buffer size,

which were defined earlier in the user program. Each of the read and write calls returns a byte count which specifies the actual number of data transferred. A returned number of zero indicates an end of file, and -1 will signal to the user that there is an error of some kind as

the returned byte count in a write call does not equal the number of bytes supposed to be written.

For shared memory read accesses, the first test is to check that the total number of characters to be read does not exceed the shared memory space limit, which was set to 4 kbytes. If the test is successful, shared memory address is set to the shared memory address $(860000)_{16}$ plus any address offset supplied by the user. A counter is set to zero, and a test is made on the number of characters. While the number of characters is valid, a *pass(c)* routine is called to return characters to the user, and the shared memory address and the counter are increased by one. For shared memory write access, a *cpass()* routine is called to pick up characters from the user's buffer. Characters are to be transferred until the byte count goes to zero or an error occurs, where a -1 is returned.

8.1.1 Operation procedure

First, the user selects the target microprocessor board which is required to be studied. Its appropriate personality module card is then plugged into the educational interface board, and the two boards are plugged into the provided system backplane slots.

In order not to generate any illegal interrupt which will cause the system to enter a halt state, it is preferable that the UNIX system is shut down before any boards are plugged in. After all the cards are in place, the system can then be rebooted. During the bootstrapping procedure, the Kernel begins an initialisation phase which includes all

the drivers. And as the educational interface board is initialised, communication with the target system can start. The bootstrapping procedure of the UNIX system has been described in chapter 5.

8.1.2 Support software

Apart from the wide range of utilities and programming tools supplied with the UNIX system, such as the C and Fortran compilers and the Omnia assembler, a number of small software programs have been written specifically for the support and testing of the master/target interface environment. The *exmem.c* is a short C language program which examines the contents of either target or shared memory locations. When it is executed, the user is required to enter T for target memory or S for shared memory, start address in hex and the number of bytes required to be examined. The *hexload.c* is another C program used to download the Intel Hex records into the specified target memory location. The loaded code can then be executed by issuing a RUN command. Other commands have also been tested. They include halt and reset the target system, stop and resume program execution and interrupting target processor. A number of Z80 and MC68000 assembly code programs have also been written to test target to shared memory read and write, moving blocks of data between the target and shared memories, reading from target input port and writing to target output port for visual display.

After verifying that the master/target interface is operating correctly and as expected, an educational debug software system can then be developed, in the programming language C, to provide a

debugging environment facility for monitoring the target activities via direct memory access.

The debug tool, which consists of a range of commands, is to provide

the following essential features :

- i. Direct data manipulation of shared memory, target memory and target I/O locations.
- ii. Data transfer between shared and target memories.
- iii. Load and save in Intel hex format.
- iv. Examine and modify target registers.
- v. Single step and execute program. In single stepping the program is executed one single instruction at a time to allow the user to inspect the contents of memory, registers and to check that the results are as those expected.
- vi. Relative jump offset insertion.
- vii. Breakpoints insertion. This feature is to enable the user to view the effects of memory accesses at specified addresses in program memory.

8.1.3 The software development cycle

The software development cycle goes through several development phases before the program is successfully executed in the target memory. The first step in the development process is defining the functions of the program, followed by the designing

phase. Then comes the phase of coding the program in either symbolic assembly language or high level language. Using the ready available powerful UNIX tools such as editors and file management, the code is typed and saved as a source file. Depending on the source file type, the Omnia assembler or a cross compiler is used to translate the sources code to an Intel hex format object file. The object code is down loaded, via direct memory access, into the target memory and debugged. At the end of the debugging phase, the development process enters its final phase by executing the program on the target system.

A key advantage of choosing UNIX, in this study, over other available operating systems is its multiuser environment and its powerful utility tools. As users benefit from sharing the expensive devices such as high speed printers and storage media, they also benefit from sharing only one target system. During the software development cycle, users usually spend a great deal of time in designing, coding and typing their programs before they actually reach the stage of downloading the object code, into the target memory, for debugging. At this stage only one user is allowed to communicate with the target system, the other users would be busy at different development phases.

