INTERFACING THE TMS_320_10

DIGITAL SIGNAL PROCESSOR

Written by : Robert Smith

Co-ordinator : Dr R. Radzyner (U.N.S.W.)

Supervisor : Mr. P. Mallon (D.M.R.)

Department of Electrical Engineering,

School of Electrical Engineering and Computer Science,

University of New South Wales,

Sydney, Australia.

This Thesis is submitted in

partial fulfilment of the requirement for

the Degree of Bachelor of Engineering.

November 1985

ABSTRACT

Being a multiple year project for the Department of Main Roads, this thesis/project will involve the interfacing of a Digital Signal Processing (DSP) Integrated Circuit (I.C.) to a development system. As an introduction to the Texas Instruments TMS_320_10 DSP I.C., the Intel 2920 was briefly looked at.

Most of the time spent was in exploring the TMS_320_10 in its Evaluation Monitor (EVM) environment, the recently industry adopted Versa Module Europa (VME) bus system and the capabilities of the various controllers: the D.M.R's EPSON HX-20 personal computer, and the Eurocard Micro_processor Controller (EMC) card.

From this exploration, the possibility of using the VME bus system was ruled out as there was no immediate access to it. However, the body of this work is centered on the application of the TMS_320_10 as a 'front end' processor to a Motorola development based system with the MC_6803 as the heart.

An interface between the TMS_320_10 and the EMC card was developed, and a simple handshaking stop and wait protocol was implemented. The 'last minute' hardware redesign proved beneficial and due to a time constraint, The software needs a maximum of three hours of patching before the system is fully operational.

ACKNOWLEDGEMENTS

I would like to acknowledge the following people for giving me the incremental direction at the right time throughout the year:

Mr. Philip Mallon, my supervisor at the Department of Mains Roads (D.M.R.) for giving me the opportunity to experience more of the outside world, and supplying the equipment to work with,

Dr. Robert Radzyner, for being my co-ordinator here at the University and having lots of patience, persistence and ideas,

Mr. Tom Millett, for being there to bounce many ideas off and being eager with many suggestions,

Mr. Douglas Randall, for having the 'right I.C. at the right time',

Mr Philip Wolstonecroft, for clearing up my initial problems with the Eurocard Micro_processor Card,

Peter Chubb, having some experience with the Amsterdam Compiler Kit,

Tony Chand, an old friend, offering to do the 'last minute' proof reading, and

My Family, having to put up with me during the course.

CONTENTS

P.T.O

P.T.O.

P.T.O.

P.T.O.

TABLE OF FIGURES

pg 95

1. OVERVIEW

To set the scene, this short preamble will deal with the general aim of the thesis, contain a statement of achievement and discuss prior work done to date.

1.1 Introduction

This was intended to be a multiple year project for the Department of Main Roads New South Wales (D.M.R. N.S.W.). My contribution was to initiate the project of coupling a Digital Signal Processing I.C., the TMS_320_10, to a Versa Module Europa (VME) bus system. As this thesis progressed, several decisions had to be made as to whether to continue exploring the current path or not. Later on, the introductory chapter have a thesis development map.

Essentially, a data logging system interface was needed for use by the Materials Engineering data acquisition systems already in operation. For instance, a driver can be relieved from the task of using a keyboard to start and stop the data logging process by having the system employ speech recognition.

At the same time the system can be sampling the transducer(s) mounted on the vehicle as well as processing video information, and performing according to the driver's verbal commands. Hence, the VME system has been the one chosen to verify if it is capable of fulfilling the requirements.

Based on my discoveries, triumphs and encountered brickwalls, next year's student will be better able to use my design or develop an improved implementation. Partly because of limited documentation and availability of

equipment, several side avenues were explored. Hopefully, my attempts throughout the year will shed some light on the 'taken for granted' ideas and beliefs of system interfacing.

The development and exploratory work covered during the year has made me more aware about questions that need to be asked. In general, any experience gained , whether in the laboratory or elsewhere, is limited by the time available. As in the case of this thesis, many avenues were explored but not in as great a depth as the author would have liked. However, a schedule must be followed and when the exploration must stop, it must stop. Otherwise, successive steps in the schedule are drastically altered.

1.2 Scope of the Thesis

Many areas have been covered: from the hardware to the software and its documentation. Included were, the hardware interface design of several programming languages, and a synthesis of the function of an evaluation monitor from limited documentation. The added incentive was to learn how to use the latest Texas Instruments family of digital signal processing I.C.s.

1.3 Statement of achievement

The objective of an interface between the TMS_320_10 and the Eurocard Micro_processor Controller (EMC) card had been fulfilled. Along with the hardware, software was written according to the simple protocol developed for the two processors that used the interface. However, due to time constraints, the software was not fully debugged, although the hardware was tested from both processors and it functioned as per specification.

Also hands-on-experience' was gained with different systems such as the Evaluation Monitor (EVM), the Epson HX-20 personal computer, and looking into the capability of the Versa Module Europa (VME) bus system.

The exploration of the VME bus system, different buses such as the RS-232C and the IEEE-488 were compared as well.

1.4 Prior Work

Since no prior work had been done, the preliminary guide-lines for this VME/TMS_320_10 interface had to be adapted during the progress of the thesis. Also because not much information for the VME bus system was available at the time.

As implied earlier, interfacing is a unique task. Only ideas can be extracted from other reports and documents. Understanding the devices to be interfaced is usually the first step to 'building the bridge' between them.

Because the 'slave' processor is a DSP I.C., it is presumably faster than the 'master' processor, Its characterstics are best looked at first, as the rest of the system usually depends on the availability and the timing of the processed information.

1.5 Signal Processing

Having a DSP I.C. operating in 'real time', allows the user a greater freedom to concentrate on the complexity (or simplicity) of the algorithm that will be used to process the information. A self adapting ability is even more beneficial to the user. It means the user can develop systems with feedback so its adaption to the environment is possible. This self modification of memory feature is

essentially the Harvard Architecture. See chapter 6 section 2 about this architecture for more information.

If the real time feature is coupled with this Harvard architecture, then the processing is virtually boundless. A prime example using these two features, is a digital modem, shown in figure 1.1, where the output of a quantiser can be fed back into the adaptive equaliser which in turn is fed into the quantiser. The Texas Instruments (TI) TMS_320_10 has at least these two features.

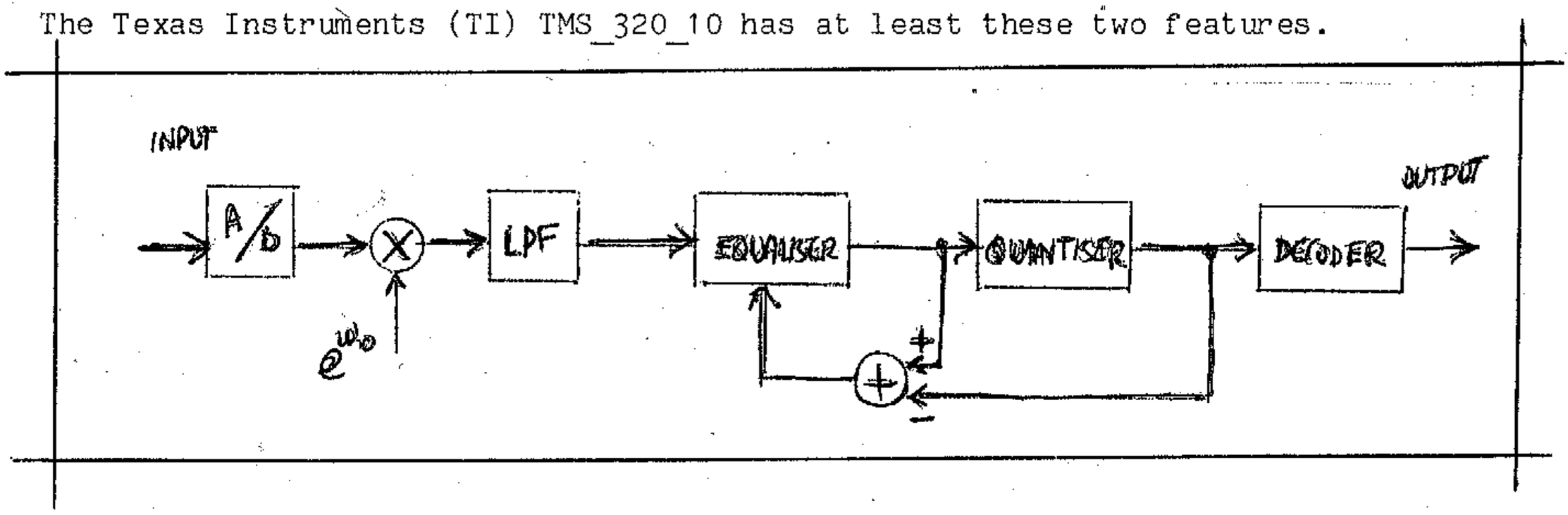


Figure 1.1. Example of a Digital Modem (receiver)

As in the case of the TMS_320_10 DSP I.C., the real time capability and the unique architecture allow some typical applications, such as

a. Signal Processing - digital filters, Fast Fourier Transforms, Speech Processing,
b. Telecommunications - High Speed Modems, Adaptive Equalisers, Data Compression, and even
c. Image Processing - Pattern Recognition, Image Enhancement.

The D.M.R., having purchased the TMS_320_10 and its evaluation monitor, has loaned it to the University for almost 2 years. The TMS_320_10 has been in Australia for about 2 years, and recently, in March 1985, the bigger brother/sister, the TMS_320_20, has been announced.

This new family member has a far greater capability than the younger member. It would have been more useful to the D.M.R. for their applications. But even at this stage, there is little access to development support, as it is mostly unreleased to the public. The TMS_320_20, being an improved version, can be understood from the familiarity of the TMS_320_10. A brief comparison between the TMS_320_10 and the TMS_320_20 is covered in Appendix 2.1. (page 76)

The topics covered in this chapter were the overall aim of this multiple year project, a statement of achievement by the author, coverage of prior work and very useful features and applications of the TMS_320_10. Also an announcement of the TMS_320_20 and its current state were glossed over.

Analogous to the view from a mountain top, it is suggested the reader read through the next chapter, the introduction, to obtain an outline of the work covered in this thesis.

//

2. INTRODUCTION

In this chapter, a brief description of the material contained in each of the following chapters is presented. A map of how the thesis has progressed throughout the year is presented with a short 'walk through discussion' so the reader can interpet it.

An old adage that states "a picture says a thousand words" is relevant to this thesis, so if the reader wishes to scan this report before deciding to read it, it is suggested the following figures are looked at: Figure 2.1 (pg 10), figure 5.2 (pg 23) and figure 9.1 (page 48) are seen.

2.1 Conventions

References to the appendices assume that the chapter number 15 is prefixed to it. For instance, Appendix 4, section 2 must be interpreted as Chapter 15.4.2. Also, to make the reading of certain words easier the '_' charater has been included. For instance, the 'TMS32010' has been changed to 'TMS_320_10' and 'microprocessor' to 'micro_processor'.

2.2 Outline of the rest of this Thesis

As an outline, let us briefly pause to see how the rest of this thesis fits into the overview described in the last chapter. The key chapters are Chapter 3, Chapter 5 and Chapter 8, but the following contains a brief summary of each chapter.

The third chapter discusses the function of the interface, purely on a conceptual level. It mentions how and when the data flows to the master or slave system. Also, a simple protocol is adopted to keep the complexity down to an understandable level for the user and reader.

In chapter 4, general considerations are taken. It shows the 'learning-curve' or the decisions needed to be made in order to develop the interface. Included are the available and unavailable systems explored such as the TMS_320_10 Evaluation Monitor (EVM) module and the Versa Module Europa (VME) bus system. One important point the reader must absorb immediately: EVM is not a mis-spelling of VME.

Chapter 5, has an overview of the system configuration. It shows a system diagram and gives a brief description and function of each building block in the system. That is, how the EVM, Eurocard Micro_processor Controller (EMC) card, interface and Host computer are connected. With the block diagram description, this chapter is placed in context of the following five chapters.

Chapter 6, contains a functional overview and the organisation of the EVM. The isolation of the TMS_320_10 is explained, resulting in the use of dual port RAM. Also the data acquisiton system is described and attempts that were made to reverse assemble the monitor code to gain better control of the TMS_320_10.

The Eurocard Micro_processor Controller card is the subject of chapter 7. Described is the usage and adaptability both processors on either side of this card in terms of hardware and software, the problems encountered with the unit and the temporary solutions.

Here, chapter 8, contains an overview of the interface design for chapter 9 and 10. It mentions the design phase and compatibility of the interface. The hardware chapter, 9, discusses the considerations involved with the TMS_320_10 and the EMC card, the circuit operation and the timing diagrams. For the software chapter, 10, described are both the programs, with a quick explanation to using their flowcharts. Further work that can be carried out ~~is also~~ included.

Chapter 11 deals with the future considerations. In other words, it attempts to predict the path of the project ~~for~~ subsequent years. However, this mainly depends on the use of a VME bus system.

2.3 Thesis Development

Before a 'walk through' example is given of the thesis development diagram shown in figure 2.1 (pg 10), it is best to briefly describe the purpose of the interface. That is, the interface must be able to communicate with the 'front end' signal processor. At this point, it might be best to describe what Digital Signal Processing is to the average person.

2.3.1 Signal Processing

According to the TMS_320_10 User's Guide, "Digital Signal Processing (DSP) is concerned with the representation of signals (and the information that they contain) by a sequence of numbers, and the transformations of processing of such signal representations by numerical computational procedures."

From this definition, a DSP I.C. performs an algorithm on data derived from analogue signals. These may even be acoustic. The algorithm loops repeatedly through the instructions that carry out these calculations on this data, calling it and coefficients up from memory as needed.

2.3.2 Example

Returning to the thesis development diagram, (next page) the 'to and from' arrows indicate the area explored was a 'dead-end'. The single arrow shows the path where several problems were encountered and overcome.

Finally, the dashed line represents the intended connection between the explored areas, but unfortunately were unresolved, for the amount of time allocated. In the interests of consciseness, only the major problems and solutions will be covered. The following paragraphs refer to figure 2.1 (next page).

At the start of the year, the familiarisation of the Intel 2920 DSP I.C. served as a good introduction to the TMS_320_10, covered later in this chapter. A program was written to make sure the concept of sampling was understood. The example 2920 program, due to the volume of this work, and the thesis submission deadline, will be added to an appendix at a later date.

With a limited number of 2920 I.C.s and three students going to use them, a disassembler was written as it was difficult to identify the program in the EPROM section immediately. If further work on the EVM firmware needs to be carried out, the reader may wish to use the idea of the disassembler program, written in Pascal, to his/her advantage. A copy of the disassembler, due to the volume of this work, and the thesis submission deadline, will be added to an

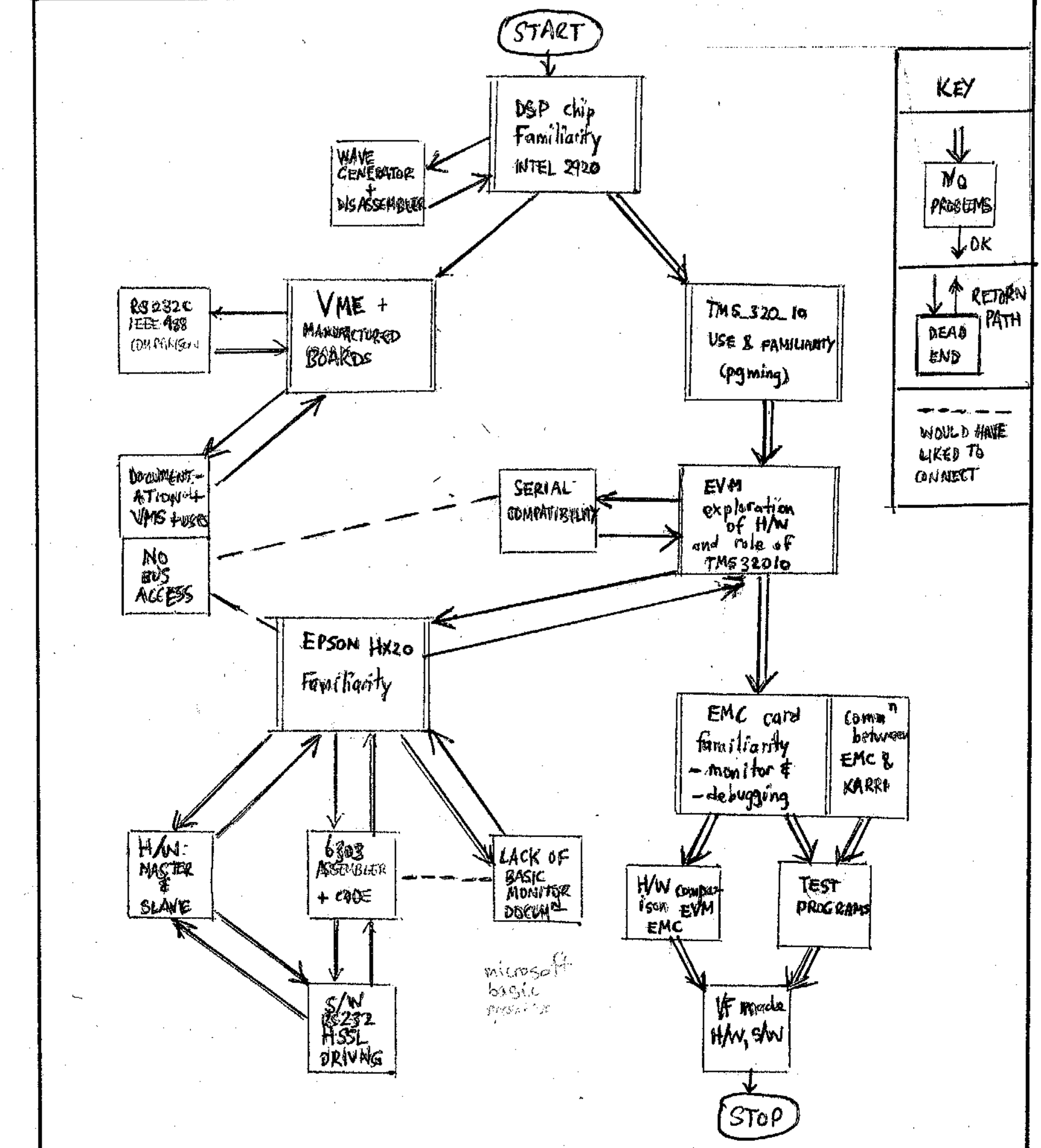


Figure 2.1. Thesis Development

appendix at a later date.

After this familiarity, the two 'processes' were started simultaneously. They were, the exploration of the VME bus system and the function of the TMS_320_10 processor as some ground had been covered on the 2920. The VME system was unable to be explored further as the documentation was limited and cryptic in places. However, the TMS_320_10 was a different device to the 2920, like a next generation DSP I.C. The code of the TMS_320_10 included conditional branching as the 2920 was only able to execute forwardly.

Next, was the exploration of the Evaluation Monitor (EVM). An attempt was made to cause the EVM to print out a string of characters onto the screen by trying to reverse assemble the EPROM source code, as no source listing from Texas Instruments was forthcoming. This exploration resulted in tremendous familiarity of the monitor. The TMS_320_10 was unable to transfer its processed information serially, directly via the monitor.

Another system, the EPSON HX-20 personal computer, was investigated as a controller of the TMS_320_10. Here, the interface was to connect the EVM and the HX-20. Several hours were spent on exploring the HX-20 to see if it was possible, via the monitor, to serially transfer (and receive) information to the TMS_320_10. Once again, a lack of documentation for the BASIC monitor system, even though the High Speed Serial Line (HSSL) was documented well but untested, prevented further exploration.

The next controller chosen was the Eurocard Micro_processor Controller Card (EMC card). This was less complex than the HX-20 but a new assembler monitor system and devices on the card had to be understood. As late in the year as it was, the interface had progressed through several revisions as the capability of the TMS_320_10 and the EMC card were being revealed with experience.

In summary, this chapter contained an outline of the following chapters, a short description of Digital Signal Processing and a short discussion on the development of the thesis. The next chapter will cover the concept of the interface in terms of how the information is transferred and its direction controlled.

//

3. CONCEPT OF INTERFACING

3.1 Introduction

Interfacing between two micro_processors from different families is not as straightforward as from the same family. It is basically, a task of fitting two or more systems to a specification. Here, the two processors are the TMS_320_10 and the MC_6801.

The first, is a specialised micro_processor from Texas Instruments and is suited for fast digital signal processing. It is a 16/32-bit device. The second, is an 8-bit single micro_computer unit from Motorola. This chapter will examine the fundamental ideas for interfacing between two micro_processors.

3.2 Interfacing

Most of the final design is a result of a 'tooling' process. This stage is rarley resolved in just one or two experimental versions. A large amount of thought needs to go into coalescing the hardware and the software so they act as one functional unit.

Consideration must be given to existing systems being utilised and the possible alternatives. Even with a slight modification, the time contraints and available resources must be carefully kept in check.

For the TMS_320_10 to transfer data to the MC_6801, it will need to be held long enough so the MC_6801 can collect it. Conversely, the TMS_320_10 will need to read the data only once. So the first property of this interface is, the bi-

directional nature of the data bus. With this in mind, ultimate control of the data flow is determined by the fastest device, namely the TMS_320_10. Figure 3.1 shows a simple approach to the interface.

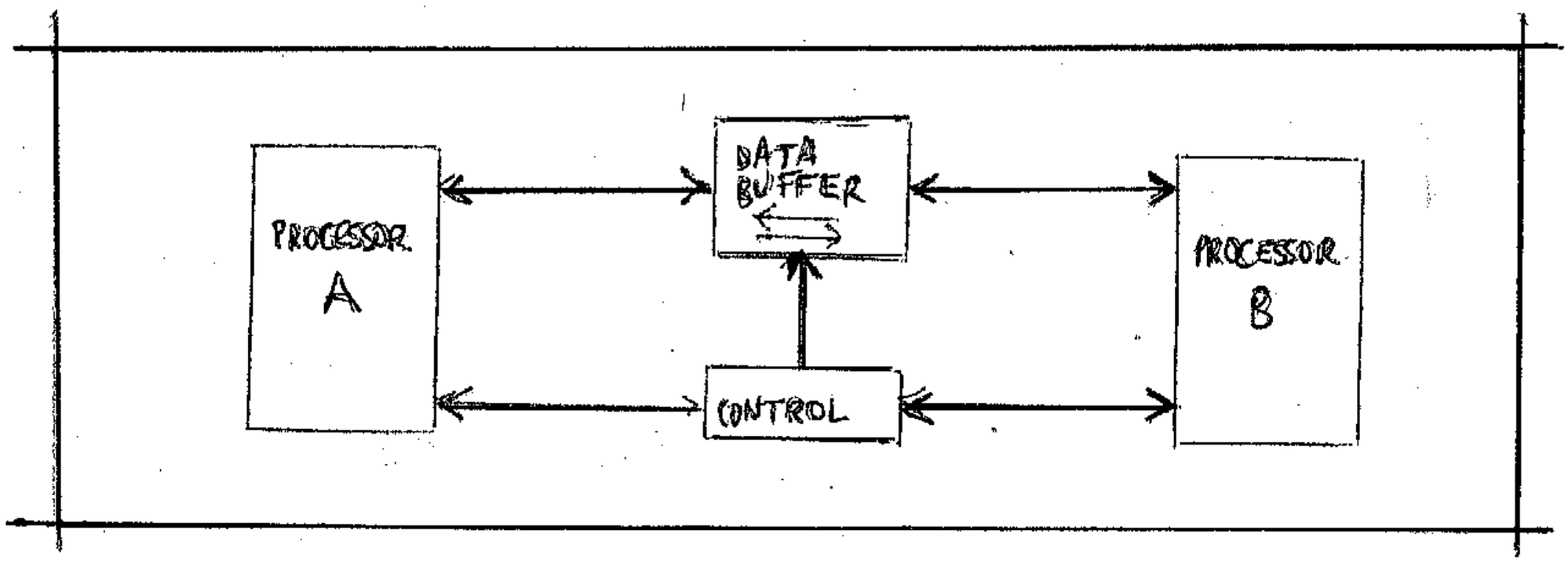


Figure 3.1. Interface

As these two processors are not speed compatible, there must be a buffer or latch or a 'half-way-house' for the data. This is to ensure the data buffer is read or filled without affecting the operation of the other processor. That is, to avoid the conflict of accessing the data bus at the same time.

3.3 Stop and Wait Protocol

From this arrangement, a half duplex stop-and-wait asynchronous protocol [See TANENBAUM] seemed to be appropriate. (See also chapter 4.4 (page 20).) Briefly referring to figure 3.1, processor A sends a message (data) to processor B, it stops and waits for an indication that processor B has received the data correctly. That is, A waits for an acknowledgement (ACK) from processor B. When it receives the ACK, the next message can then be sent. This also applies to the opposite direction, B to A, if both processors have 'agreed' to use the same protocol: set of rules.

The control block in Figure 3.1 sets the correct direction of the data flow. Depending on the specific application, processor B can just load the buffer up with the data and at the same time generate a 'data available' signal for processor A to collect the latched data. This method relieves processor B of trying to synchronise the data to the other processor. All it needs to do is place the data into the buffer and let the receiving end take care of reading in into its memory.

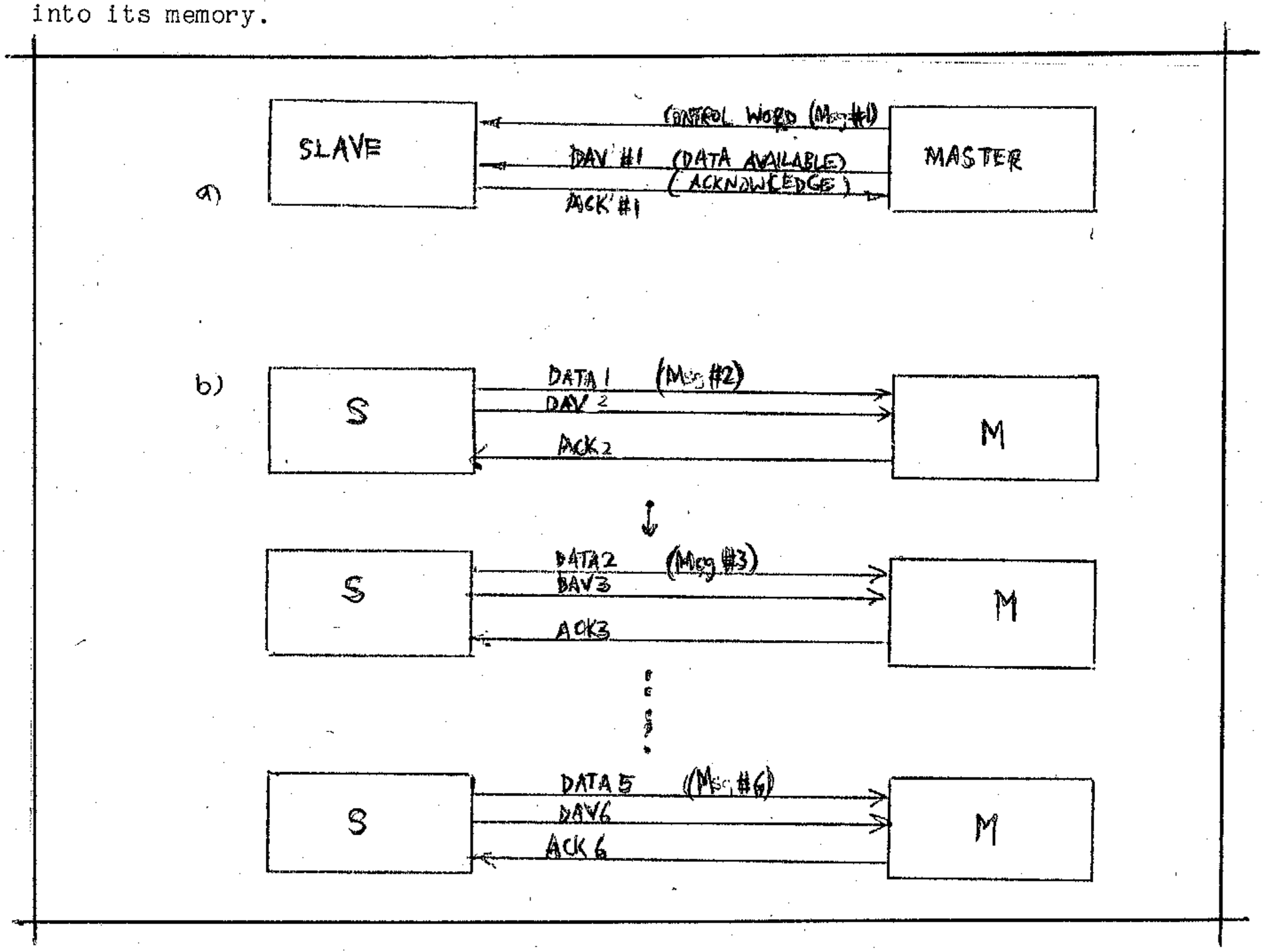


Figure 3.2. Protocol

Figure 3.2 shows how the stop-and-wait protocol can be used between a master and slave processor system. To ensure sensible communication, the master sends a control word containing information such as,

i. the requested buffer size for the about-to-come data,

ii. baud rate of the master, and

iii. other initial parameters to be set [See KERMIT].

In this case, the type of instruction can be a request for a limited number of words to be sent or a command to change specific parameters.

For the 'read' instruction, the slave sends a message (#2) and waits. The master receives message #2 and returns an ACK for message #2. The transmission continues until the session has ended. If serial communication is used for the data link, then in the protocol there must be a way of re-transmitting the same message if it is corrupted by noise during transmission. Here, a negative acknowledgement (NACK) of some form is sent back to the sender. This is just one of the problems associated with transmission over long distances.

What has been covered in this chapter, was the fundamental concept of interfacing, and the use of protocols to establish communication between the complex units. The next chapter, looks at the considerations involved in deciding on the final system that will be used in the time available.

//

4. GENERAL CONSIDERATIONS

In this chapter, many of the reasons why the final system, as presented in the system configuration chapter 5, are outlined. It will include the learning curve of the thesis with a brief discussion of the obstacles and their resolution. Also a final decision about the system will be made.

4.1 Learning Curve

This whole thesis is part of the learning curve. A person must be willing to make mistakes. If not then very little will be gained from the exercise. The following sections outline the many parts of this curve.

4.2 Resource Availability

In designing anything, this must be the first question that is asked: "What is available that will do the job that is specified ?" The next group of sections discuss the major areas shown on the thesis development map in the second chapter, figure 2.1 (page 10).

4.3 Manufactured Boards

Why re-invent the wheel if it has already been synthesised into a compact unit ? Depending on the amount of capital available and the capability of the unit, it is best to obtain as much information as possible about it.

Questions that may be asked are: "Does it do what I want ?", "Does it do too much ?", "Is it too slow in processing the information ?", "Are there other

boards that do almost what I want, but with some modification, will it do exactly what I want ?" The questions can literally 'go on forever'.

As in the case of the SPV-100: a DSP board from Burr Brown, it uses the TMS_320_10 and neatly interfaces to the Versa Module Europa (VME) bus system. The major drawback was partly the price of $4,000 Australian (March 1985), but mostly it was the double height and not a single hieght card. This board also included many extra features that were not relevant for this front end processor.

4.3.1 VME bus system

The industry had recently adopted this VME bus system, but sufficient documentation was unavailable. Even more scarce was the system itself, here at the University. All that could be done was to read and try to comprehend the jargon used in the preliminary manuals.

However, some information was crystallized and can be found in the seventh appendix A (page 87). It also contains the comparison of this bus system to the traditional RS-232 and IEEE-488 buses. At this point in time (see figure 2.1) the exploration was ruled out, as the practical objective of interfacing the TMS_320_10 to a controller was kept in mind.

4.3.2 Evaluation Monitor

Exploring the monitor to see how the TMS_320_10 was controlled seemed like a 'tall order'. This was the alternative to designing a board with the TMS_320_10 on it, its own RAM, EPROM and other standard integrated circuits.

The only connection the TMS_320_10 had to the serial RS-232C line was through the on board controlling micro_processor - the TMS_9995. So it was decided to try to find a section of code in the monitor that can print a string of characters onto its screen. Thus enabling the TMS_320_10 to output some significant 'index of performance'. Chapter 6 (pg 26) contains more detail of the EVM.

Eventually, this decision was 'aborted' as the time allocated for it expired. However, with the lack of documentation, a memory map of the EVM was collated from the current information. Due to the volume of this work, and the thesis submission deadline, this material will be added to an appendix at a later date.

4.3.3 EPSON HX-20

Referring to figure 2.1 again, the task here was to use the High Speed Serial Line (HSSL), to transfer information to and from the TMS_320_10 via an interface. This line was initially designed by EPSON to drive a floppy disk. In the EPSON HX-20 Technical Reference Manual, it stated the HSSL was an option, but it had some untested software to drive the line.

On the next higher concept level, the BASIC monitor, in the HX-20, was to control this high speed interface. This was where the documentation had a 'big gap in it'. There was no mention of how the monitor was able to access the high speed link. Stumbling around the monitor code is not a recommended procedure if no BASIC source code listing was available. Once again, time ran out.

4.3.4 Eurocard Micro processor Controller card

P.T.O.

This was the general purpose card the DMR had developed and was based on Motorola integrated circuits. An offer was made to use their dual processor QUASAR system at the DMR workshop, but it was inconvenient with respect to time, having to book at least one day ahead each time it was needed. Not conducive to development work.

With time 'rapidly' decreasing, this was the system chosen to be the master and controller of the TMS_320_10. The card was electronically fexible enough for it to be configured as a 'local controller'. See the next chapter as to how it was adapted.

4.4 Half Duplex Communication

The choice of the half duplex over the full duplex protocol was relatively straightforward. Figure 4.1 (next page) shows the differences between them. Apart from complexity, there was no need to use the full duplex, as the half will suffice. The protocol in mind, was to have the master change specific paramters in the slave system but not to have the slave sending and unnecessary information.

In summary, there were several considerations involyed in the final design of the interfacing system. They were, the availability of existing systems and their documentation, the ease with which they can be implemented, and the view of designing from basics. However, the next chapter, explains the system that was decided from these considerations.

P.T.O. for figure 4.1

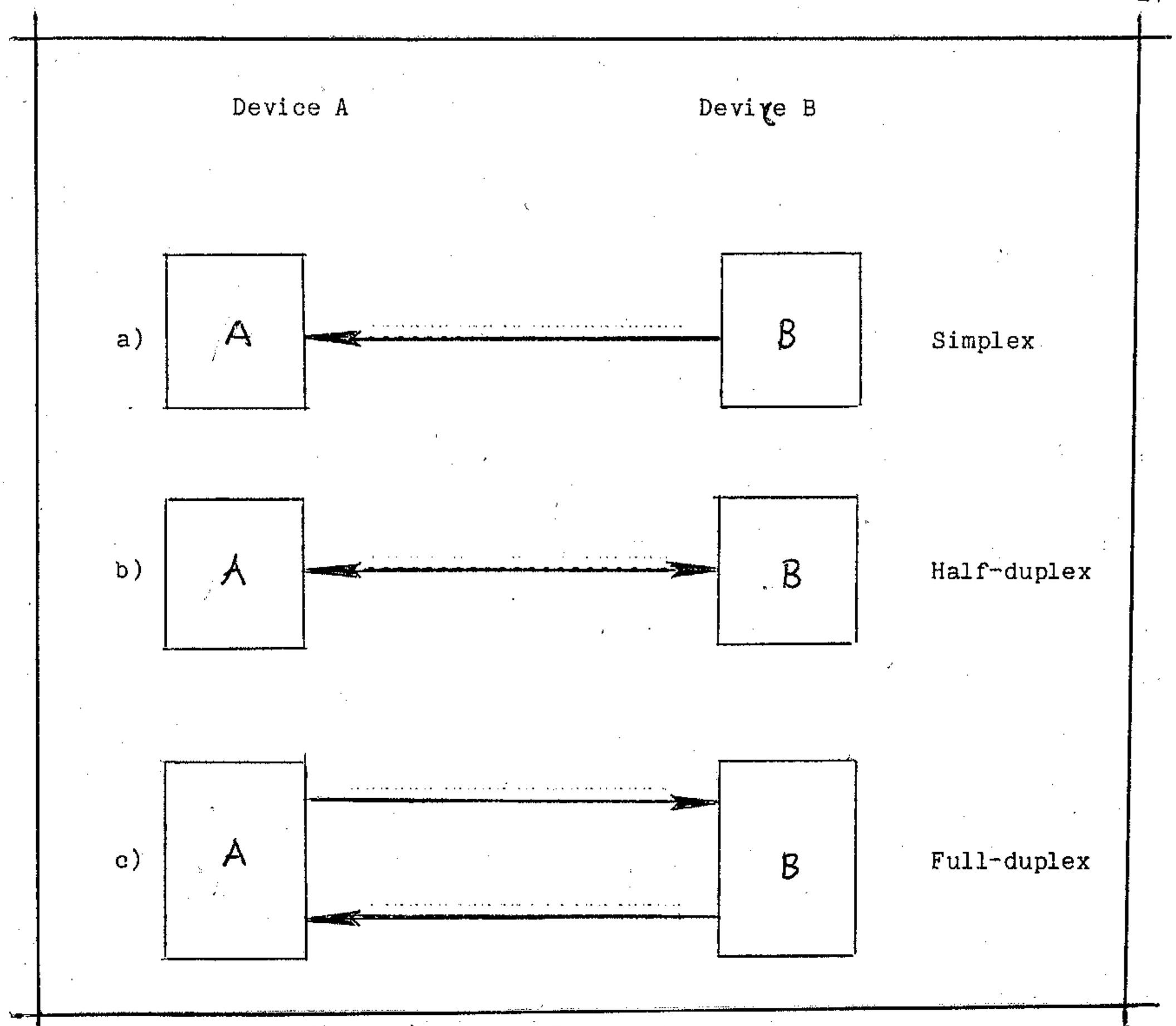


Figure 4.1. n - Duplex Communication

5. SYSTEM CONFIGURATION

5.1 Introduction

As mentioned in the introductory chapter, the reader may altogether bypass this and the next two chapters. However, this chapter will discuss the building blocks of the decided system, in terms of the interface, the front end processor and the local controller. It will briefly describe the master and slave devices where each unit is a development system in its own right. Figure 5.1 shows how this and the next five chapters are linked together.

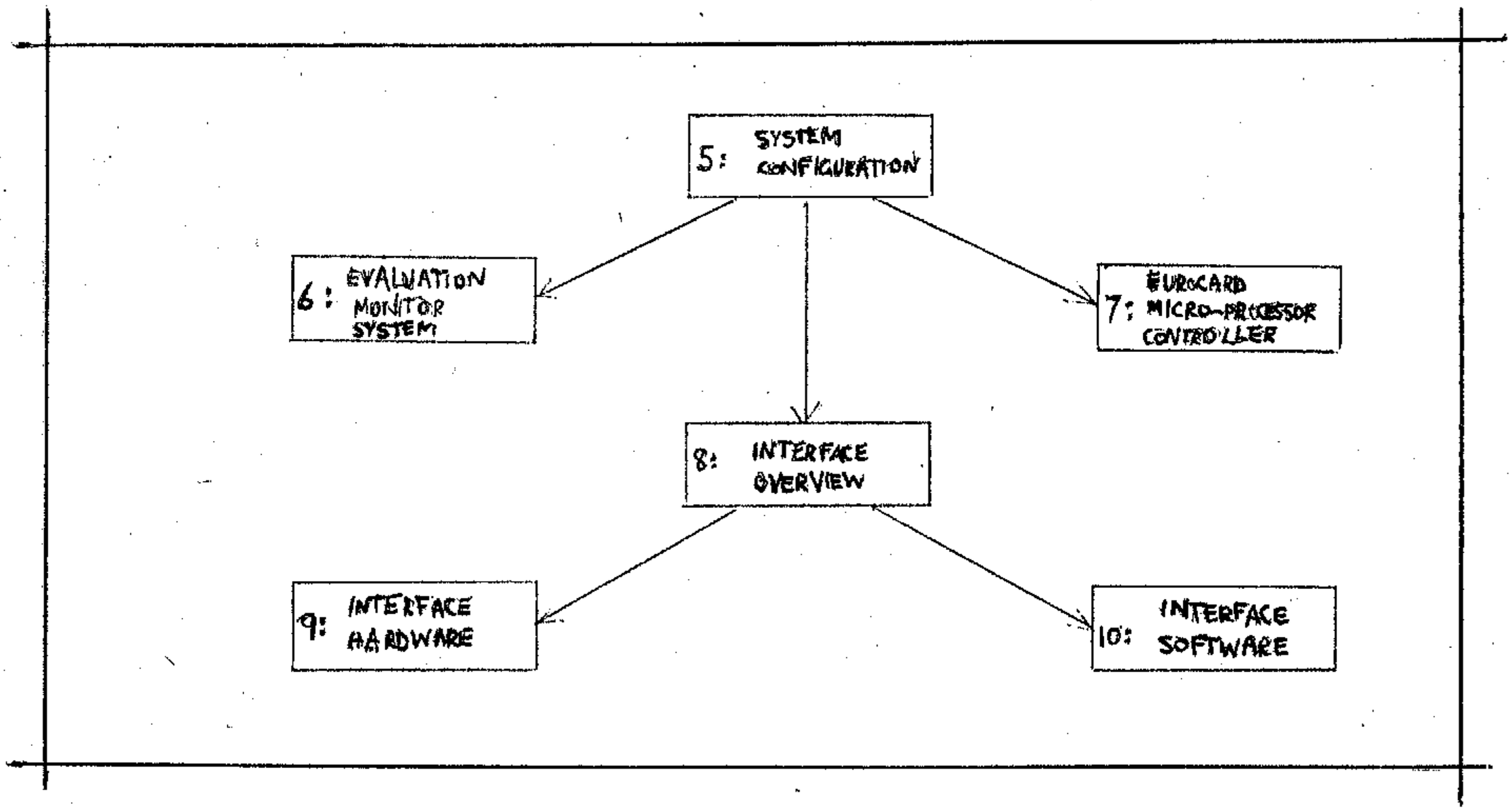


Figure 5.1. Links bewteen chapter 5 to 10

The following figure, 5.2, shows a block diagram of how the respective master and slave is linked together via the interface. Note, for the sake of clarity, the Visual Display Units (VDUs) have been resolved to exist on each device. When, in the laboratory, they are the same unit: separated by a switch.