8.2 The TRIPOS development software environment

As have been seen in chapter 5, TRIPOS and its programming language provide a good and simple environment for hardware development. Because TRIPOS device drivers (unlike UNIX drivers) are not an integral part of the Kernel, no special Kernel is needed to be build or rebuild each time any changes or corrections are made to the driver. This feature will result in a simple and straight forward implementation of new devices. The time taken to produce a TRIPOS device driver is considerably reduced by this feature, and the time for debugging and implementing test programs is also less.

Some TRIPOS devices can even communicate directly with the microprocessor without the need for drivers. These devices are memory mapped and they use the system backplane bus for communication. If a device is required to interrupt the processor for any reason, then a device driver is required. The packet transfer technique, which is used to communicate between two tasks or a task and a device driver, and the structure of TRIPOS device driver have been described in chapter 5.

Once the educational interface board and the selected target board are plugged into the provided slots, power is applied to the system. The supervisory disc based system is then boot strapped, from the system floppy disc, to load the TRIPOS operating system, and various operating system start up procedures, such as setting time and date, are performed.

A set of software test programs, similar to those written under the UNIX environment, have been developed in BCPL language to support the multi-family microprocessor interface environment running TRIPOS. Two different target boards, the Zilog Z80 and the Motorola MC68000, have been used successfully in the testing process.

The use of two different target processors is to demonstrate the universality of the technique used in this project. Eight, sixteen and thirty two bit processors can be interfaced to the MC68000 supportive system through the educational interface board. But due to pin limitations in J1 connector of the supervisory system backplane, only byte and word accesses are possible. And the integration of educational interface hardware with two different software environments is to demonstrate the flexibilities of the approach used to the problem of multifamily microprocessor education and development.

9. CONCLUSIONS

The aim of this study has been to develop an economical educational environment to allow students to examine and understand the behaviour of the currently available 8, 16 and 32 bit microprocessor families.

It is apparent, from the review of the available microcomputer educational systems described in chapter 2, that the in-circuit emulator is one of the most powerful techniques available for this purpose. However it is probably also the most complex and expensive approach. The high cost of such specialized systems has forced the manufacturers of in-circuit emulator based development systems to offer communication link programs and high-level software development and debugging tools for use with a wide range of host computers in order to allow users to connect in-circuit emulators to their own host computer. For educational institutions, the provision of a sufficient number of such working stations is often a severe financial constraint.

An examination of the bus structures of various microprocessor families has shown that there is little fundamental difference between the control timing sequences of many processors. These sequences are used for address and data validation as well as direction control and most microprocessors also have provision for

direct memory access. The success of this study has been based on the ability to exploit these common features to transform between the bus signals of different processors in order to provide a universal development environment.

The general purpose Educational Interface Board (EIB) which has been designed and implemented provides a communication environment for users to study, monitor and control the operation of a wide range of different microprocessor based target systems. The supervisory and target processors form an asynchronous, shared memory multiprocessor system. This development environment can be used to compare the performance of microprocessors from different families quickly and simply, without having to invest in the complete development system marketed by each manufacturer.

The MC68000 processor was chosen as the supportive processor because it was considered to be the most powerful and versatile microprocessor available when this study started. Several features of its architecture support the implementation of the two sophisticated operating systems, the single user TRIPOS system and the multi-user UNIX system, used during this study. These features include the large linear addressing space available, dual-state processing and a seven level interrupt priority scheme. The architecture of the MC68000 is also conducive to the use of high level languages such as BCPL and C which are, respectively, the primary development languages for the two operating systems implemented. Use of the relatively advanced MC68000 processor with its large hardware capability on the

supportive section of the development system has meant that the component count on the interface board used with a particular target processor has been minimised.