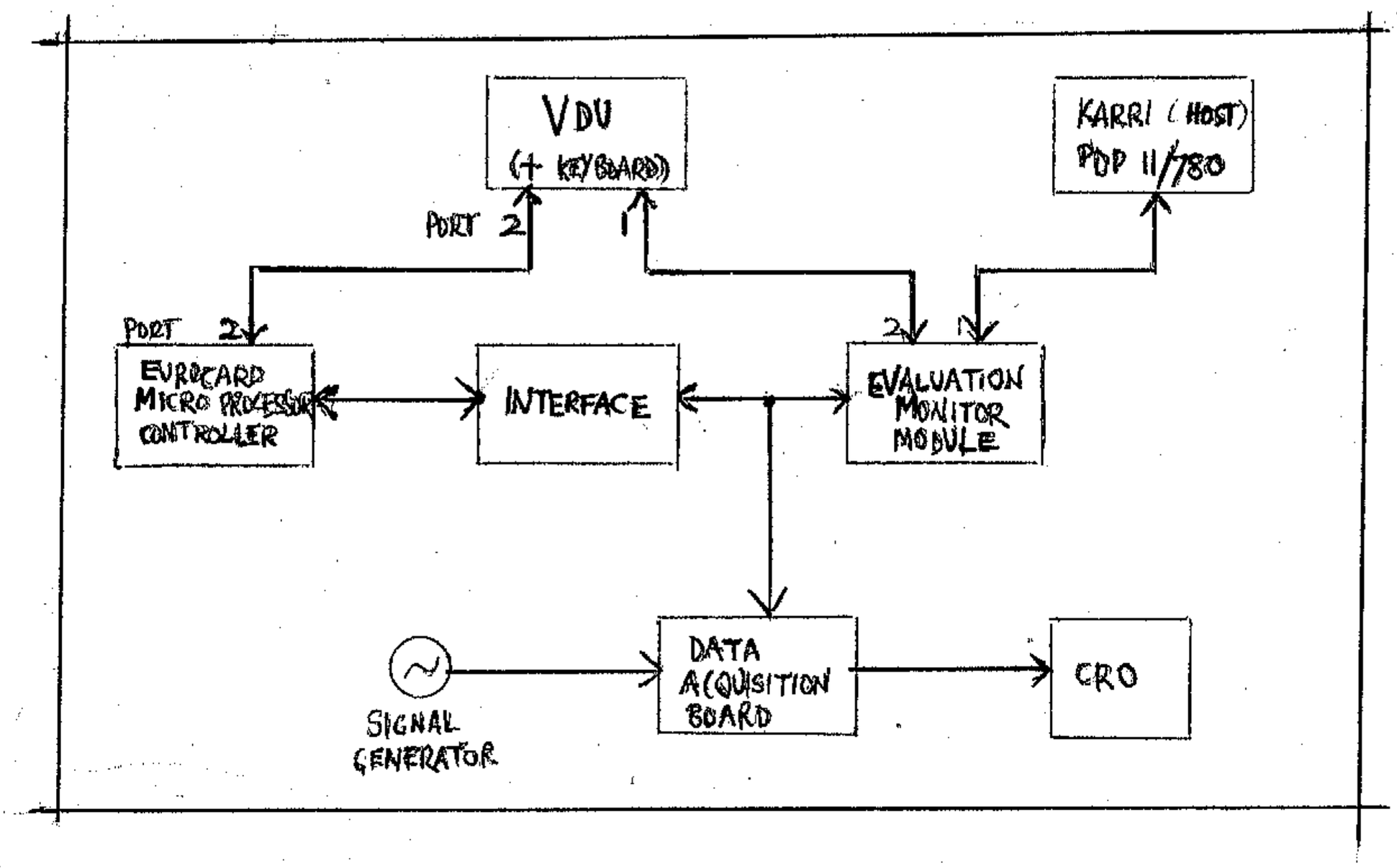


Figure 5.2. System Block Diagram

Firstly, both the EVM and the EM_550 controller (EMC card) are connected to the VDU by an RS-232C cable. The interface was configured to have 16 parallel data and several control lines. As this card is a local contoller, the effects of group delay on each of the data and control lines will be negligible. An alternative was to serially upload the 16 parallel bits of data from the TMS_320_10 into the memory of the Host computer via the EMC card. This alternative was not chosen as full duplex serial communication was not possible with the PDP-11/780.

It was a limit set by the Unix Operating system, allowing only half duplex communication. Also it was verified by having the EPSON HX-20 attempt to upload a program from the BASIC monitor system. For more detail on the interface, see chapter 8. In this configuration (Figure 5.2), the EVM with the TMS_320_10 in it, is the front end processor, and the EMC card is the local or master controller. The idea was to simulate a closed or local system in a vehicle.

5.2 EVM : The Slave

Using the Evaluation Monitor, many useful key features can be implemented. Features such as the number conversion from hex_a_decimal to decimal and back, and the displaying of memory, both program and data. One of its roles was to inspect and/or change the operating data code for the TMS_320_10. As the system is configured, the interface accesses the TMS_320_10 directly, thereby bypassing most of the EVM.

Within the EVM chapter, chapter 6, a description will be presented of the input and output interface to the TMS_320_10 I.C. itself, namely, the the data acquisition board. Note, the digital-to-analogue (D/A) and the analogue-to-digital (A/D) converter is tapped off port 5. (See figure 5.2) This provides the interface can access the A/D and D/A samples if necessary.

What makes the EVM a powerful unit, is the fact that it has an on-board assembler and EPROM programmer. There is also a transparency mode, where the user can directly communicate with the host computer. This facilitates easy assembly of code while downloading into the EVM. See appendix 1.2, for a description of the downloading procedure. An added feature is the master and slave capability by connecting another EVM and designating it as the master or the slave, as the case may be.

5.3 EMC card (EM 550) : The Master

With a power supply, this little 17 cm by 10 cm general purpose card can be set up as a stand alone system. It has a Motorola MC_6803 as the controlling device. The user is not limited to just this I.C. As example, the MC_68701, an

EPROM version of the MC_6801 can be placed into the I.C. socket.

This card also has 2 PIAs, for parallel transfer and an ACIA for the serial transfer of information. During the development, the program will be executed in the RAM I.C. on the card. Included on the card is a monitor to execute and display code like the EVM but it is not as sophisticated: 2-Kbytes versus 32-Kbytes. It's function is to be the master of the data transfer and send the control instruction to the TMS_320_10.

Using the transparency mode of the EVM, it is possible to assemble the code on the PDP-11/780 (the "karri") and download the object code into the EMC card. See the appendix 1.1 for downloading into the EMC card. Note, the second port on the VDU (serge type) only 'listens' to the data stream racing through port 1. That is, there is no handshaking between the second port and the 'outside world'.

The purpose of the interface is to hold the data until the receiving processor is able to collect it and indicate to the sender it has collected it. Once both programs are in the memory of the master and slave, the interface will function.

The next two chapters, we examine the respective master and slave units. They both have their own monitors to verify memory locations and execute code. Some light will also be shed on the difficulties encountered in using them. For instance, it will include the initial assertions held by the author and the true assumptions to make after the system has been ascertained.

//

6. EVALUATION MONITOR SYSTEM

As stated earlier in chapter 4, the monitor is indeed powerful. But at the same time, it is quite limited. Here, it makes sense to state: if the user wishes to simply run it under the guise of what it was meant for, then no problems would be experienced. However, anything slightly more or different to be demanded from it, then be prepared to find out the 'hows' and the 'whys'.

In this chapter, the Evaluation Monitor Module will be explored in the light of interfacing it to another system. This exploration is made difficult when there is a limited amount of documentation is available. This chapter will describe the front end processor for the system described in the last chapter.

It will show how the TMS_320_10 is related to the hardware and software of the whole EVM board. With this evaluation module, the function of the existing data acquisition board will be briefly mentioned. But first, the function of the EVM must be clarified.

6.1 Functional Overview

The following is an extract from the EVM User's Guide, page 2, as the author was unable to express it clearer.

"The TMS_320_10 EVM is a single-board development system for the TMS_320_10 signal processor. The EVM can stand alone as a development system, using the on-board full feature text editor for the creation of TMS_320_10 assembly language text files, and the audio cassette tape interface as a mass storage media.

Or, the EVM can accept text files from a host central processing unit (CPU) through one of the two EIA ports. In either situation the resident assembler will convert the incoming text into executable code in just one pass by automatically resolving labels after the first pass is completed. This object code is stored in a 4K 16-bit-word memory space allowing the utilization of the entire TMS_320_10 address space for developing programs."

The features of the EVM include number conversion routines, to and from hex_a_decimal and decimal, transparency to the host, an upload and download capability and a line by line assembler. See appendix 4 for datail on the range of options. Essentially, the EVM is the link between the TMS_320_10 and a "higher order" machine.

6.2 TMS 320 10 Itself

This is a specialised micro_computer I.C. that has all the standard data, address, reset, and interrupt lines of a conventional micro_computer. What makes it special are the extra features it supports. It can perform multiple operations in a single instruction.

For instance, the 'LTD' moves the contents of the data memory into the T register, adds to the accumulator the contents of the product register and increments the data memory address by one - setting it up to the next location in data memory. See the architecture in appendix 2.3 (p. 77) for the location of the data memory, T and product registers and the accumulator.

Pipelining these intructions makes possible the use of the modified Harvard architecture. A common form of program execution requires sequential access of

for commands and data. The Harvard II architecture, as used here in the TMS_320_10, makes concurrent use of both and program memory over independent buses. Along with this, a common address bus is used to access both sections of memory [see KRAFT & TOY].

However, the Princeton machine structure has both instructions and data in main memory and here, the single bus communicates between the CPU and the main memory. This structure gives a lower cost and programming versatility compared to the Harvard II's speed and its highly specialised instruction set.

Another very useful feature of the TMS_320_10 is the BIO pin. This is an input pin supporting test and jump operations that allows the user externally synchronise this DSP I.C.. It can be used as interrupt pin when performing time critical loops.

6.3 Organistion

The next two sections will attempt to show how the EVM system is orgranised in terms of hardware and software. It will indicate how the EVM was 'evaluated' for use as a major part of the front end processor. The purpose of using the EVM was to see if a serial port was accessible from the TMS_320_10 so that its data could be transferred through it. Nowhere in the EVM User's Guide does it state simply, how the TMS_320_10 is connected and controlled by the TMS_9995, the on board controller.

6.3.1 Hardware

P.T.O.

The hardware for the EVM system is the EVM development board and the data acquisition board - an important unit.

6.3.1.1 Isolation of the TMS 320 10

Looking at both figures, 6.1 and 6.2, the dual port RAM is the link between the TMS_9995, a 16-bit micro_computer, and the TMS_320_10. It is so called because the RAM appears to have two ports, each controlled by the TMS_9995 and the TMS_320_10. These two figures have been synthesised from the circuit diagram, shown in appendix 10.1 (pg 93) and are not to be found in any of the currently available documentation.

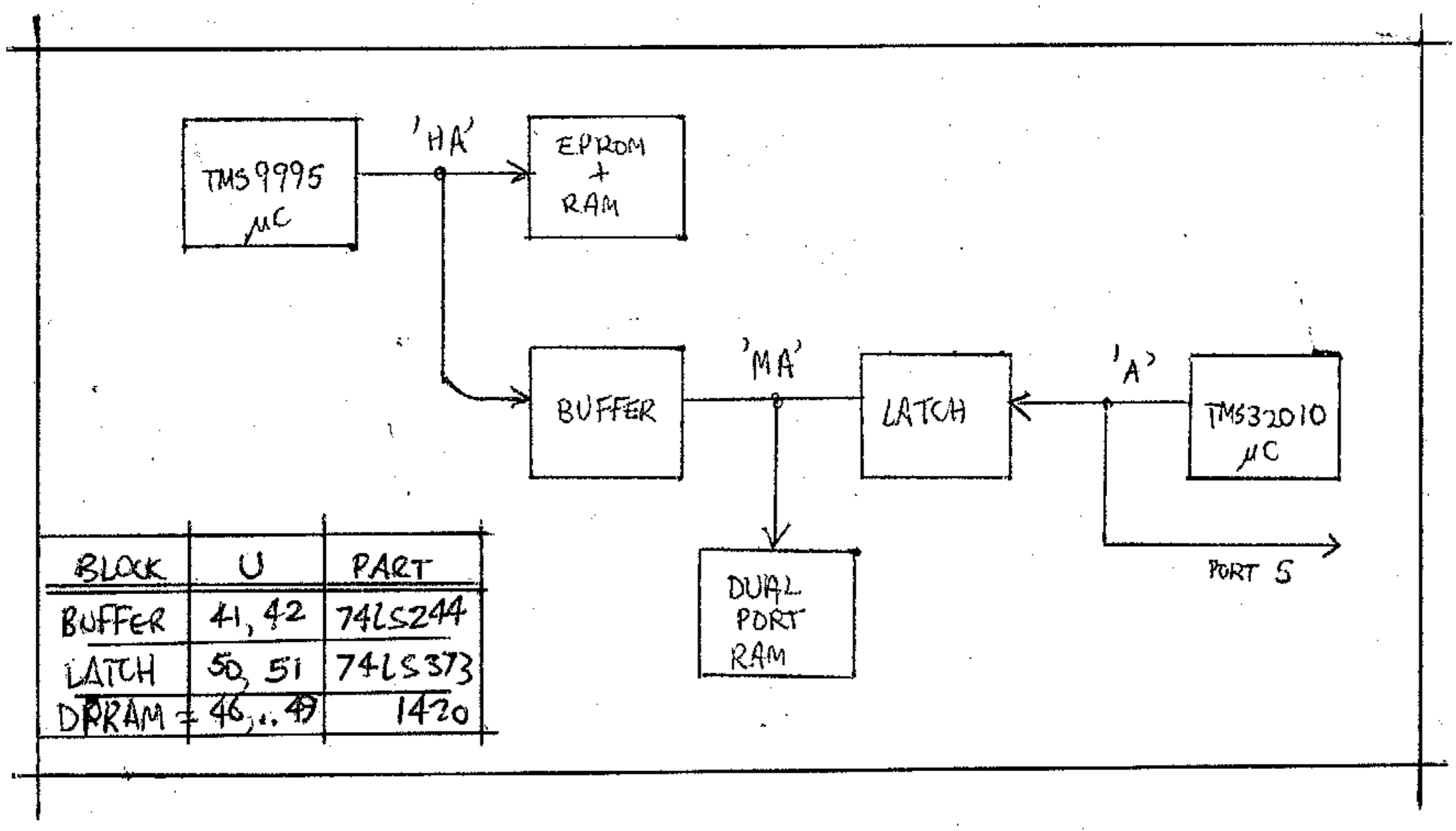
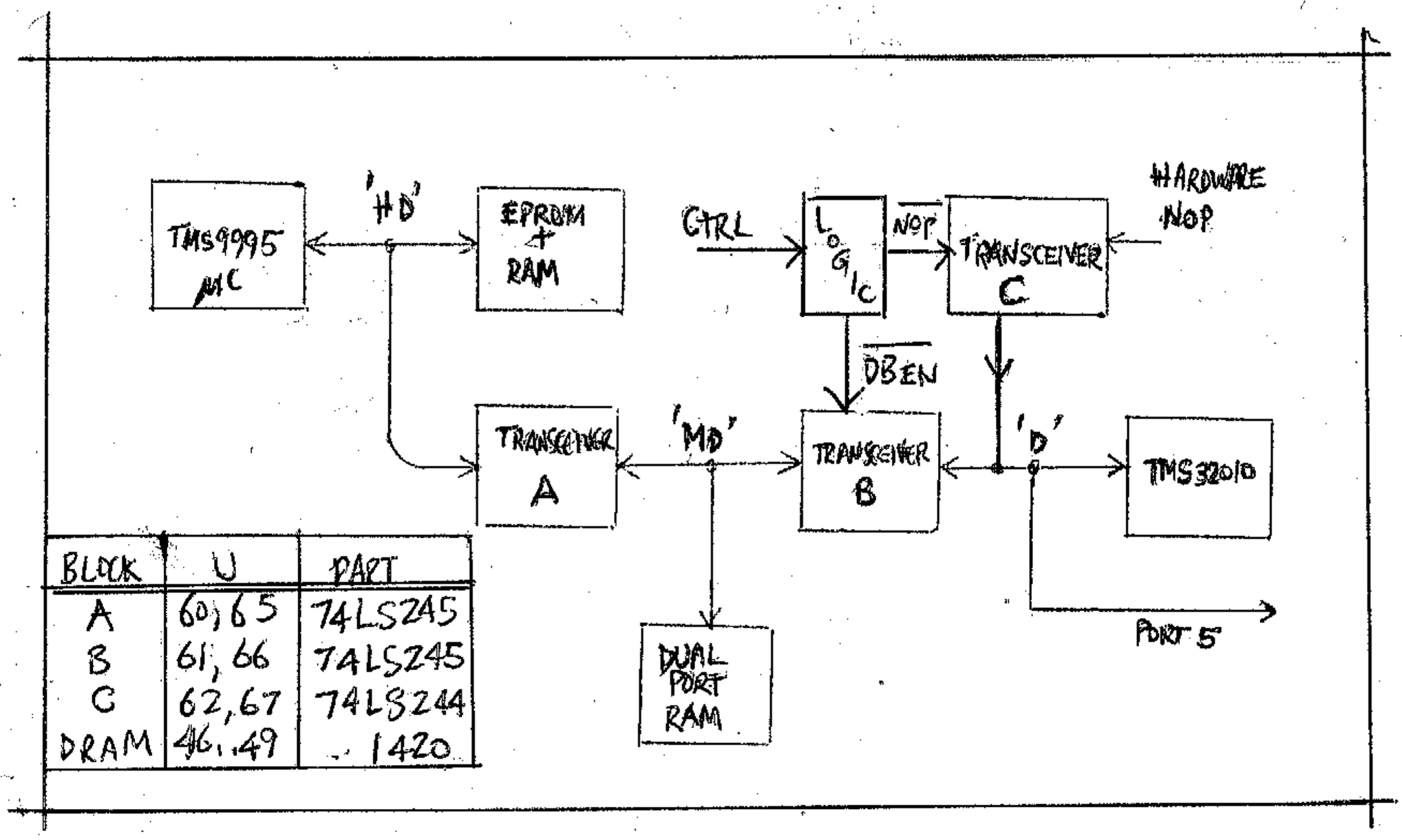


Figure 6.1. Address Lines of the EVM

The two figures have been separated into an 'address line' map and a 'data line' map for clarity. Note, the control lines of the latches, buffers and transceivers have been omitted as the resident controlling micro_computer, the



BLOCK	U	PART
A	60, 65	74LS245
B	61, 66	74LS245
C	62, 67	74LS244
DRAM	46..49	1420

Figure 6.2. Data Lines of the EVM

TMS_9995, has control over them. See the circuit diagram for verification. Now, the reader can see how little control the TMS 320 10 has over any of the hardware on the EVM. It is the TMS_9995 that dictates how, when and where the TMS_320_10 executes its instructions.

When the TMS_9995 is assembling the TMS_320_10 source code, the TMS_320_10 itself is executing the 'NOP' instruction through transciever C in figure 6.2. Note, transceiver B is disabled as B and C are complementarily enabled by the TMS_9995. It is evident, the TMS_320_10 has no access to the up/down loading of the code. Hence, no access to the RS-232C serial line.

6.3.1.2 Data Acquisition Board

P.t.o.

A data acquisition board is a vital component in a Digital Signal Processing system as this. The next two figures, 6.3 and 6.4, show the block diagram of the data acquisition boards designed by Tom Millett here in the laboratory. These two block diagrams have been transferred from Peet Kerjan's thesis, as it cannot be drawn another way without distorting the function of the circuit.

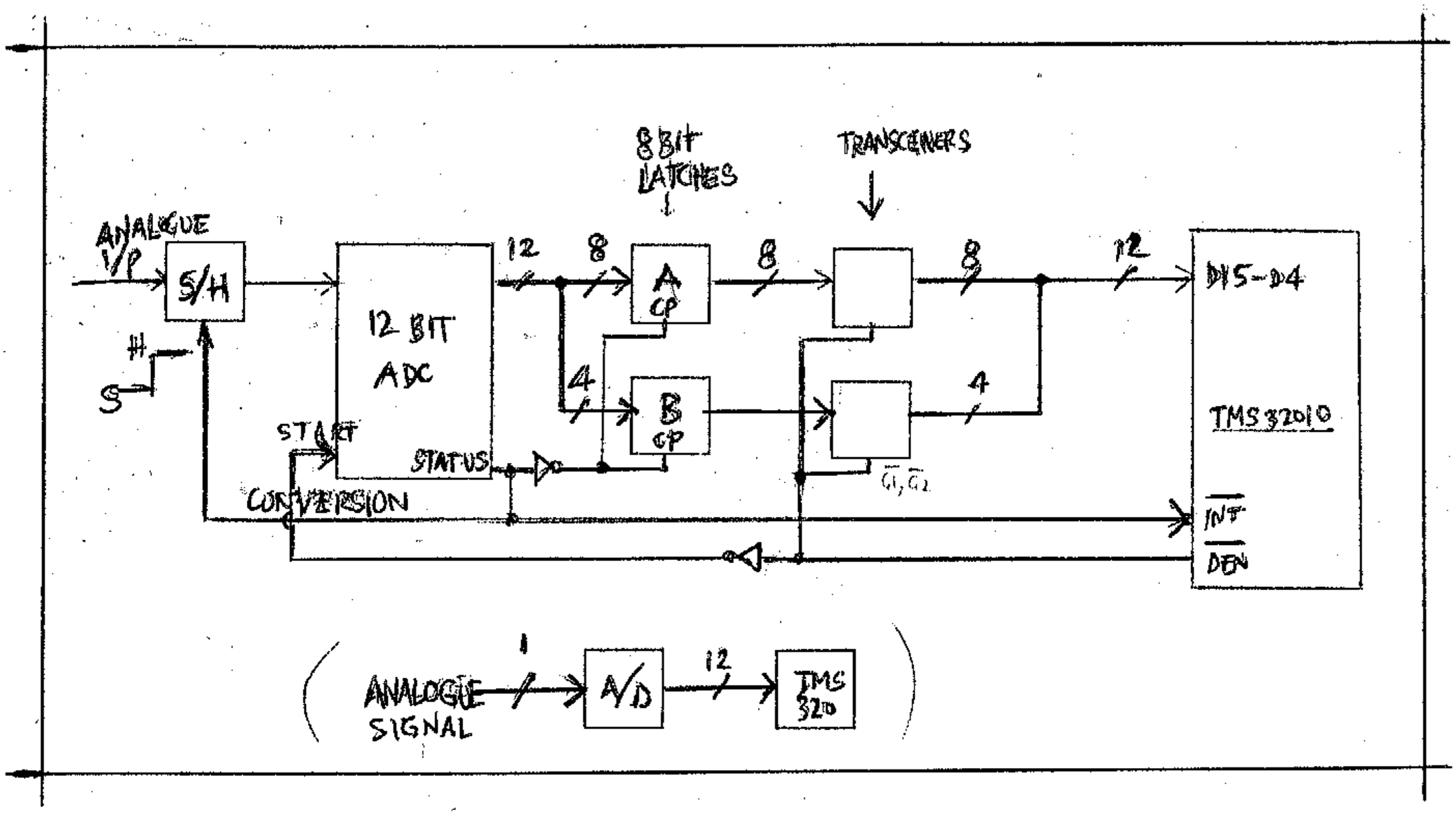


Figure 6.3. A/D conversion

They both use a 12-bit digital-to-analogue (D/A) and analogue-to-digital (A/D) converter.

Basically, for the A/D conversion, the analogue signal is sampled and held for a predetermined time, converted to a 12-bit parallel digital bits and latched into latches A and B. As the conversion just finishes, the TMS_320_10 is interrupted and somewhere in the interrupt service routine an input instruction is executed, causing transceivers A and B to channel the information through to the TMS_320_10 data bus and hence, into the data memory.

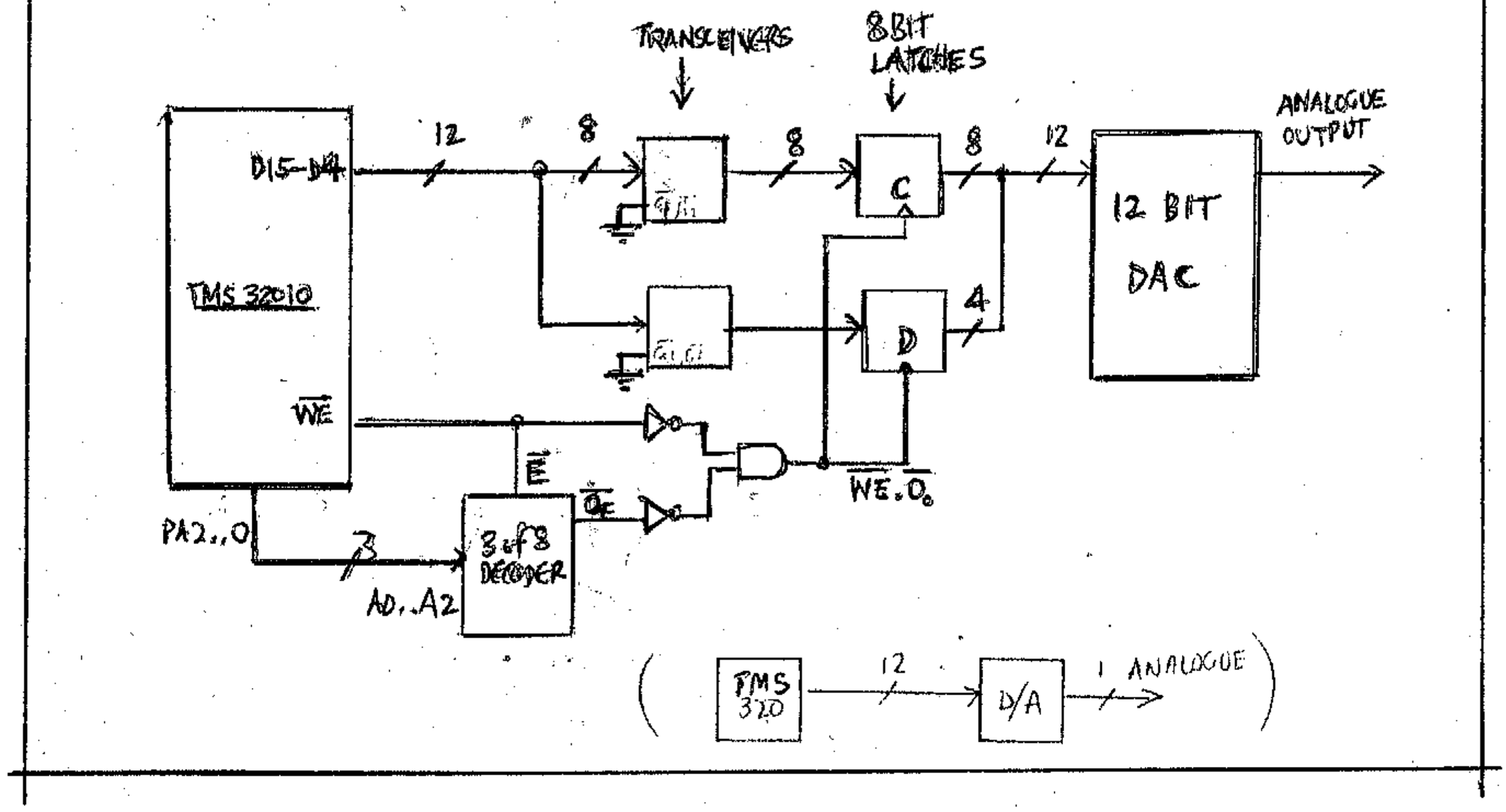


Figure 6.4. D/A conversion

Conversely, for figure 6.4, the D/A converter is fed the correct 12 data bits when the correct output port (hence address) is selected, and latched into latches C and D. Here, the 12 D/A converter reads the latch outputs and converts the bits into an analogue signal.

A sketch of the front panel layout showing the input and output to the data acquisition board is shown in appendix 4.1 (pg 81). The next section will discuss the difficulties, encountered by the author, with the associated EVM software.

6.3.2 Software

With a view as to how the TMS_320_10 was the isolated slave of the EVM board, a step was taken to acquire the source code for the resident controller.

6.3.2.1 Reverse Assembly

This was attempted as the listing of the EPROM, hence the operating system, was not immediately forthcoming. The aim was to discover a routine that controlled the running of the TMS_320_10 so at the end of the TMS_320_10 program execution cycle, a string can be printed out onto the VDU screen. The monitor had provided several functions of locating bytes and words in memory, but the author had to be well acquainted with the operation (op) codes.

If the section of code was found, the EPROM must be re-burnt, after the new code was written and tested. It will mean, the writing of an assembler to test the new code for correctness. This was a large problem, as mentioned earlier in chapter 4 section 3.2, beacuse the School's micro_processor equipment is based on the Motorola family, not Texas Instruments.

However, some code was found, but it was decided to abort the searching due to the time schedule. Some of the results obtained from the exploration, due to the volume of this work, and the thesis submission deadline, will be added to an appendix at a later date.

6.3.2.2 Modifications

With the system here in the laboratory, the start, stop and parity bits were changed so the RS-232C serial code format will match. From the EVM User's Guide, page 2-14 indicates how this can be done. The current settings are: 7 data, 1 start, 1 stop and 1 parity bit, to give the hex_a_decimal number 'E2E2' at address 8E and 8F on the EVM memory map.

The on board EPROM programmer was used to make these changes so the TMS_9902 Asynchronous Communications Controller (ACC) can send and collect the proper bit stream to the TMS_9995. See figure 6.5.

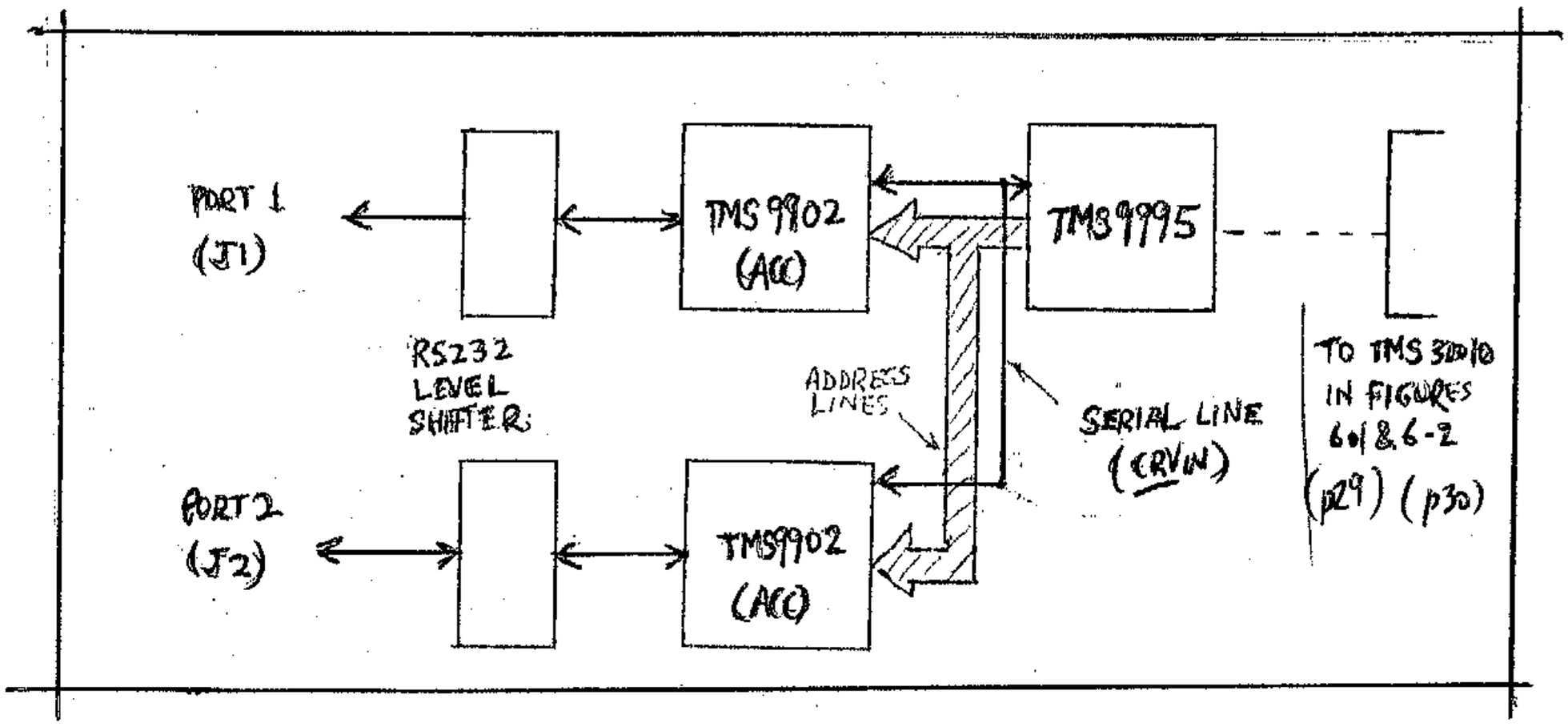


Figure 6.5. EVM RS-232 Communication

To assemble the TMS_320_10 source program, some hurdles had to be overcome to ensure correct program assembly. For instance, the clock initialisation, resetting and assembly of the code are detailed in appendix 1.2.(372).

6.4 Hangups

At this late stage, the following problems were still unresolved.

RUN/EX — Breakpoints were somehow turned off (ignored) when the 'RUN' command was used. However, the 'EX' command appeared to perform correctly. See appendix 1.2.1.

sacl *+,AR1 — This was the strangest one of all. Even though the syntax was legal, the line by line assembler was unable to resolve it. The instruction must be written

```
sacl    *+
larp    AR1
```

intead of "sacl *+,AR1". See appendix 1.2.1 with the test program "sacl.test.t".

PTo

Out/Tblw — As only the lower 3 address bits are decoded, the interface will 'malfunction' if a specific 'table write' (TLBW) instruction is used in the program. This also causes the 'write enable' (WE) to go LOW. On an 'out' instruction, the top 9 (most significant) bits are zeroed, but on the TBLW, the destination program memory address is put on the address bus. This means that if a 'TBLW 63' is executed, the interface will see the WE* line go LOW and see a valid port has been selected. Note, bits 2, 1 and 0 are HIGH (63 decimal is 3F in hex_a_decimal), generating port 7.

In this chapter, a functional overview and the organisation of the EVM module was shown. This included the dual port RAM being the common link between the TMS_9995 and the TMS_320_10. Also the possibility of reverse assembling the EVM firmware, to gain better control of the EVM, was covered but aborted due to the 'water being too deep'.

A significant unit in this EVM system, is the data acquisition board. Without it, it cannot function at all. Consequently, the operation of this board was glossed over as it was not part of the design in this thesis. The next chapter will mention the hardware and software associated with the master and the EMC card of the system. Its usage and problems will also be described.

//

7. EUROCARD MICRO PROCESSOR CONTROLLER

As referred from the chapter on the System Configuration, chapter 5, this chapter will enlighten the reader to the usage of this general purpose card. It will deal with the adaptability, feedback to the user and the information transferring features. Also the problems encountered and their temporary solutions will be mentioned. In addition, both the hardware and the software associated with this unit will be described.

7.1 Introduction

Since this is a commonly used piece of equipment at the D.M.R.'s workshop, it has been used as the master of the system that was described in chapter 5. The way it has been designed, makes it very adaptable to virtually any need that can be thought of, within reason of course.

7.2 Hardware

This is an 8-bit single I.C. micro_computer unit that can operate in a multiplexed mode with RAM. A Motorola MC_6803 is the heart of this card. It is adaptable so that a 2716 EPROM can be in the place of the 6116 RAM. It has a 64 pin Eurocard edge connector, room for two Periperal Interface Adapters (PIAs) and an Asynchronous Communication Interface Adapter (ACIA). The block diagram of this card is shown in figure 7.1.(pg87).

7.2.1 Advantages

P.T.O.

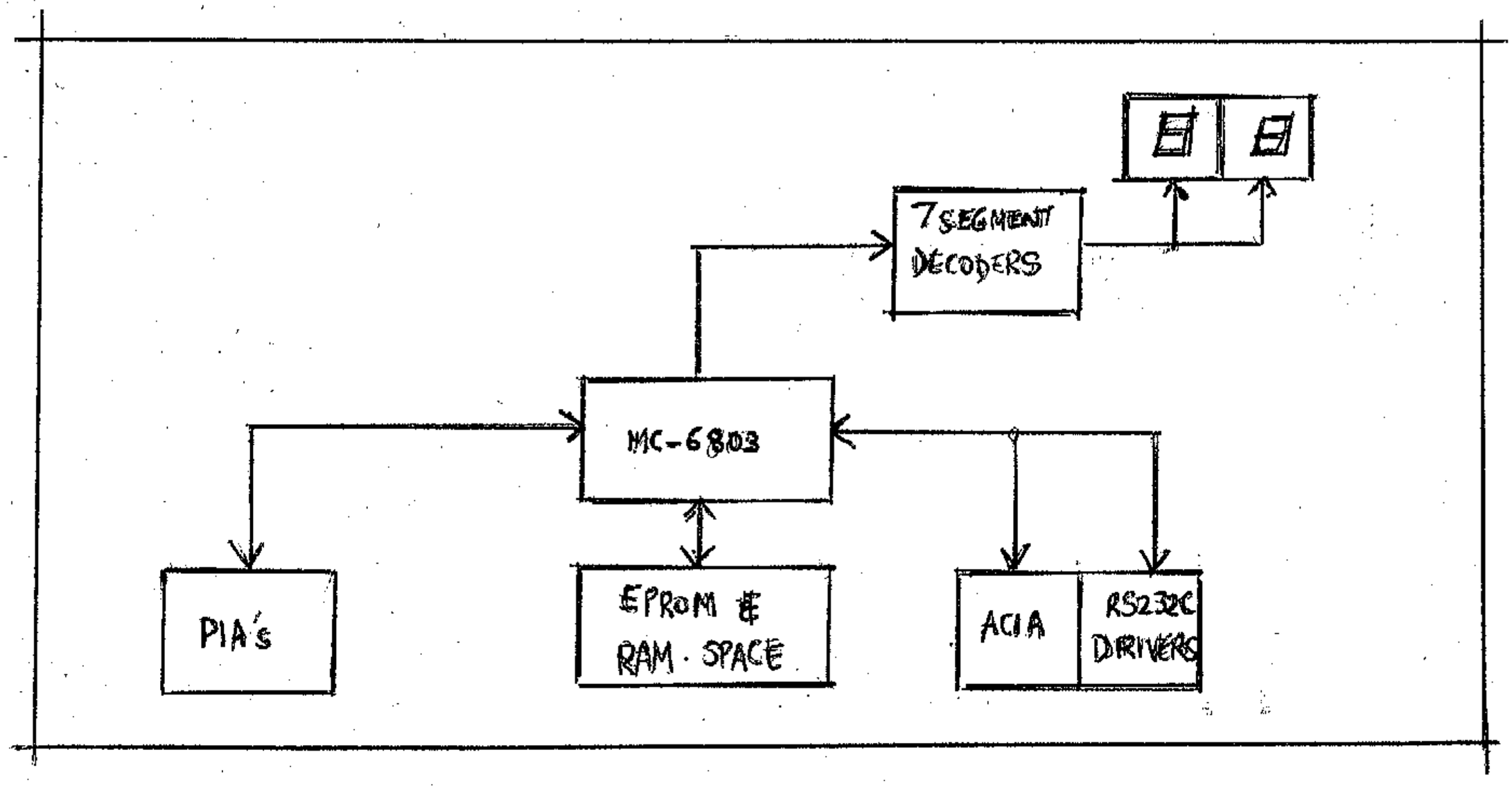


Figure 7.1. EMC card Block Diagram

Using the available extender card enhanced debugging immensely. A combination of serial, parallel communication using the MC_6821, MC_6850 was also possible. The interrupt request (IRQ) line was tied up in a wire-AND configuration, allowing the on board PIA and ACIA, or an external device to interrupt the processor.

For self checking, 2 seven segment decoder and driver I.C's are provided and are connected to port 1 of the MC_6803. This is extremely useful for immediate visual feedback for, say, reflecting the contents of the data register in the PIA.

7.2.2 Modifications

BTO.

There were some changes made, otherwise the steady development of familiarity will be interrupted. The following is a short list of changes that have and would have been made.

Reset Switch

Having a switch like this is very important in any development system. A small push-button switch was added to reset the power-up 555 timer.

power supply ripple

Perhaps this was an oversight in the design of the controller card development work was unexpected. At the moment, the existing D.M.R. equipment nicely interface to it. A temporary solution was to connect a capacitor at the input of the voltage regulator. This limited the to 1 volt peak to peak ripple and it was able to drive the interface and the monitor. More detail, due to the volume of this work, and the thesis submission deadline, will be added to an appendix at a later date.

Baud Rate

Default at 1200 baud. Karri is set at 2400 baud, can be changed but other student would need to revert it back. Too much of a problem seeing the system supervisors to toggle between 1200 and 2400 baud continuously. Tried to change the crystal. As result, was not able. Internal counter provided only 1200 and 4800 baud. Not feasible. Solution: used the EVM to change the rates every time the EMC was to be used. More detail, due to the volume of this work, and the thesis submission deadline, will be added to an appendix at a later date.