This hardware interface has been successfully tested under two different software environments. The TRIPOS operating system was used first because of its inherent simplicity and the straight forward manner in which it handles hardware. This simplified debugging and made test software easy and quick to implement. The recent introduction of the Commodore Amiga machine (which operates under a TRIPOS based operating system and uses the MC68000) has further increased the popularity of TRIPOS and a range of high level languages and software development packages are now available.^[47]

TRIPOS is, however, essentially a single user system but its portability makes it cost effective and would allow the provision of enough work stations for multi-family microprocessor educational purposes.

During the course of this study, the UNIX operating system became well established for MC68000 based computing systems and the development system has been integrated into and operated in this environment. It offers several advantages to the user. Additional to the wide acceptance of this operating system, it offers many tools directly applicable to software development for microprocessors. These include tools for program editing , document creation and

formatting, file maintenance and project management. The richness of the native capabilities of the system makes it easily extensible to tasks it was not designed to perform originally, such as control of external microprocessor based target systems. In the education and training field, a unified procedure for file editing, storage, downloading of object code, debugging and the monitoring of target systems is an attractive feature. Since the multi-user/multi-tasking capability is the most important feature of this operating system, users have the benefit of sharing the full system resources.

They can, therefore, each simultaneously access the system during all the phases of the microprocessor software development cycle and development programs can be shared among several users. No duplication of target hardware is required and the UNIX environment has proved to be highly successful when used with multiple microprocessor software development system in the educational environment.

The proposed microprocessor development system has the major advantages of universality, flexibility and economy. The number of components needed in each target system has been minimized and most of the necessary complexity is associated with the universal supportive system. It should be possible to interface any currently available microprocessor to the supportive system using the EIB. The hardware is capable of being integrated into several software environments, thereby providing the user with his own choice. The system is easy to use and appears to be cost effective. The EIB used in

the UNIX environment therefore seems to satisfy the ever increasing demand for multi-family microprocessor education at low cost.

APPENDIX A : Supportive System Bus Specification

Back-plane pin-outs

Edge Connector J1		
Pin No.	Row a	Row b
32	+5V	+5V
31	-5V	-5V
30	D14	D15
29	D12	D13
28	D10	D11
27	D8	D9
26	D6	D7
25	D4	D5
24	D2	D3
23	D0	D1
22	\overline{AS}	\overline{BG}
21	ECLK	\overline{HALT}
20	\overline{RESET}	\overline{BR}
19	R/ \overline{W}	\overline{DTACK}
18	\overline{VMA}	\overline{VPA}
17	\overline{BERR}	\overline{BGAGK}
16	\overline{UDS}	\overline{LDS}
15	CLK	A20
14	A18	A19
13	A16	A17
12	A14	A15
11	A12	A13
10	A10	A11
09	A8	A9
08	A6	A7
07	A4	A5
06	A2	A3
05	A21	A1
04	A22	A23
03	-12V	-12V
02	+12V	+12V
01	0V	0V

Back-plane pin-outs

Edge Connector J2		
Pin No.	Row a	Row b
32	+5V	+5V
31	+15V	-15V
30	FC0	DLY1
29	FC1	DLY2
28	FC2	DLY3
27	FC3	DLY4
26	\overline{INTACK}	DLY5
25		DLY6
24		DLY7
23	$\overline{IRQ 7}$	DLY8
22	$\overline{IRQ 6}$	\overline{IOPG}
21	$\overline{IRQ 5}$	$\overline{68PG}$
20	$\overline{IRQ 4}$	
19	$\overline{IRQ 3}$	
18	$\overline{IRQ 2}$	
17	$\overline{IRQ 1}$	
16	\overline{BGOUT}	\overline{BGIN}
15	\overline{IAOUT}	\overline{IAIN}
14		
13		
12		
11		
10		
09		
08		
07		
06		
05		
04		
03		
02		
01	0V	0V