7.2.3 Difficulties

The first problem was loading the assembled code into the EMC card. There was a baud rate incompatibility with it and the host computer, the karri. See appendix 1.1 for a temporary solution.

7.3 Software

With its monitor, Lilbug, based on the Motorola MC_6803, many operations of development work can be carried out.

7.3.1 Obstacles

There was no cross assmebler from the 6809, or even 'C' to the 6803 code. Since the 6803 was an upgraded version of the 6800, the Amsterdam Compiler Kit (ACK) was used as no Motorola Assembler for the MC 6800 on the PDP-11's and the VAXes, including the karri. See appendix 1.3, for the conversion between ACK and Motorola code.

The user must find the time to use it fluently, as it is more sensitive than the commonly School used 6809 Jeffbug monitor. When used to Jeffbug, Lilbug terminates further interpretation at unexpected places. Another annoying 'feature' is the alphabetic characters must be in upper case. If not, the lines will terminate.

7.3.2 Monitor

This small monitor, 2 Kbytes worth, has the basic functions of examining memory locations, changing them, executing and tracing the code, and has its own input and output routines. From the view of confidentialty, the listing has not been included but how to use it is found in Staugaard.

7.3.3 Traps

Reading the documentation carefully is a great aid in interfacing. For instance, port 1 in the MC_6803 was assumed to act exactly like a PIA. It was not as complex. It was far simpler. All that had to be done was to set the direction of the data flow register and pump the data through it. This can save many hours of debugging. See appendix 6.2 ("p1countup") for some programs to test the EMC card.

8. INTERFACE OVERVIEW

8.1 Introduction

For an interface to function as it is intended, the characteristics of each complex unit to the interface must be ascertained. At least, know what input and output lines are available. If there is software, as there usually is, there must be familiarity with the operating system and the programming language.

In this brief chapter, mention will be made of both the hardware and software approaches to designing an interface. Of course, this sophistication of the approach depends on the experience gained so far. It is hard to separate the hardware and the software, but this has been resolved into the following two chapters, 8 and 9.

Most often, the hardware must be first designed then the software utilises the hardware. This modular approach of separating the hardware and the software with the general format of the next two chapters. The tasks going to be performed by the software sometimes influences many decisions that cause the hardware to evolve.

8.2 Design Phase

It is best to have an idea of what basically needs to be done, merge that with a general idea of the hardware and software, and solve the problems on the way. This is, of course, assuming the interfacer does not have an intimate working knowledge of both systems to be linked together.

For the hardware side of design, a timing diagram is a very useful map. From it, "signatures" can be established so that in future servicing, it can be quickly used to pin-point the fault. For the software half, a flowchart is the equivalent. Figure 8.1 shows a typical design cycle of a prototype.

In terms of software, a logical process needs to be followed.

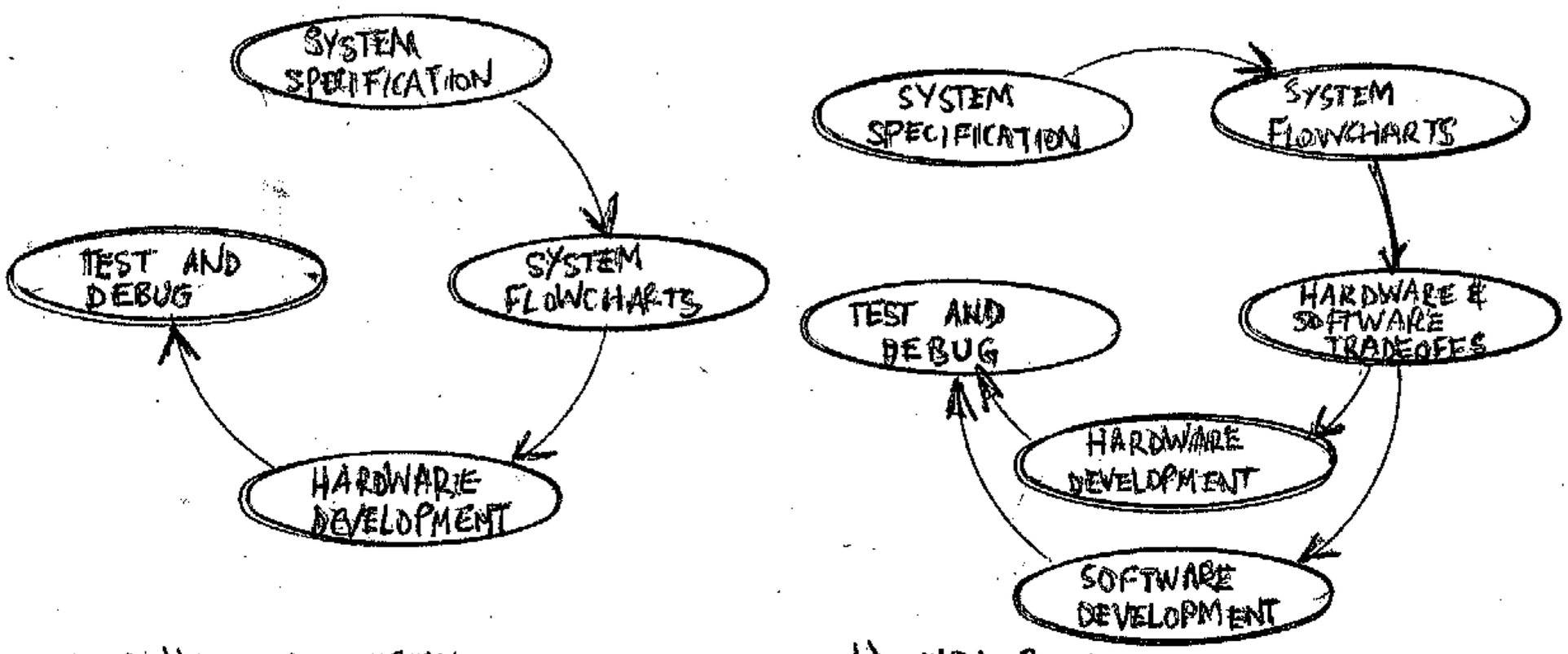


Figure 8.1. Design Process

8.3 Com patibility

There were several interface considerations. Overlooked usually, is the baud rate of the two systems. In this case the EVM has the ability to change it, but the EMC card cannot. Usually the master has this capability. Here it does not. See chapter 7, section 2.3 and chapter 6 on the EVM.

8.4 Interface Development

The understanding of the EVM grew out from continuous cycling through the "strangeness" of the design. The first version gave the author a better feel for the operation of the devices that were being used. Once the familiarity was there, the bugs mostly 'fell by the wayside'.

In the case of the software, the first version was purely an exercise in getting familiar with the Motorola MC_6800 programming language again. Going back from MC_6809 to MC_6800 was a big step. As a result, the author was trapped by assuming too great a flexibility in the software. The chapter on the Eurocard Micro_processor Controller Card, section 3.3, mentions the major traps.

One advantage of having a development system is that the program can be modified in RAM as opposed to burning it into EPROM. This allows the programmer to easily modify the program as it is in the RAM. For instance, if bit 3 was set in the PIA control register, it will mean that the number already there, '38' say, must have been '30'. This means the 'CB2' line will be in a HIGH state when it should have been LOW - preventing the whole handshaking process. See chapter 9 section 4.1 for the relevance of this example. Hence, a few seconds is more appreciated than a half an hour. Especially in developing programs.

There were, of course, parts in the documentation of the TMS_320_10 where actions of certain operations were not clear. For instance, for the TMS_320_10, the BIO pin was assumed to be latched, as nothing in the May 1984 edition of the User's Guide had stated whether it was or was not latched. The confirmation came from the November 1984 edition. It was latched. Not being aware of all the documentation can be a great disadvantage.

This chapter has covered the area of the design phase, the compatibility and the developmental process of the interface.

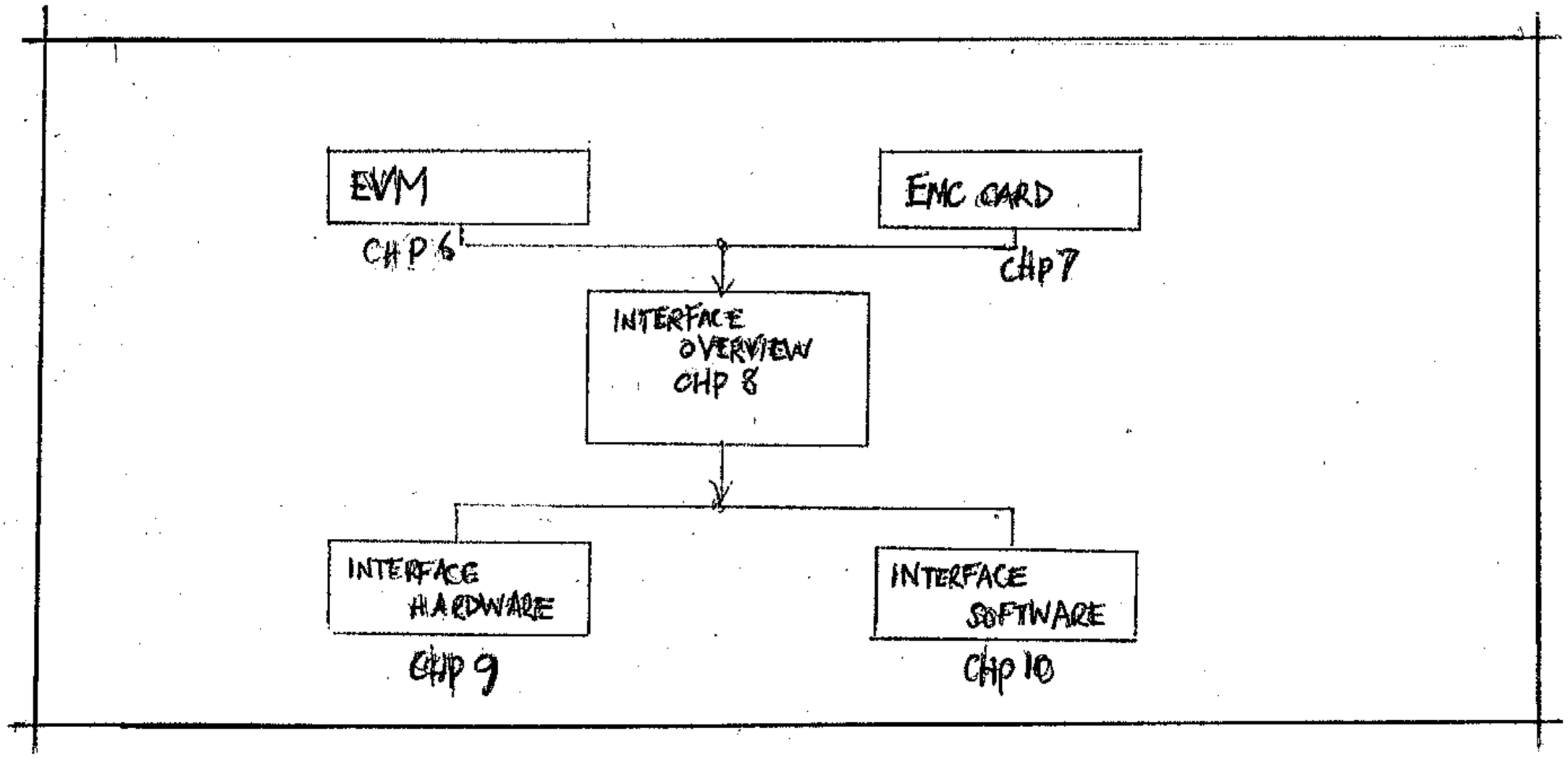


Figure 8.2. Intermediate Chapter Overview

The next two chapters will discuss the hardware and the software from the point of view of chapters. Figure 8.2 shows where the reader is, in the stage of interface development.

//

9. INTERFACE HARDWARE

9.1 Introduction

This chapter will describe the hardware of the interface, state the considerations, circuit operation, and advantages of the interface.

9.2 Re statement of Problem

The interface must be able to cope with the speed incompatibility of the two processors. Therefore, it must be asynchronous and not interfere with the operation of the other processor. It also must be able to hold the data until the 'data available' and 'data accepted' signals have been sent and received via this interface.

9.3 Considerations

Low power schottky (LS) was chosen because the speed of the processors had to be catered for. It will be quick enough for the handshaking, the enabling of the bus transceivers and the data latching. Also another choice was to use the National Semiconductor 74LS245, an octal transceiver, even though the AMD_8197 and AMD_8198 octal buffer was commonly stocked in the School store.

As it has a bi-directional data bus, there must be some conflict controlling mechanism. For instance, the accepting processor would have to read the data and an acknowledge (ACK) be generated so the sending processor will not overwrite the existing data intended for the receiver.

Most importantly, are the initial conditions of the output line because if some action must occur on a specific edge, then it must not occur during a power up or reset condition. That is, the edge may be caused when the system is reset.

9.3.1 TMS 320 10

The EVM allows access to the TMS_320_10 via port 5. See the circuit diagram in appendix 10.1 or figure 6.1 or 6.2. The available lines from the TMS_320_10 are the address, data, two active (low) output signals: the write enable & the data enable, and the BIO pin for synchronising inputs.

The program is downloaded and stored into dual port RAM via the EVM monitor. In future, an EPROM with the program in it to save the downloading time.

9.3.2 EMC card

This has a PIA where both ports are used to provide a 16-bit word transfer. Included are the control lines of the PIA. That is,

a. CA2 and CB2 to control the Tri-state outputs of the latches, and

b. CA1 and CB1 to act as 'interrupt' lines: the CA1 as a 'data available' and the CB1 as 'data accepted'.

On the card there is a 6116, 2-Kbyte RAM I.C., for storing the data and a MC_6850, an ACIA, for serially shifting out the data.

Without bringing the software into the description, the timing diagram appears to be the best medium for explaining how the interface functions. As pointed out earlier in this report, the TMS_320_10 is the 'slave' and the EMC

card is the 'master' processor. See figure 9.1 for a functional diagram of the interface.

If a detailed diagram is required, then see appendix 10.2. For a signal line with an asterisk after it will cause that line to be in an active low state. For instance, WE*, means that the 'write enable' line (WE) is active low.

9.4 Circuit Operation

9.4.1 Initial Conditions

When the system has been reset, the CA2 lines goes HIGH (active low in operation) and the CB2 line goes LOW (active high in operation). This was nearly overlooked because the data sheets for the MC_6821 (a PIA) were misleading. That is, on the timing diagrams, the CB2 lines is seen as going (active) low when being used for a write strobe function. For proper function of the interface, both the CA2 and CB2 output defined control lines must be in the HIGH state.

9.4.2 Master Sending (Slave Receiving)

Assuming the data has been presented through the PIA onto the 'PIA data' lines, the CB2 control line causes the negative edge to set latch C in figure 9.1, next page, thus causing the BIO line to be held LOW. That is, the 'data available' condition has been sent. Meanwhile, the data in latch B will be waiting to be clocked through to the output by the TMS_320_10.

When the 320 in the code recognises the BIO pin is LOW, it will branch to the routine to collect the 16-bit word via the 'in' instruction. So, it will generate the correct address (A0 = A1 = A2 = HIGH) and cause the data enable line (DEN*) to clock the data through the latches and be channelled through the transceivers into the data memory of the TMS_320_10.

As this edge clocks the data through so it could collected, it also caused latch C to be reset, letting the BIO pin to be internally pulled high, and

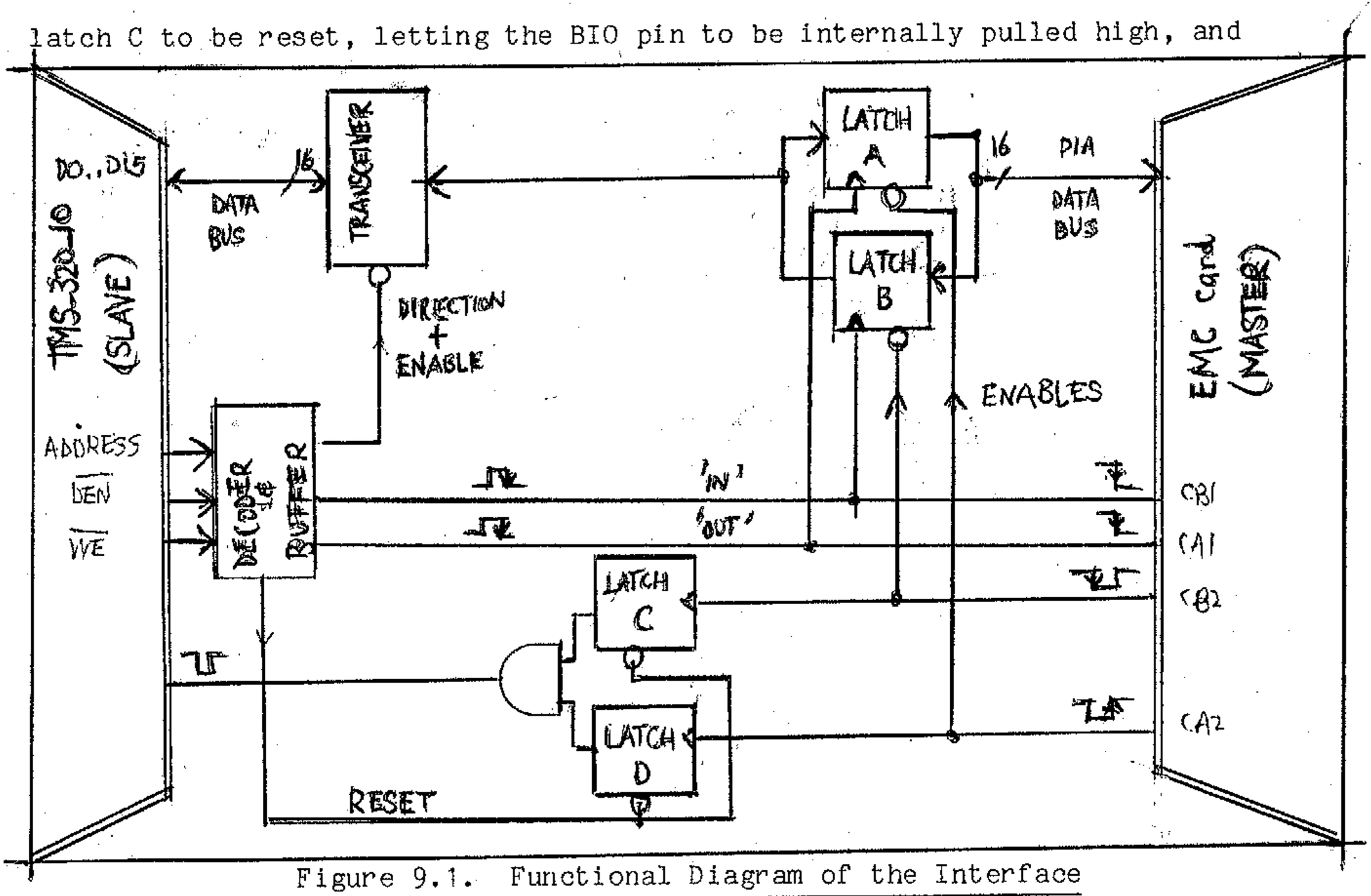


Figure 9.1. Functional Diagram of the Interface

a negative transition on the CB1 line, that is, a 'data accepted' condition has occurred.

This transition in turn, caused a flag to be set in the control register of the PIA, so that when the MC_6803 recognises it has been set, it will force CB2 HIGH, disabling the output of latch B and then will act accordingly. Note, the automatic reset of the BIO pin by an input or output instruction one of the

7 ports.

The following is a short step by step summary of the procedure involving the sending of a control word from the master (EMC card).