Master Signal Descriptions :-

D_n	Data bus lines
A_n	Address bus lines
$\overline{68PG}$	Partial address decode to signal 6800 device page address.
\overline{IOPG}	Partial address decode to signal IO device page address.
\overline{AS}	System address strobe
\overline{UDS}	Upper data strobe - when asserted signals $D_{15}-D_8$ valid
\overline{LDS}	Lower data strobe - when asserted signals D_7-D_0 valid.
R/\overline{W}	Read write line
\overline{DTACK}	Data acknowledge - signals successful completion of bus cycle.
\overline{BERR}	Bus error - signals and terminate bad bus cycle.
\overline{FCn}	Function codes - signals bus cycle type.
DLY_n	Delay signals - DLY_1 is asserted 1 clock cycle following assertion of data strobe. Used by peripherals for timing \overline{DTACK} .
\overline{VPA}	Valid 6800 peripheral address - driven low by 6800 type devices to start 6800 bus cycle.
\overline{VMA}	Valid memory address.
$ECLK$	6800 device clock - 1 MHZ clock used for 6800 device synchronous bus cycles.
CLK	System and arbitration clock - 8 MHZ.
\overline{BR}	Bus request - driven by bus master to request bus access.
\overline{BGOUT}	Bus grant - out daisy chain signal. Backplane connects \overline{BGOUT} to \overline{BGIN} of successive slots.
\overline{BGIN}	Bus grant - in daisy chain signal - bus masters receive bus grant on this pin and propagate it on \overline{BGOUT} .

- \overline{IREQ}_n Interrupt request-driven low by requesting peripheral.
- \overline{IAOUT} Interrupt acknowledge out daisy chain signal-backplane connects \overline{IAOUT} to \overline{IAIN} on successive slots.
- \overline{IAIN} Interrupt acknowledge in daisy chain signal-peripherals receive interrupt acknowledge on this pin and propagate to successive slots by driving \overline{IAOUT} .

APPENDIX B : Z80 Target Bus Specification

Back-plane pin-outs

Edge Connector J1		
Pin No.	Row a	Row b
32	+5V	+5V
31	BAI	BAO
30	PULLED UP	2*CLK
29	PULLED UP	PULLED UP
28		
27		
26	D6	D7
25	D4	D5
24	D2	D3
23	D0	D1
22		
21	$\overline{M1}$	\overline{HALT}
20	\overline{RESET}	\overline{BREQ}
19	\overline{RD}	\overline{WR}
18	\overline{NMI}	\overline{INT}
17	\overline{WAIT}	\overline{RDY}
16	\overline{IORQ}	\overline{MREQ}
15	CLK	\overline{RFSH}
14	0V	0V
13	0V	0V
12	A14	A15
11	A12	A13
10	A10	A11
09	A8	A9
08	A6	A7
07	A4	A5
06	A2	A3
05	A0	A1
04	IEO	IEI
03	-12V	
02	+12V	
01	0V	0V

APPENDIX C : M68000 Target Bus Specification

Back-plane pin-outs

Edge Connector J1		
Pin No.	Row a	Row b
32	+5V	+5V
31		A22
30	D14	D15
29	D12	D13
28	D10	D11
27	D8	D9
26	D6	D7
25	D4	D5
24	D2	D3
23	D0	D1
22	\overline{AS}	\overline{BG}
21	\overline{ECLK}	\overline{HALT}
20	\overline{RESET}	\overline{BR}
19	$\overline{R/W}$	\overline{DTACK}
18	\overline{VMA}	\overline{VPA}
17	\overline{BERR}	\overline{BGAGK}
16	\overline{UDS}	\overline{LDS}
15	CLK	A20
14	A18	A19
13	A16	A17
12	A14	A15
11	A12	A13
10	A10	A11
09	A8	A9
08	A6	A7
07	A4	A5
06	A2	A3
05	A21	A1
04		A23
03	-12V	\overline{INTA}
02	+12V	\overline{INT}
01	0V	0V

APPENDIX D : PAL Equations

This Appendix describes the signals generated by the PALs and the equations used.

As has been shown in Figure 7.10, the memory and I/O decode of the M68000 target system is achieved using a 14L4 PAL driving a pair of 74LS138 decoders.