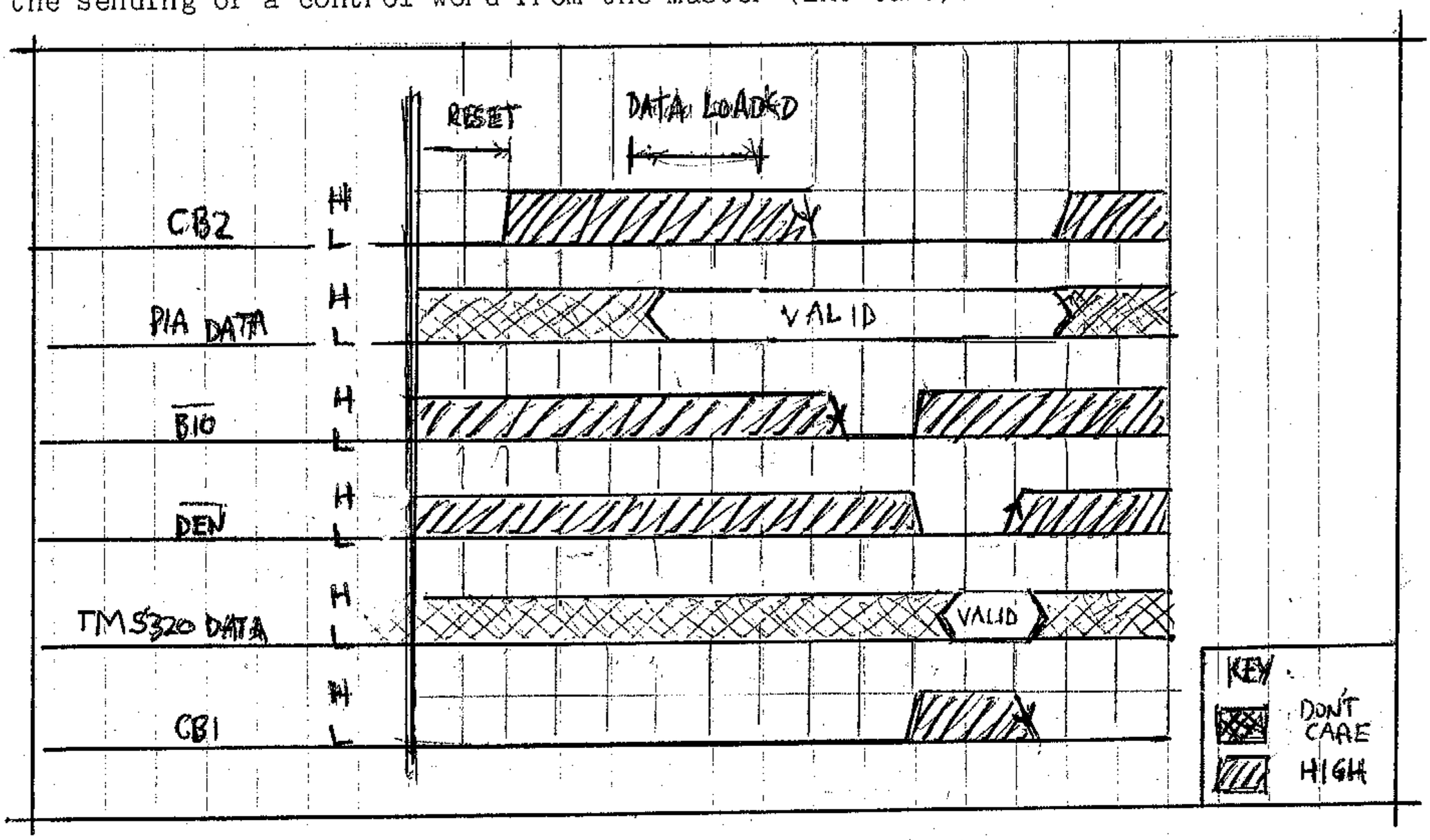


Figure 9.2. Master to Slave timing

1. load data onto PIA data lines
2. CB2 goes low
3. BIO* of latch goes low
4. check BIO, if not LOW then continue main program and check next time around
5. input data (DEN* --> LOW)
6. latched cleared
7. 320 collects data when valid .
8. DEN* --> HIGH
9. CB1 --> LOW
10. CB2 reset to HIGH automatically by CB1 transition.
11. end of session

The reader may understand the sending routine better if the timing diagram, figure 9.2, is looked at.

9.4.3 Slave Sending (Master Reading)

The sending from the slave is like a 'mirror image'. Only it's done in reverse. Assuming the slave (TMS_320_10) has been requested by the master to send the 16-bit word, the following is a description of the process in transferring this information.

Referrring to figure 9.1 again, the slave outputs the data from data memory, via the 'out' instruction, causing the write enable (WE*) line to go (active) LOW, the direction on the buffer to be set, and clocking it into latch A. At the same time, the negative transition causes the input control line, CA1, to register the change. So, as in the master sending case, the master sees the 'data available' signal and then forces the CA2 line to go (active) LOW.

At this point, the data appears at the output of latch A. The master collects the data off the PIA data lines and then forces the CA2 line back to the HIGH state. But this active (HIGH) transition cause latch D to hold the BIO line down (LOW). This is now the TMS_320_10's 'data accepted' signal. Now the data has been transferred. It is up to the software to recognise the correct sequence of events. See the next chapter for the software side of the protocol.

The following, as for the case of the master sending sequence, is a step by step account of the transfer, or if again, the reader wishes to peruse the timing diagram, then see figure 9.3.(next page).

1. master waits for flag in control register to be set (CA2 = HIGH in the meantime)
2. slave executes 'out' (WE* --> LOW)
3. data on the '320 bus is now valid
4. WE* --> HIGH, data clocked into latches
5. CA1 --> LOW
6. master recognises flag is set

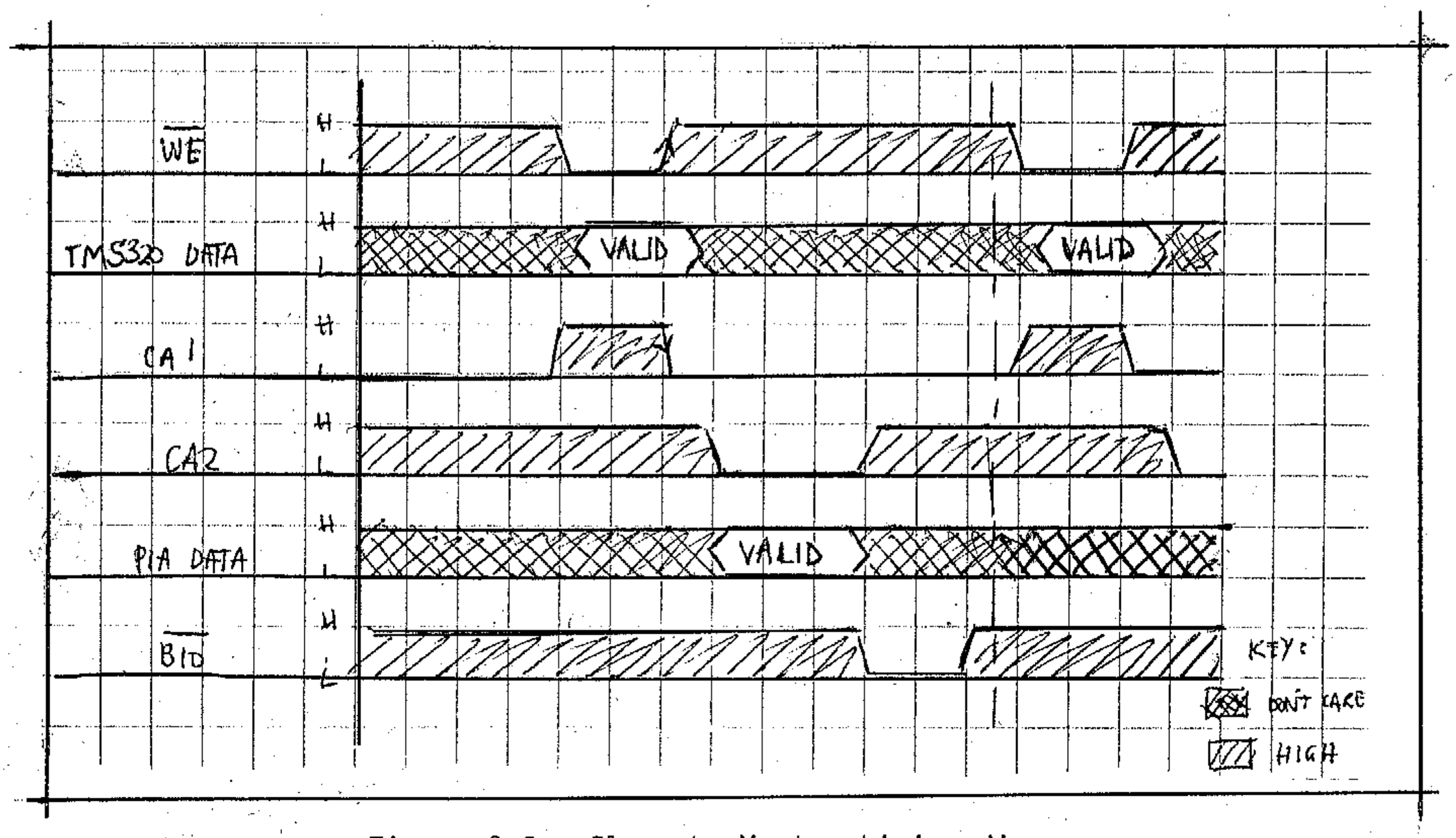


Figure 9.3. Slave to Master timing diagram

7. CA2 --> LOW
8. PIA data lines are valid
9. MC_6803 reads data
10. MC_6803 causes CA2 --> HIGH
11. BIO latch set LOW
12. wait until '320 clears by 'in' or 'out'
13. end of session

9.5 Advantages

As it is inherent, the first advantage of the interface is the use of the edge triggered signals. For instance, the DEN* or the WE* used to reset the BIO pin and cause a negative transition on the corresponding CA1 or CB1 lines.

Secondly, the outputs of latches A and B are held in tri-state by the master. Thus, saving excessive power dissipation and more importantly, preventing any bus conflict. Note, tri-state buffers were used so as not to halt the TMS_320_10 data bus, during its execution, when the data was being transferred.

Thirdly, the '320 literally needs on to supply the address, data, enable lines and the BIO line. This means any other device having these signals will be easily interfaced to this unit.

Finally, the correct port is recognised. From the unique input instruction but the not-so unique output instruction, the output port can initiate the data transfer. Note, the output is not the only instruction that causes the WE* lines to go (active) LOW. See chapter 6, section 4, for a discussion of the problem.

In summary, this chapter refreshed the reader of the interfacing problem, mentioned the major considerations, described the hardware transfer of the data in both directions and listed the advantage of the interface built. The next chapter will cover the software side, as difficult as it was to separate it, of the interface.

//

10. INTERFACE SOFTWARE

10.1 Introduction

Most of this chapter will assume the reader has read the previous chapter on the hardware of the interface. This chapter is basically divided into two sections. The first, dealing with the Eurocard program, and the second, with the TMS_320_10 program. It will cover both programs in simple flowchart form, but if the reader wishes to read the program, he/she can refer to Appendix 5 for the listings.

Last chapter described the hardware side of the interface. This chapter will remind the reader of the protocol to be used, outline the software involved on both sides of the interface by including the use of flowcharts, and mention further work of the program.

10.2 Protocol Reminder

As mentioned in the concept of the interface chapter 3, the master (EMC card) will send a control word containing the 'request' or 'change' command, how many words involved in the transaction, and the starting address in the data memory. After this control word is sent, the data exchange will occur.

10.3 TMS 320 10 Data and Program Memory

The aim was to have the master modifiy specific parameters in the memory of the slave processor. Naturally, it was assumed these changes were made in the program as is normally done in any Princeton architecture, like that of the 6809

code.

From this, the first version of the interface was built and the corresponding protocol written. The real advantage of the modified Harvard architecture wasn't realised until a few weeks later, then the conceptual breakthrough was made.

Because of the Harvard architecture, the coefficients are read from the program memory into the data memory using the 'table read' (TBLR) instruction. All the calculations are carried out in the data memory, including the auxiliary registers and accumulator. When the program memory needs to be updated from the value in data memory, the 'table write' (TBLW) intruction is used. See appendix 6.2 for more information on the use of the TBLW and the TBLR. Here, version 2 was created.

10.4 for the EMC card

The main problem was in getting familiar with the assembler language in differentiating between the direct, indexed and immediate modes for instructions. Appendix 3 (chapter 15) lists the possible commands for this software written as the master program. For someone who is familiar with assembler code, they will see the program, "main.s", listed in appendix 5.1, as very straighforward.

Everything is straightforward, IF the idea has been conceived. The only roadblock to understanding is the presentation of the idea. Hopefully, the flowchart depicted in figure 10.1 will be self explanatory to the reader. If the reader wishes to skim through the program, then see appendix 5. Also in appendix

6 there are short test programs that were used to verify the functioning of the interface.

The following will quickly walk through the flowchart in the event that some aspect assumed by the author is not clearly evident. The interface program prompts the user for a command.

If the master wishes to 'read' from the slave a block of ten words, starting from the data memory address equal to fifteen in the slave, then the control word is constructed to contain the number of words requested, from the address specified, and the type of command that is being sent. In this case, a 'read', once sent, the master sets itself up to collect the next incoming ten 16-bit words into its RAM, as discussed in the previous chapter on the hardware.

In the case of the 'change' directive, a similiar control word is formed. Only the 'type of command' section in the control word is altered. Now the data words entered from the keyboard are transferred into the slave's data memory. Other functions available are, an 'escape' to the monitor, and a display, for verification of received data. See also section 4 in this chapter for further work to be done.

10.5 for the TMS 320 10

Here, it will be assumed the reader has been familiarised with the down loading of the program for the TMS_320_10. See appendix 1.2 for a reminder.

This side of the protocol hinges entirely on the recognition of the BIO pin. As shown in the flowchart, figure 10.3, the user is virtually able to write and run any program. The only limitation, of course, is the restriction on using the BIO pin. For this application, an infinite impulse response (IIR)

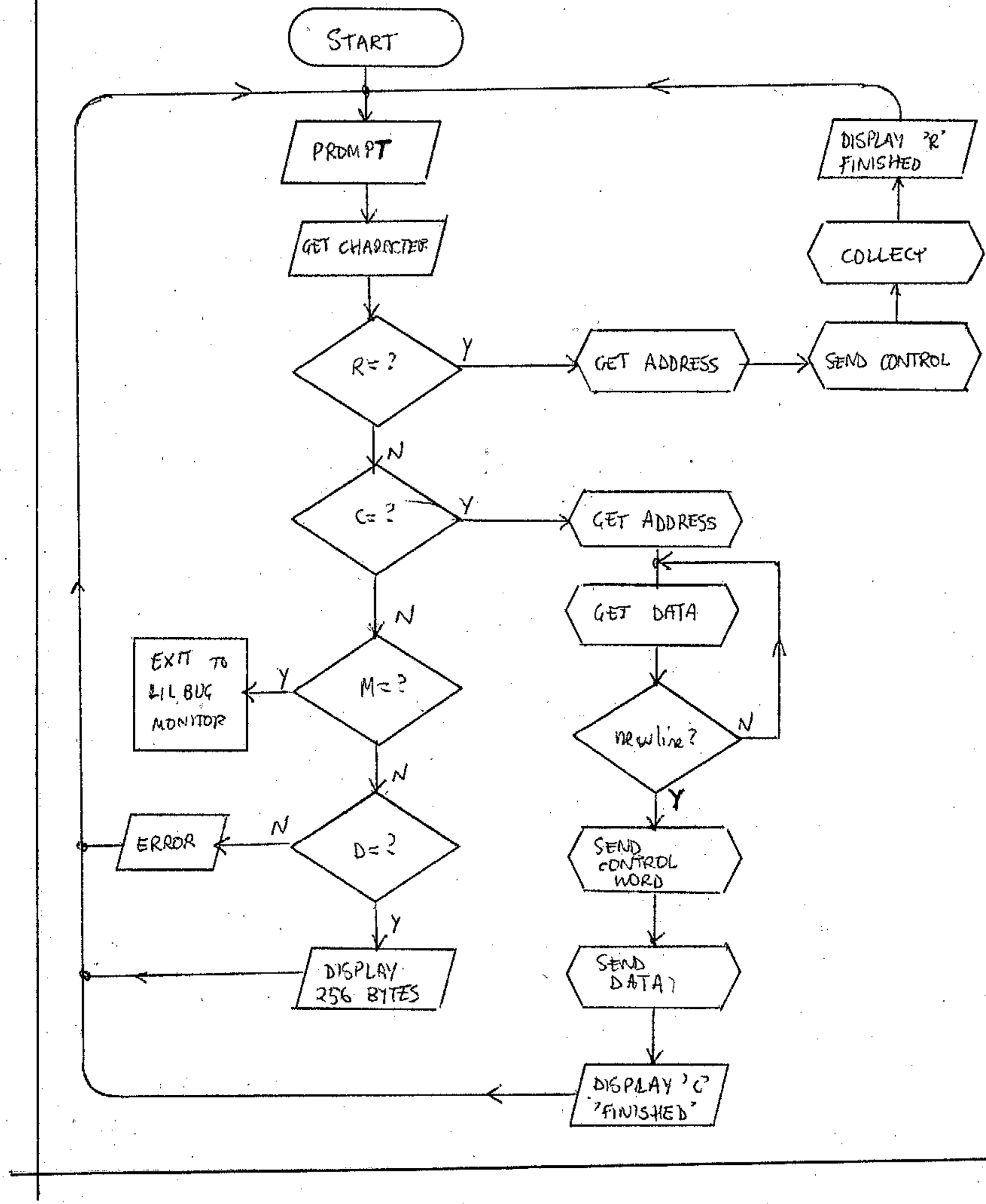


Figure 10.1. Flowchart for the Master : EMC card

digital filter, derived by the bi-linear transform method to calculate the coefficients was used.

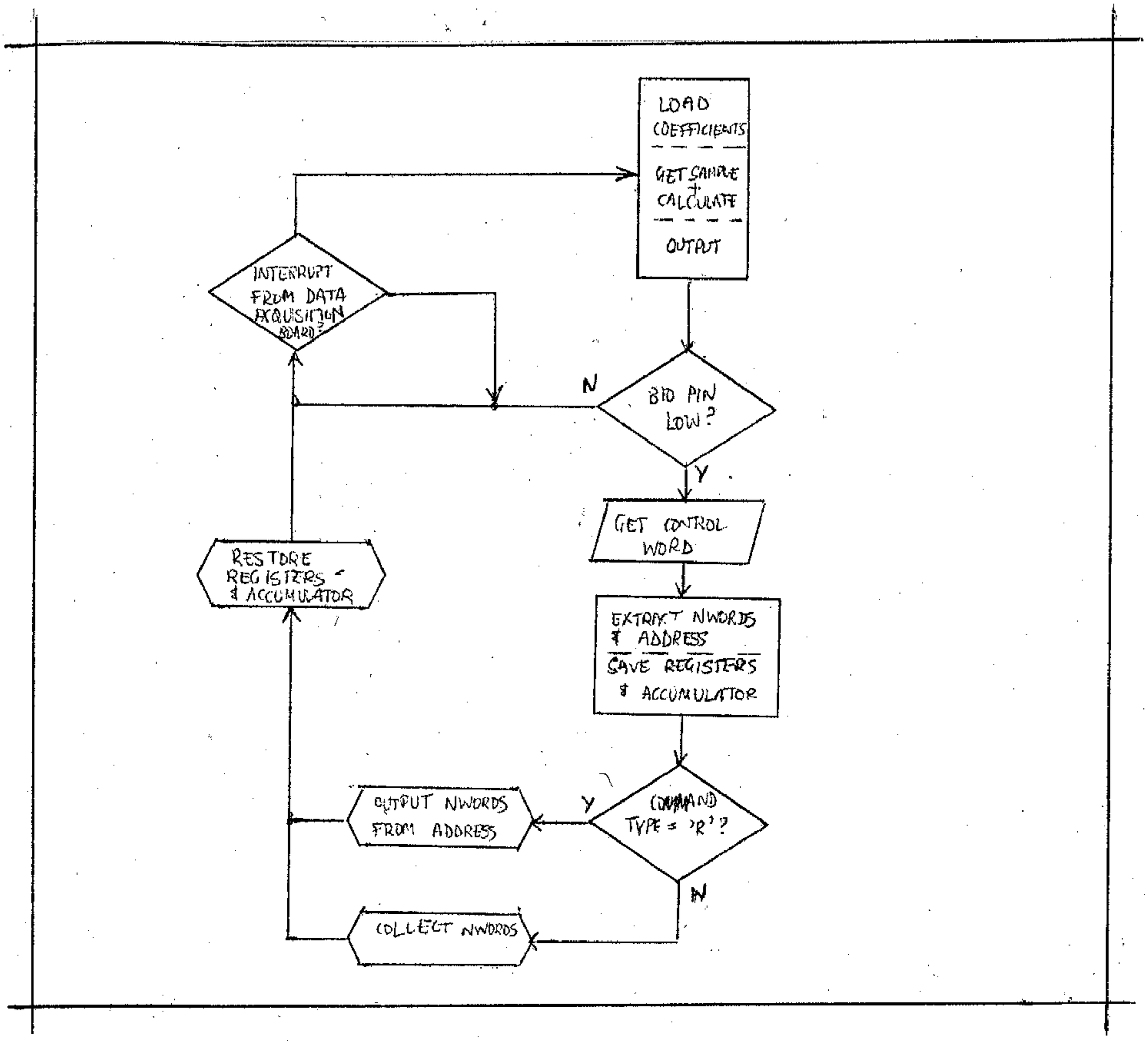


Figure 10.2. Flowchart for the Slave : TMS_320_10

10.5.1 Test Program

To demonstrate the interface, it was intended to change the cut-off frequency of the filter, from 1 kHz to say 4 kHz, so that the observer can see the change in the filter coefficients on the oscilloscope, instead of using the monitor to look through the memory. This appears to be more authentic and acceptable than searching through memory for the 'right' numbers.