Three outputs from the PAL drive the A,B and C inputs of the LS138 pair, the remaining output is used to enable the I/O decode circuit with a global enable of a 2K area at 80000_{16} to $80FFF_{16}$.

$$\begin{aligned} \overline{MEM} &= \overline{Ax} \cdot \overline{A15} \cdot \overline{A14} \cdot \overline{RAM} \cdot \overline{ROM} \\ &+ \overline{Ax} \cdot \overline{A15} \cdot \overline{A14} \cdot \overline{A13} \cdot \overline{RAM} \cdot \overline{ROM} \\ &+ \overline{Ax} \cdot \overline{A15} \cdot \overline{RAM} \cdot \overline{ROM} \\ &+ \overline{Ax} \cdot \overline{RAM} \cdot \overline{ROM} \\ &+ \overline{Ax} \cdot \overline{A15} \cdot \overline{RAM} \cdot \overline{ROM} \\ &+ \overline{Ax} \cdot \overline{A15} \cdot \overline{A14} \cdot \overline{A13} \cdot \overline{RAM} \cdot \overline{ROM} \end{aligned}$$

$$\text{Where } \overline{Ax} = \overline{A23} \cdot \overline{A22} \cdot \overline{A21} \cdot \overline{A20} \cdot \overline{A19} \cdot \overline{A18} \cdot \overline{A17} \cdot \overline{A16} \cdot \overline{A15} \cdot \overline{A14} \cdot \overline{A13} \cdot \overline{A12}$$

The first term enables the memory decode dependent on the address lines and the RAM and ROM size selection.

The next two terms, A and B, can be considered as a two bit encoded signal that is then decoded by the 74LS138 and via a further PAL to select one of four pairs of RAM or ROM. As the target is a 16 bit processor the devices are selected in pairs.

$$\begin{aligned} A &= \overline{Ax} \cdot \overline{A15} \cdot \overline{A14} \cdot \overline{A12} \cdot \overline{RAM} \cdot \overline{ROM} \\ &+ \overline{Ax} \cdot \overline{A15} \cdot \overline{A14} \cdot \overline{RAM} \cdot \overline{ROM} \\ &+ \overline{Ax} \cdot \overline{A15} \cdot \overline{A14} \cdot \overline{A13} \cdot \overline{A12} \cdot \overline{RAM} \cdot \overline{ROM} \end{aligned}$$

$$\begin{aligned}
& + \overline{A_x} \cdot \overline{A_{15}} \cdot \overline{A_{14}} \cdot \overline{A_{13}} \cdot \overline{A_{12}} \cdot \overline{RAM} \cdot \overline{ROM} \\
& + \overline{A_x} \cdot \overline{A_{15}} \cdot \overline{A_{14}} \cdot \overline{RAM} \cdot \overline{ROM} \\
& + \overline{A_x} \cdot \overline{A_{14}} \cdot \overline{RAM} \cdot \overline{ROM}
\end{aligned}$$

$$\begin{aligned}
B & = \overline{A_x} \cdot \overline{A_{15}} \cdot \overline{A_{14}} \cdot \overline{A_{13}} \cdot \overline{RAM} \cdot \overline{ROM} \\
& + \overline{A_x} \cdot \overline{A_{15}} \cdot \overline{RAM} \cdot \overline{ROM} \\
& + \overline{A_x} \cdot \overline{A_{15}} \cdot \overline{RAM} \cdot \overline{ROM} \\
& + \overline{A_x} \cdot \overline{A_{15}} \cdot \overline{RAM} \cdot \overline{ROM}
\end{aligned}$$

The fourth output term is asserted over the 2K page at 80000_{16}

$$\overline{IO\&68} = \overline{A_{23}} \cdot \overline{A_{22}} \cdot \overline{A_{21}} \cdot \overline{A_{20}} \cdot \overline{A_{19}} \cdot \overline{A_{18}} \cdot \overline{A_{17}} \cdot \overline{A_{16}} \cdot \overline{A_{15}} \cdot \overline{A_{14}} \cdot \overline{A_{13}} \cdot \overline{A_{12}}$$

IC19 is a 12L6 PAL which is used to encode the low address lines in order that, when decoded by a 74LS138, an even map can be obtained. Three of the outputs drive the A,B and C inputs on the decoder the remaining three are used as a global '6800' I/O device decode, a global '68000' I/O device decode (both available at the J2 connector) and a \overline{VPA} decode to initiate 6800 cycles and Auto-vectoring.

$$\begin{aligned}
 A &= \overline{IO\&68.AS.A11.A10.A9.A8.A7.A6.A5.A4.A3.A2} \\
 &+ \overline{IO\&68.AS.A11.A10.A9.A8.A7.A6.A5.A4.A3.A2} \\
 &+ \overline{IO\&68.AS.A11.A10.A9.A8.A7.A6}
 \end{aligned}$$

$$\begin{aligned}
 B &= \overline{IO\&68.AS.A11.A10.A9.A8.A7.A6.A5.A4.A3} \\
 &+ \overline{IO\&68.AS.A11.A10.A9.A8.A7}
 \end{aligned}$$

$$C = \overline{IO\&68.AS.A11.A10.A9.A8.A7.A6.A5}$$

$$\overline{VPADRV} = \overline{IO\&68.AS.A11}$$

$$\overline{IOPAGE} = \overline{IO\&68.A11}$$

$$\overline{M6800} = \overline{IO\&68.A11}$$

APPENDIX E : The Educational Interface Board Circuit Diagram

The following page shows the complete circuit diagram of the MC68000 educational interface board.

REFERENCES

- [1] Noyce R.N., and Hoff M.E.,jr : "*A history of microprocessor development at Intel*", IEEE Micro, Vol.1, No.1 Feb.1981
- [2] Gupta A., and Toong Hoo-Mind : "*Microprocessors The First Twelve Years*", Proceedings of the IEEE, Vol.71, No.11 nov.1983
- [3] Farrell J.J. : "*Advanced Personal Computers and Their Processors*", Mini/Micro 1983 conference records.
- [4] Fernandez E.B. : "*Comparison and evaluation of 32-bit microprocessors*", Mini/Micro S.E 1984 conference records.
- [5] Strang B., and Woodhams F. : "*Microprocessor training equipment*", microprocessors and microsystems, Vol.4, No.5 June 1980.
- [6] Cosserat D. : "*MicroSim-a new approach to program development*", microprocessors and microsystems, Vol.3, No.2, March 1979, pp.95-98.
- [7] Whitworth I. : "*Teaching microprocessor techniques to nonelectronics engineers*", microprocessors and microsystems, Vol.4, No.5 June 1980.
- [8] Teja E.R. : "*In-circuit emulators aid designers as they move from 8 to 16-bit processors*", EDN, August 4, 1982, pp.65-75.

- [9] Everett C. : "*New 16-bit microprocessor emulators add features, but performance quirks limit usefulness*", EDN, August 9, 1984, pp.93-104.
- [10] Glover J.R.,JR : "*Integrating hardware and software in a computer engineering laboratory*", IEEE Transaction on Education, Vol.E-24, No.1, Feb.1981.
- [11] Lumley R.M. : "*An industrial microcomputer education program*", IEEE Transaction on Education, Vol.E-24, No.1, Feb.1981.
- [12] Holdstock k. : "*An interface between a PDP11/20 and an M6800*", Final year undergraduate project, University of Bath, 1979.
- [13] Whitworth P.F. : "*A Multi-Family Multi-Microprocessor Education and Development System*", PhD thesis, 1983, University of Bath.
- [14] Smith D. : "*32-bit microprocessr chips offer system-like benefits*", EDN, September 19, 1985.
- [15] Osborne A. : "*An introduction to microcomputers volume 2 some real microprocessors*", Osborne & Assoc.,Inc.
- [16] Motorola : "*MC68000 16-Bit Microprocessor User's Manual*", Second Edition Motorola Inc, 1980.
- [17] Winpigler D.J : "*The 32-bit architecture of the M68000 family*", Mini/Micro N.E, 1984 conference records.