The general form of the second order Butterworth filter function in the 's' and the 'z' planes are shown in figure 10.3. Note, the variables 'a', 'b' and 'c' are the coefficients. H(s), is the analogue transfer function. 'y0' is the digital output where 'y1' is the unit delayed sample of the output 'y0' and similarly 'x0' is the current input, 'x1' the delayed version of 'x0'.

$$H(s) = \frac{a}{s^2+bs+c} \quad \text{- - - - - - - (s plane)}$$

(z plane)

$$y_0 = \frac{-(2c-2)}{1+b+c} y_1 - \frac{(1-b+c)}{1+b+c} y_0 + \frac{a}{1+b+c} x_0 + \frac{2a}{1+b+c} x_1 + \frac{a}{1+b+c} x_2$$

Figure 10.3. Filter function for Second Order Butterworth

The TMS_320_10 uses integers for its calculation unlike the Intel 2920 device using negative powers of 2. So, the TMS_320_10 numbers need to be scaled. In this case, the scaling was 4096, a "Q12" number. It so happens that 2 to the power of 12 is 4096. This means that the coefficient of 'y1' being 1.56102, is scaled up to 6394. See appendix 9 for a worked example synthesised from Gold and Radar.

10.5.2 Operation

Referring to the flowchart in figure 10.2, when the main program is executed, it, in turn, encounters the 'BIOZ' instruction and branches to the

routine to input a 16-bit word, if the BIO pin was in a LOW state. But first, it stores both the auxiliary registers and the 32-bit accumulator in the user defined stack, to allow for the protocol as much freedom as needed in the event of future changes.

Now, it extracts the number of words and the address from the control word. It also checks for the type of command being sent. If it is a 'read', then it branches to a routine to output the correct number of words. After this, it branches to a routine that restores the auxiliary registers and accumulator and the back to the main program - continuing execution. Note, the way it is drawn, corresponds to the data acquisition board using the interrupt line of the TMS_320_10. See the program listing 'lp2nd.t' in appendix 5.2 for more information. The next section discusses problems and possible extensions of the software.

10.6 Further Work

Most of the test programs are contained in appendix 6. The convention for the filenames listed, is that the ones ending in '.t' are the TMS_320_10 and the ones ending in '.s' or '.i' are the Motorola coded.

10.6.1 Problems

The following is the major fault that cropped up during the writing up of this thesis. As the software currently works, both sides do not know when to stop sending or receiving. A possible solution can be to use a counter, as the number of words, nwords, is transferred.

Another problem lies in the interpretation of the monitor, Lilbug. In the 'display memory' command, an optional feature on the interface, the initial and final address for the routine 'io' appeared to be correctly set up, but the displaying does not cease at the final address. This needs to be looked into carefully.

A maximum of three hours of debugging, is predicted, to have the system handshaking fluently. This time was essentially replaced by the planning of the writeup.

10.6.2 Extensions

With the data in the RAM on the EMC card, more software can be written to use the MC_6850 (ACIA) to serially transfer the data out to a host. Perhaps, even to receive ? But it must be at the 1200 baud as the receive and transmit clock to the ACIA are connected to the MC_6803 clock (port2 bit2: P22).

As the program was developed in the RAM, the eventual aim was to store the master's program an into EPROM. This seems a natural progression.

In summary, this chapter discussed the software of the interface in both processors, the TMS_320_10 and the MC_6803. It very quickly stepped through the operation of the program via the interface. Also, an example program was provided, to use the interface, as well as further work to be done was mentioned. The next chapter will consider the overall decisions to further the aim of the project in subsequent years.

11. FUTURE CONSIDERATIONS

11.1 Introduction

At this point of the thesis, it is well and good that the interface is on the brink of functioning, many questions can be posed as to how might the VME bus system might support the interface when it arrives at the D.M.R. What about the accomodation of the new TMS_320_20 I.C. ? Will the ideas from the interface designed for the TMS_320_10 be applicable to the newer I.C. ? This chapter will attempt to predict some of the paths that may be taken, if and when the circumstances eventuate.

11.2 Limitations

The fact that any future attempts based on the series comparable to the TMS_320_10 EVM, software and hardware is available in the School to make the project feasible. Unlike the problems already encountered with Texas Instruments code, chapter 6.3, it would be a wise decision.

An idea can be to switch over the block of RAM, as in the dual port concept (chapter 6, section 3.1.1), to the TMS_320_20, giving it the ability to use the MC_6809 to read or write into it. That is, it can be a more dedicated version of Evaluation Monitor. But the drawback is, a Motorola MC_6809 based EVM can take a significant time to set up.

11.3 TMS 320 20

A cursory inspection of the TMS_320_20 revealed, it has no longer the limits of the other DSP I.C's - the TMS_320_10 and the Intel 2920. It has more program and data memory. There are more signal lines such as 7 levels of interrupts, data bus grant and request, and strobes for the data, program and input output spaces of the memory map.

Also there is a serial port and an on-chip timer for high speed communication. The data to be transmitted (and received) can be either an 8-bit byte or a 16-bit word. From the preliminary User's Guide for the TMS_320_20, the calculated maximum rate of the serial port was found to be 0.5 Megabits per second for an 8-bit word. See appendix 2.2 for further material.

11.4 VME bus

The VMEbus is mainly suited for big systems. All the boards are independent of each other, leading to asynchronicity. If the second student is to use the VME bus system they must get familiar with the Motorola MC_68,000 series of micro_processors. This implies the system is working before any significant work can be carried out. Next, is the concluding chapter of this thesis/exploratory project.

//

12. CONCLUSION

Many avenues have been uncovered and explored as a result of this opportunity given by the Department of Main Roads. Unfortunately, the interface was uncompleted but the major part of the 'uphill battle' was overcome. It can be classed as 'near the end of the breadboard' stage.

However, an interface between the TMS_320_10 and the EMC card was developed, and a simple handshaking stop and wait protocol was implemented. The 'last minute' hardware redesign proved beneficial. Due to a time constraint, the software needs a maximum of three hours of patching before the system is fully operational.

Many assumptions are made, in the light of the interfacer's experience, but when the standard is changed, several pieces of information are not self evident. Consequently, they are viewed as missing. So, it is suggested, the EPSON HX-20 be briefly reviewed in the event that the voluminous material covered by the Technical Reference Manual, the key may have been overlooked.

The purpose of this thesis was to convey information about assumptions made and the ideas taken for granted. On the superficial level, it is said "there's nothing to it". When problems such as bus conflicts and timing rear their heads, then the interfacer must have a deeper knowledge of the building blocks of the system to be interfaced.

There were problems, mainly not to the author making erroneous assumptions, but to the lack of proper documentation. As experienced, the first copy is the one to be the most wary of. This was pointed out in the TMS_320_10, Chapter 9

and Chapter 3 section 4.3.3.

The new TMS_320_20 just released, appears to be more suited to the overall aim of the project. It has more program and data memory and a serial port. Better still, the instruction set is upward compatible from the TMS_320_10.

There must be some dedicated time in reading or scanning the documentation at hand. But not too much, as progress is made by "smaller but faster bites". An immense number of decisions had to be made, whether to keep exploring or not. This meant that the time and the amount of material discovered was kept in mind.

143

It makes sense to state: if the user wishes to simply run a system under the guise of what it was meant for, then no problems would be experienced. However, anything slightly more or different to be demanded from it, then be prepared to find out the 'hows' and the 'whys'.

As a summary of this thesis / project, it is suggested the reader glances at figures 2.1, 5.2 and 9.1, before putting this report down for the last time.

pg 10 pg 23 pg 48

13. REFERENCES

1. KEY: Author's SURNAME, Initails; Author2; Author3; ..,
"Title of Publication",
Publisher, Date when Published [mth] [yr], [Vol., Nr., Chps, pages.].

2. BELL Telephone Laboratories.,
"Unix Word Processing Manual : Level Six",
Dept. of Civil Engineering, U.N.S.W., June 1980, Sections 1,4 & 5.

3. BROGAN, John A,
"Clear Technical Writing",
McGraw Hill, 1973, .

4. CONNOLLY Ed,,
"New Products section in Electronic Design",
Computer Standards: Designer's Reference, Dec 23, 1982, p117-132.

5. DA Cruz, Frank,
"Kermit Protocol Manual",
Columbia University Centre, 3 April 1984, Fifth Edition.

6. DUTCH, Robert A,
"Roget's Thesaurus of English",
Penguin Books, 1975, .

7. EPSON,
"Technical Reference Manual",
EPSON Corporation, Japan, 1983, .

8. EPSON,
"HX-20 Technical Reference Manual",
EPSON Corporation, 1983, .

9. EPSON Corporation,
"Operation Manual: HX-20 portable computer",
EPSON, 1982, .

10. FAIRCHILD,
"TTL Data Book",
, 1985, .

11. GOLD, Bernard; Rader, Charles,
"Digital Processing of Signals",
Lincoln Labortatory Publications, McGraw Hill, 1969, .

12. GREENFIELD, J. D.; Wray, W.C.,
"Using Microprocessors and Microcomputers: The 6800 family",
Wiley & Sons, 1981, .

13. INTEL Corporation,
"Intel Telecommunication Handbook",
, 1984, Chp 4: pp 4-1 to 4-37.

14. INTEL Corporation.,
"2920 Analogue Signal Processor Design Handbook",
Intel Corporation, Aug. 1980, .

15. KERJAN P,,
"Digital Signal Processing and Real Time Filtering Using the TMS32010 Digital Signal Processor",
Undergraduate Thesis, University of N.S.W., Nov 1984, .

16. KRAFT, George D; Toy, Wing N,
"Mini/Micro_computer Harware Design",
Prentice Hall, 1979, pp 112-125.

17. MOTOROLA Semiconductor Products INc.,
"Single Chip Micro_computer Data",
, 1984, Series C.

18. MOTOROLA Semiconductor Products INc.,
"8 bit Micro_processor and Peripheral Data",
, 1984, Series C.

19. RADZYNER, R.,
"Single-Chip Digital Signal Processors",
The U.N.S.W. Division of Continuing Education, 1984, Lec. 6-7 in "Keeping Up-To-Date with Electronics Developments".

20. SIGNETICS,
"TTL Logic Data Manual",
, 1982, .

21. STAUGAARD Jr, Andrew C.,
"6801, 69701 and 6803 Micro_computer Programming and Interfacing",
Howard Sams, 1974(?), .

22. TANENBAUM, Andrew S,
"Computer Networks",
Prentice Hall, 1981, p 143-153.

23. TEXAS Instruments,
"TMS_320_20 User's Guide - Preliminary",
, March 1985, .

24. TEXAS Instruments,
"TMS_320_10 User's Guide",
Texas Instruments Inc., May 1983, Chp 1-6, 8, Appendix A.

25. TEXAS Instruments,
"TMS_320_10 Evaluation Module User's Guide",
, February 1984, .

26. TEXAS Instruments,

"Interfacing to Asynchronous Inputs with the TMS_320_10 (Application Report)",
, 1984, p 5-6.

27. **TEXAS Instruments,**
"9900 Family Systems Design and Data Book",
Texas Intruments Incorporated, 1978, first edition.

28. **TEXAS Instruments Inc.,**
"TMS_320_10 Assembly Language Programmer's Guide",
Nov. 1983, Chp 1-4, .

29. **TEXAS Instruments.,**
"Precision Digital Sine-Wave Generation in the TMS_320_10: (Application Report)",
, Feb 1984, pp 1-10.

30. **TEXAS Instruments.,**
"Implementation of FIR/IIR Filters with the TMS_320_10: (Application Report)",
, 1984, .

31. **TEXAS Intruments,**
"TMS_9995 16-bit Microcomputer - data manual",
Texas Intruments Incorporated, 1982, Chp 1-4.

32. **VME maufacturer's Group,**
"VMSbus Specification Manual - preliminary",
, Nov 1983, Revision A3.

14. COMMENTS BY MARKER

These three pages are reserved for any comments the reader may wish to make.

15. APPENDICES

APPENDIX 1

Changing the Baud Rate by the EVM

A: 2400 → 1200 Baud

	UNIX	EVM
①	STTY 1200	
②		BAUD2 1200
③		BAUD 1 1200

(Set THUMBWHEEL SWITCH TO 5 FROM 4)

OR

IN UNIX	
%	STTY 1200
%	^C ; CONTROL C
%	BAUD2 1200
%	BAUD1 1200
%	^C ; CONTROL C
%	

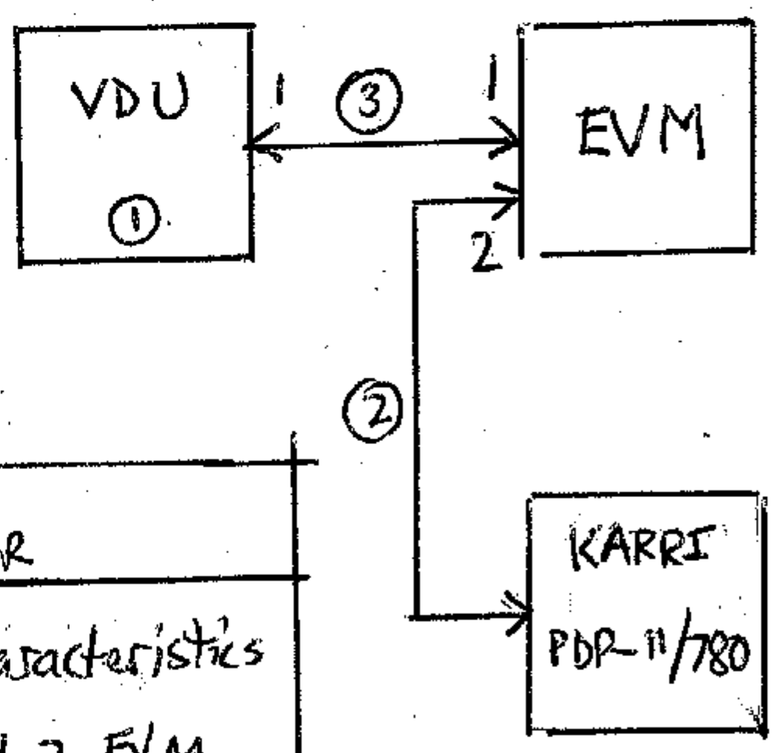
; Now can use Litbug Monitor Pgm

B: 1200 → 2400 BAUD

	UNIX	EVN
①	STT 2400	
②		BAUD2 2400
③		DAUD1 2400

OR

IN UNIX
STTY 2400
^C
BAUD2 2400
BAUD1 2400
^C
; as before



MUST BE IN THAT ORDER

① Change terminal characteristics
② " host → port 2 EVM
③ " VDU → port 1 EVM

APPENDIX 1

15.1 Down Loading

15.1.1 into the EMC card

These are two programs used to assemble the 6800 code into an object and a source listing. The second file, load2, downloads the object into the EMC card at a slower rate than 2400 baud. The third file, dog.c, is the C program that delays each output line by two seconds.

```
File : load1
   1    m00 -d -s $1 > m$1.lst
   2    sld > m.out
   3    chmod 755 m.out m$1.lst
   4    echo "get LILbug listening\7"
```

```
File : load2
   1    echo "get LILbug listening\7"
   2    sleep 10
   3    echo "L"
   4    sleep 2
   5    cat m.out ^ dog
   6    echo "S9"
   7    sleep 5
```

```
File : dog.c
   1    /*      To slow things down for the receiving end of the Monitor
   2     *
   3     *      Modified:       22 July '85                 by Bob Smith
   4     *
   5     *      For the DMR's Euro_card Controller.
   6     */
   7    /*      Usage:   .....   cat filename ^dog  */
   8    /*      to compile use cc -O dog.c -o dog  */
   9    main()
  10    {
  11    char c;
  12            sleep(10);      /*      let the monitor settle down */
  13            while((c = getchar()) != -1)
  14            {
  15                    putchar(c);             /* echo input */
  16                    if(c =='\n')            /* newline ? */
  17                            sleep(2);       /* wait 2 seconds per line */
  18            }
  19    }
```

15.1.2 into the EVM

This section shows how the user is able to down load the TMS_320_10 source code and assemble it using the EVM assembler. Note: the first character on a lines by itself in the file must be the ">", and the last character (by itself) must be the "<". These are 'markers' for the assembler to start and stop interpreting the TMS_320_10 code.

Also, the word 'end' must be on the second last line for the assember to stop assembling. It serves the same purpose as the 'end' or 'stop' statement in the BASIC and FORTRAN languages.

My conventions are:

a. The user types in characters between the double quotes ("), the EVM prints out the characters between the signle quotes ('...'),

b. words after the semicolon (;) are the comments,

c. the set of characters <CR> denote the user presses a Single Carriage Return,

d. <SP> denotes One press of the Space Bar,

e. and "^X" denotes the Control Charater X.

Activation of the TMS320 Evaluation Monitor (EVM): (**** no **<CR>** **** unless it is specifically stated.)
Initial Conditions: you are in the Unix editor of the file that you intend to assemble. Note, the '?' is the MONITOR PROMPT.

---

1. **"INIT <CR>"** ; To initialise the EVM

2. EVM responds with: '? CLOCK SOURCE = INTERNAL' and waits on the SAME line for a **"<SP>"** or an **"E"** (external),

3. **"E <SP>"** ; continue with initialisation. To exit this questioning, press **"<CR>"** instead. This will take you to step 6 of this list.

4. response: '? PROGRAM MEMORY = INTERNAL' ; and waits on the same line,

5. **"<SP>"** ; use the internal 4K (4096 bytes),

6. response: '?' ; waiting for next command,

7. **"ASM 2 <CR>"** ; assemble from port 2,

8. response: '? LINE NUMBERS? (NO)'

9. **"<CR>"** ; your input file has NO line numbers so the EVM generates them,

10. response:

'** TMS320 EVM ASSEMBLER **'

11. **"^c"** ; switch back to Unix via the "toggle" character,

12. **"1,$p"** ; prepare to concatenate your 320 source file,

13. **"^c"** ; return control to the EVM

Now the EVM reads, assembles the code and echoes it onto the screen of your terminal. Now all you type is **"EX"** (to execute) to set things in motion.

15.1.2.1 Bugs

One thing Tom Millett pointed out was that once you set some breakpoints in your assembled program you must use "EX" not "RUN". Also you can't use "EX" after "RUN" was typed in. That's the way the EVM understands things. Something to do with initialising some registers ? I haven't seen this problem indicated or documented clearly in the EVM User's Guide. Could've missed it.

On the hardware side of the EVM, there is a spring loaded red lever switch that is used to reset. Press it Once, you reset the program (known as a **soft reset**). Twice you reset the system (i.e. **hard reset**).

Another problem was with the line by line assembler in the EVM firmware. The following a TMS_320_10 program shows the bug and a way around it.

```
File : sacl.test.t
   1    >
   2    *       to see why the assembler doesn't recognise
   3    *       the SACL *+,AR1 instruction
   4    *                                        Bob Smith (4 Nov '85)
   5    *       this code will run
   6    *
   7    AR0     equ     0
   8    AR1     equ     1
   9    *
  10            lark    AR0,9
  11            larp    AR0
  12    *
  13    *       sacl    *+,AR1          * doesn't assemble at all
  14            sacl    *+
  15            larp    AR1
  16            end
  17    <
```

15.1.3 Differences

The next two sections contains some of the differences between the Amsterdam Compiler Kit (ACK) and the tandard Motorola assembler, and the Unix shell file to convert from ACK to the Motorola assembler code for the benefit of the reader who is familiar with it.

P.T.O for TABLE

APPENDIX 1

	ACK	mas00
	! comment	* comment
hex #:	0x1234	$1234
strings:	.ascii	fcc
bytes:	.byte	fcb
origin:	.org	org
equates:	=	equ
label:	fred:	fred

File : to_mas

```
    1   #       to convert from ACK to mas00
    2   #
    3   #       Usage:  to_mas file.m
    4   #
    5   #       source file must end in a '.m'
    6   #
    7   #                       Written by Bob Smith (20 Sep '85)
    8   cc -P $1
    9   mv `basename $1 .s`.i $1.m
   10   e - $1.m <<!
   11   g/^#/d
   12   gs/.org/org/
   13   gs/0x/$/
   14   gs/[!.]/*/g
   15   gs/.byte/fcb/
   16   gs/.word/fdb/
   17   gs/.ascii/fcc/
   18   gs/.space/rmb/
   19   gs/)'/1/
   20   gs:'*)':/1/:
   21   gs/.asciiz)/fcc1(
   22           fcb     0/
   23   gs/bra/jbr/
   24   gs/bsr/jbs/
   25   gs/;/(
   26           /
   27   w
   28   q
   29   !
```

P.T.O for APPENDIX 2

15.2 TMS 320 Brief

In this appendix, there are three sections related to the TMS_320 family. The first contains a comparison of the TMS_320_10 and the TMS_320_20, the second, the added features of the TMS_320_20, and finally the architecture of the TMS_320_10.

15.2.1 Comparison between '10 and '20

The next table highlights the main differences between both the TMS_320_10 and the TMS_320_20. To note, is the extra memory and registers in the newer member of the fmaily.