- [18] Osborne A. and Kane G.: " *16-Bit Microprocessor Handbook*", Osborne/Mc Graw-Hill, 1981.
- [19] Scanlon L.J. : " *The 68000 Principles and programming*", Howard W.Sams & Co.,Inc.
- [20] King T. & Knight B. " *Programming The M68000*", 1983, Addison-Wesley Publishers Ltd.
- [21] Tanner D.G " *Real-Time Simulation of Power Systems*", PhD Thesis, 1982, University of Bath.
- [22] Williams S.K " *Power System Optimisation and Stability Studies using Real-Time Simulation*", PhD Thesis, 1986, University of Bath.
- [23] Dale L.A " *Real-Time Modelling of Multimachine Power System*", PhD Thesis , 1986, University of Bath.
- [24] Western Digital Corp. : " *Western Digital 1983 Components Handbook*", 1983.
- [25] King T.J. : " *Tripes user guide* ", school of Mathematics, University of Bath, 1983.
- [26] King T.J. : " *Tripes programming guide* ", school of Mathematics, University of Bath, 1983.

- [27] King T.J. : " *Tripos technical guide* ", school of Mathematics, University of Bath, 1983.
- [28] Richards M., Alyward A.R., Bond P., Evans R.D., & Knight B.J. : " *TRIPOS-A Portable Operating System for Mini-computers* " Software-Practice and Experience, Vol. 9, 1979, pp.513-526.
- [29] Bourne S.R. : " *The UNIX system*", International Computer Science Series, 1983.
- [30] Ritchie D. and Thompson K. : " *The UNIX Time-Sharing System*", The Bell System Technical Journal, July-August 1978.
- [31] Thompson K. : " *UNIX Implementation*", The Bell System Technical Journal, July-August 1978.
- [32] Ritchie D.M. : " *The UNIX I/O System*", The Bell System Technical Journal, May 1979.
- [33] Dijkstra E.W. : " *Cooperating Sequential Processes*", in programming languages, ed. F.Genuys, Academic Press, New York, 1968.
- [34] Bach M.J. : " *The design of the UNIX operating system*", Prentice-Hall International, Inc., 1986.
- [35] Bourne S.R. : " *The UNIX Shell*", The Bell System Technical Journal, July-August 1978.

- [36] Naur P.(Ed.) : " *Revised Report on the Algorithmic Language ALGOL 60*", The Computer Journal, 5(1963).
- [37] Barron D.W.et al. : " *The main features of CPL*", The Computer Journal, 6(1963).
- [38] Emery G. : " *BCPL and C*", Blackwell Scientific Publications, 1986.
- [39] Richards M. : " *BCPL the language and its compiler*", Cambridge University Press, 1980.
- [40] Kernighan B. and Ritchie D. : " *The C programming language*", Prentice-Hall Inc., 1978.
- [41] Hoffner Y. and Smith M. F. : " *Communication between two microprocessors through common memory*", microprocessors and microsystems, Vol:6, No.6, July/August 1982.
- [42] Weitzman C. : " *Distributed micro/mimi computer systems*", Prentice-Hall, 1981.
- [43] Deitel H.M : " *An Introduction to Operating Systems*", Addison-Wesley Publishing company, 1984.
- [44] Hudson M. and Hausmann G. : " *A designer guide to virtual memory management*", Electronic Engineering, July 1985, PP.55-68.

- [45] Phillips D. : "*Memory-management varieties suit different application areas*", EDN, September 6, 1984, pp.135-143.
- [46] Mitchell H.J. : "*32-Bit Microprocessors*", Collins Ltd., 1986.
- [47] Gledhill L. : "*Tripes-life after the Amiga*", Electronics & Wireless World, Vol.93, No.1619, September 1987.