	TMS_320_10	TMS_320_20
type	modified Harvard	reconfigurable
instructions	60	109
data memory	144 words (INTERNAL)	64 K words (EXTERNAL)
program memory	4 K words (INTERNAL)	64 K words (EXTERNAL)
input/output ports	8	16
on-chip RAM	144 words	544 words
auxiliary registers	2 (16 bit)	5 (16 bit)
peripherals	-	timer

A dual path arises with the TMS_320_10 and the TMS_320_20. If the user intends to use one of them, consideration must be given to the application at hand. The respective prices are $ 60 and $ 500 (1 November 1985), but they still do much the same task with the TMS_320_20 being a more comprehensive device.

15.2.2 Features of the TMS 320 20

The TMS_320_20 is still a dedicated micro_processor but it has features of a general purpose processor. These features make it easier to use. Although having used the TMS_320_10, we basically saw in Chapter 6, section 3, how the TMS_9995 controls it, the TMS_320_20 appears to be a combination of both the TMS_320_10 and TMS_9995. For instance, the serial port in the TMS_320_20 and the TMS_9995 are very similar on operation. That is, they both can operate in byte mode or 16-bit word mode.

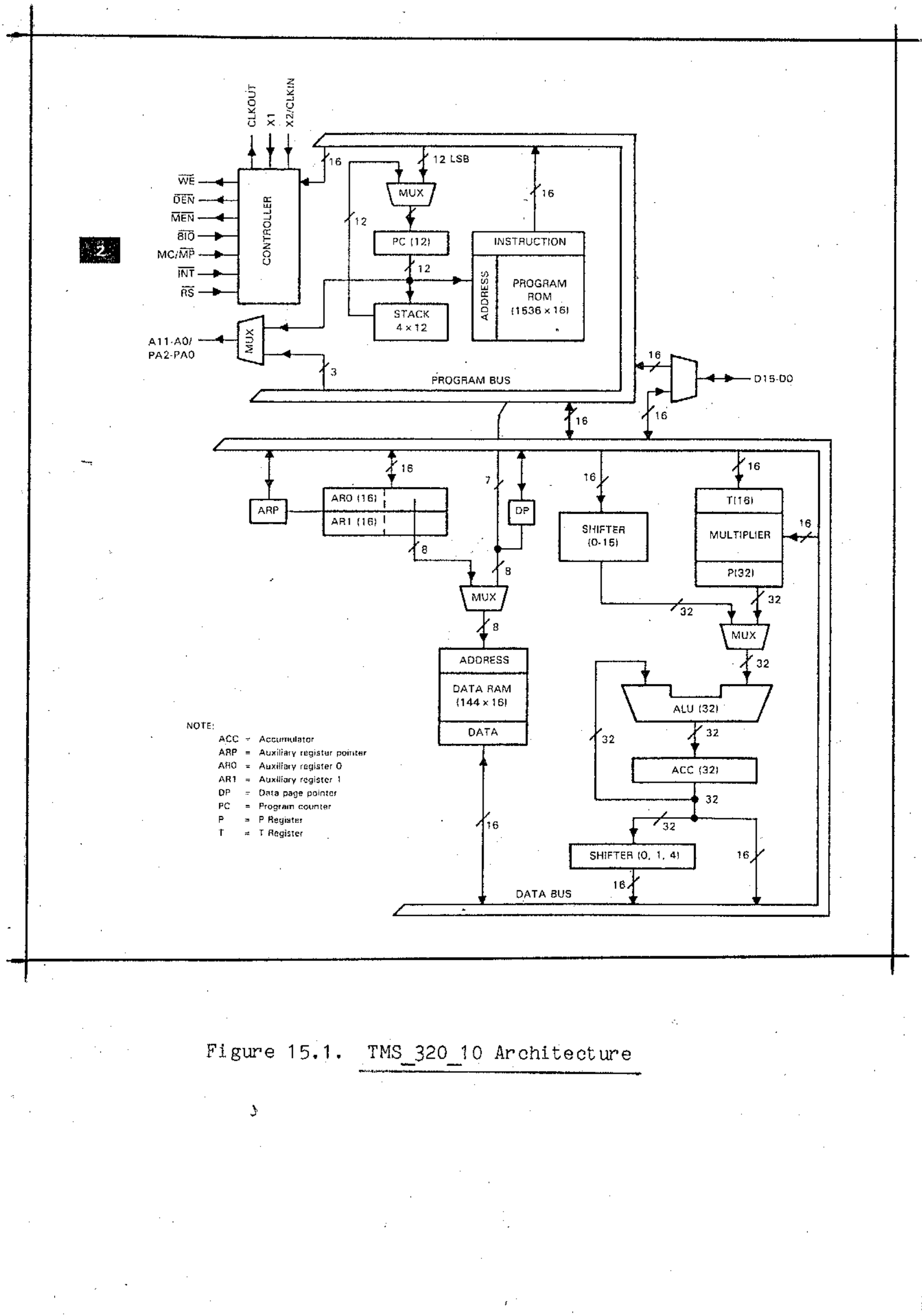
Many more operations are possible. Moving blocks of data is easily done in a few intstructions. A timer is in the chip that can be used for its serial port. There are strobe signals for the data, program and input and output space selection. This I.C. is easier to interface to the bus as there are bus request and grant lines. Also there is an enhanced and comprehensive instruction set.

The auto increment and decrement operations are more sophisticated, in the sense that an offset can put into the first of the 5 auxilliary registers and use it to automatically increment any of the other 4 registers. There is a repeat instruction that allows the looping of instructions as in coefficient multiplication for digital filters. Even a separate stack for pushing and pulling data memory.

15.2.3 The Architecture

In figure 1, there is a diagram showing the organisation of the internal hardware inside the I.C. This includes the separate data and program buses, the harware multiplier, the stack and the auxiliary registers. The TMS_320_20 architecture is an extrension of this, having more registers and memory.

P.T.O. for BLOCK DIAGRAM



NOTE:

ACC = Accumulator
ARP = Auxiliary register pointer
ARO = Auxiliary register 0
AR1 = Auxiliary register 1
DP = Data page pointer
PC = Program counter
P = P Register
T = T Register

Figure 15.1. TMS_320_10 Architecture

15.3 Commands for the EMC card interface

Possible commands for the Eurocard Micro_processor Controller Card are:

R n m	: read 'm'	16-bit words from the TMS_320_10 data memory starting from address 'n'.
C n d1 d2 d3	: change	the 16-bit words in the TMS_320_10 data memory starting at 'n', replacing with the data words: i.e. (n) <-- d1 (n+1) <-- d2 (n+2) <-- d3
D m	: display	the 'm' words read into the EMC card RAM. bug : doesn't stop displaying (goes on forever).
M	: return	to Lilbug monitor.

P.T.O. for appendix 4

15.4 Evaluation Monitor Features

The following is a list features the user may wish to use. Looking at the 'menu' of the EVM unit there were commands on it like: 'SD32', 'HELP', 'HX16', 'STATE', 'CLEAR', 'TABLE', 'UD16', 'HATP', 'DDM', 'DPM', 'MDM', 'DT', 'ST', 'CT', 'DB', 'SB' and 'CB'. You could

i. Display, set and clear

   a. breakpoints ("DB", "SB", "CB"), 8 of them,

   b. traces ("DT", "ST", "CT"), 6 of them.

ii. Modify data memory ("MDM") or Display program memory ("DPM").

iii. Display the contents of the Accumulator, the T register, and the P register in hexadecimal ("HATP"),

iv. With all the 320 registers

   a. clear them ("CLEAR"),

   b. display the contents ("STATE").

v. Do some number crunching like:

   a. converting from Hexadecimal to 32 bit Decimal,

      i. Hex. to Signed Dec. ("SD32"),

      ii. Hex. to Unsigned Dec. ("UD32"),

   b. converting from 16 bit Hex. to Signed of Unsigned Dec. ("SD16" and "UD16").

   c. Even the reverse: from Dec. to Hex.

      a. to 32 bit Hex. ("HX32"),

      b. to 16 bit Hex. ("HX16").

vi. "EC" ; sets the event counter - useful for looping. If you need to execute a loop a 1000 times you set this up instead of single stepping through. This is analogous to the trace function in Lilbug monitor for the Eurocard Micro_Processor controller in Chapter 7.

vii. Other things that affect the program such as executing ("EX"), Single Stepping ("SS") and Running ("RUN"),

viii. and last but not least the Help ("H") command the regurgitate the menu

15.4.1 Housing for Data Acquisition Board

ETO.

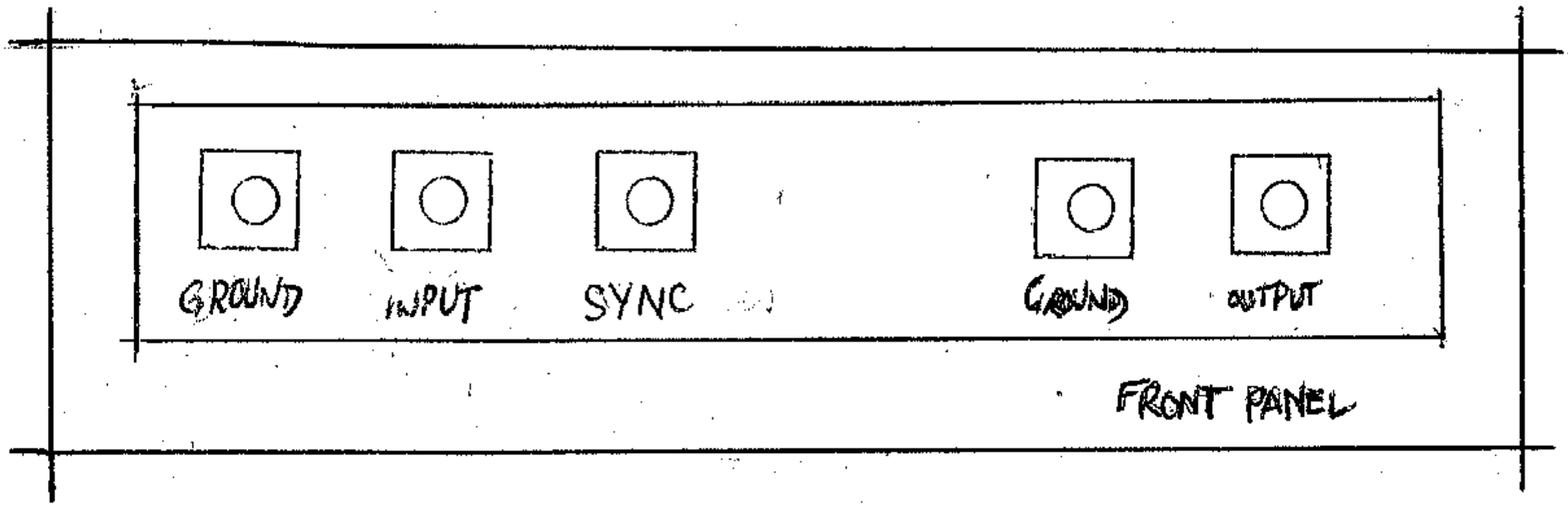


Figure 15.2. EVM system Housing

onto the screen.

P.T.O for appendix 5

APPENDIX 5

15.5 Program Listings

15.5.1 Master (EMC card)

15.5.2 Slave (TMS 320 10)

the assembler doesn't understand blank lines: need an asterisk as the first character of that line.

? protocol

```
*** module "main.s"
                        !
                        !       DMR's EUROCARD CONTROLLER (EM 550)
                        !
                        !       Written in Motorola 6800 assembler
                        !       using the UNSW Amsterdam Compiler Kit:
                        !       the assembler is "m00" and the conversion
                        !       program, "sld", into S1S9 format.
                        !
                        !       Interface Program Version 2.2
                        !
                        !                               Written by
                        !                             Bob Smith (21 Oct '85)
                        !
                        !       Not fully debugged (Week 13, 8 Nov '85)
                        !       Performs the control word, sends it out and waits
                        !       for the acknowledgement from the '320.
                        !       (See "lp2nd.t" in the appendix)
                        !
                        !       ADDRESSES
                        !
1700                    TMSmem  =       0x1700          ! last 256 btyes reserved for
                        !                                       128 TMS_320_10 words
F806                    getchar =       0xf806          ! input char from keyboard
F80F                    print   =       0xf80f          !  cr/lf then msg pointed to by the X reg
F8DF                    io      =       0xf8df          ! input/output routine
                        !
                        !       PIA     LOCATION in the memory map
                        !
E400                    ddra    =       0xe400
E401                    cra     =       0xe401
E402                    ddrb    =       0xe402
E403                    crb     =       0xe403
                        !
                        !       CONSTANTS
                        !
0038                    _cx2_hi =       0x38            ! for PIA control registers
0030                    _cx2_lo =       0x30
0003                    addrdgts =      3
0005                    datadgts =      5
0004                    EOT     =       4               ! end of text character for print
0005                    maxwords =      5
000D                    newline =       0xd             ! carriage return character
0003                    nwdgts =        3
                        !-------------------------------!
                        !                               !
                        !       MAIN    PROGRAM         !
                        !                               !
                        !-------------------------------!

                                .text
                                .org    0x1000          ! start of RAM

1000 CE 11 0E           main:   ldx     #words
1003 BD F8 0F                   bsr     print
1006 BF 13 2E                   sts     stacksave       ! save the stack pointer

1009 BE 13 2E           start:  lds     stacksave       ! restore stack
100C CE 10 FB                   ldx     #prompt
100F BD F8 0F                   bsr     print
```



```
1012 BD F8 06           fred0:  bsr     getchar
1015 81 20                      cmpa    #' '
1017 27 F9                      beq     fred0           ! skip blanks
1019 81 0D                      cmpa    #newline
101B 27 EC                      beq     start           ! restart on newline

101D 84 5F                      anda    #0x5f           ! convert to upper case
101F 81 52                      cmpa    #'R'
-------------------------------------------------------------------------------
1021 26 05                      bne     fred1
1023 BD 11 BA                   bsr     read            ! collect words from
1026 20 E1                      bra     start           ! the TMS_320_10

1028 81 43              fred1:  cmpa    #'C'
102A 26 05                      bne     fred2
102C BD 12 9C                   bsr     change          ! modify words in
102F 20 D8                      bra     start           ! the TMS_320_10

                        fred2:
1031 81 4D                      cmpa    #'M'            ! back to monitor
1033 26 01                      bne     fred3
1035 3F                         swi
                        fred3:
1036 81 44                      cmpa    #'D'
1038 26 16                      bne     illegal

103A CE 17 00                   ldx     #TMSmem         ! start of the display block
103D FF 13 24                   stx     dblock
1040 CE 17 80                   ldx     #0x1780
1043 FF 13 26                   stx     dblock+2        ! end of the display block
1046 CE 13 24                   ldx     #dblock         ! set up X reg to point into dblock
1049 C6 0E                      ldab    #0xE
104B BD F8 DF                   bsr     io              ! ref: pJ-24, and pJ-6
104E 20 B9                      bra     start
                        illegal:
1050 CE 11 5D                   ldx     #wrfmt
1053 BD F8 0F                   bsr     print
1056 20 B1                      bra     start

                        !-------------------------------!
                        !       ERROR   MESSAGES        !
                        !-------------------------------!
1058-1072               errmsg: .ascii  "must be between 0 and 127\n\r"
1073 04                         .byte   EOT
1074-108E               out_of_range:   .ascii  "address + nwords is > 127\r\n"
108F 04                         .byte   EOT
1090-10A5               panic:  .ascii  "panic: invalid digit\n\r"
10A6 04                         .byte   EOT
10A7-10B9               too_big:        .ascii  "number is too big\n\r"
10BA 04                         .byte   EOT
10BB-10CC               too_many:       .ascii  "too many numbers\n\r"
10CD 04                         .byte   EOT
                        !
10CE-10E7               eoln:   .ascii  "\n\runexpected end of line\n\r"
10E8 04                         .byte   EOT
```


MASTER'S PROGRAM LISTING

```
10E9-10F9
10FA 04
10FB-10FC
10FD 04
10FE-110C
110D 04
110E-1135
1136-115C
115D-1183
1184-11B8
11B9 04
```

```
!-------------------------------------------------------------------------!
!       USEFUL  MESSAGES                                                  !
!-------------------------------------------------------------------------!
c_finish:       .ascii  "change finished\n\r"
        .byte   EOT
prompt: .ascii  "* "
        .byte   EOT
r_finish:       .ascii  "read complete\n\r"
        .byte   EOT
words:  .ascii  "          Interface Program\n\rready:\n\r\n\r"
        .ascii  "m          - exit to Lilbug Monitor\n\r\n\r"
wrfmt:  .ascii  "Format:   r   address   nwords     OR\n\r"
        .ascii  "          c   address   data[1] data[2] ... data[5]\n\r"
        .byte   EOT
!
!-------------------------------------------------------------------------!
!                                                                         !
!       READ    FROM    TMS_320_10                                        !
!                                                                         !
!-------------------------------------------------------------------------!
!
!       Read 'nwords' starting from 'addr' in the '320.
!       Checks for out of range: trying to read past the end of memory.
!
!
!       Read Format:
!               r  address   nwords
!
!       eg      r 20 5          "collects contents of dma 20, 21, .. 25"
read:
11BA 86 03              ldaa    #addrdgts       ! read in the ADDRESS
11BC B7 13 2A           staa    ndgts
11BF BD 13 54           bsr     get_number
11C2 81 0D              cmpa    #newline
11C4 27 11              beq     r_err3          ! unexpected end of line
11C6 BD 12 4E           bsr     one_byte        ! test of number is one byte & + ve
11C9 25 12              bcs     r_ok            ! carry set -> one byte.
!
!       err msgs
!
r_er1:
11CB CE 10 58           ldx     #errmsg
11CE 7E F8 0F           bra     print           ! make prmsg do the rts back to
                                                ! the main loop instead of read.
r_er2:
11D1 CE 10 74           ldx     #out_of_range
11D4 7E F8 0F           bra     print
r_err3:
11D7 CE 10 CE           ldx     #eoln
11DA 7E F8 0F           bra     print
r_ok:
11DD B6 13 2D           ldaa    number+1
11E0 B7 13 18           staa    addr            ! valid address

11E3 BD 13 54           bsr     get_number      ! read in the Number of WORDS
11E6 8D 66              bsr     one_byte
11E8 24 E1              bcc     r_er1
11EA B6 13 2D           ldaa    number+1        ! xfr the number read into nwords
11ED B7 13 2B           staa    nwords          ! valid number of words
```



```
11F0 BB 13 18           adda    addr
11F3 2B DC              bmi     r_er2           ! address + nwords > 127 ?

11F5 7C 13 29           inc     rcmode          ! set flag for read option

11F8 8D 08              bsr     send_crtl
11FA 8D 60              bsr     collect

11FC CE 10 FE           ldx     #r_finish
11FF 7E F8 0F           bra     print
!       rts
!
!-------------------------------------------------------------------------!
!                                                                         !
!       SEND    CONTROL         WORD    TO TMS_320_10                     !
!                                                                         !
!-------------------------------------------------------------------------!
send_crtl:
1202 86 38              ldaa    #_cx2_hi        ! output enable the latches
1204 B7 E4 01           staa    cra
1207 B7 E4 03           staa    crb             ! make sure CX2 = high to disable the

120A 4F                 clra            !       first set the output direction
120B B7 E4 01           staa    cra
120E B7 E4 03           staa    crb
1211 C6 FF              ldab    #0xff           ! all pins = output
1213 F7 E4 00           stab    ddra
1216 F7 E4 02           stab    ddrb

1219 86 04              ldaa    #4              ! select the output register
121B B7 E4 01           staa    cra
121E 8B 20              adda    #0x20           ! cb2 as write strobe with cb1 restore
1220 B7 E4 03           staa    crb
make:                                           ! CB2 initially LOW (at reset)
1223 B6 13 2B           ldaa    nwords
1226 5F                 clrb
1227 F1 13 29           cmpb    rcmode
122A 27 02              beq     send

122C 8A 80              oraa    #0x80           ! msbit = 1, crtl word == 'read'

send:
122E F6 13 18           ldab    addr            !       make the control word
1231 F7 E4 00           stab    ddra
1234 B7 E4 02           staa    ddrb            !       output the 16 bits
1237 B6 E4 00           ldaa    ddra            ! clear IRQ flags (bit 7,6)
123A B6 E4 02           ldaa    ddrb
123D 86 30              ldaa    #_cx2_lo        ! send 'DATA AVAILABLE' to '320
123F B7 E4 03           staa    crb             !               force CB2 to go low
!       acca(msbyte) thru ddrb (msbyte) -> cb2 (-)ve edge
!
1242 F6 E4 03 s_ack1:   ldab    crb             ! wait for ACK from '320
1245 59                 rolb                    ! -ve edge on CB1
1246 24 FA              bcc     s_ack1
!                               bpl     s_ack1          ! i.e. the cb1 restore

1248 86 38              ldaa    #_cx2_hi        ! CB2 = high ==> o/p disabled.
124A B7 E4 03           staa    crb
124D 39                 rts
```

```
!
!-------------------------------------------------------!
!                                                       !
!       TEST    IF NUMBER IS    ONE BYTE        LONG    !
!                                                       !
!-------------------------------------------------------!
!
!                                       ---------------
!       number organisation:            number, number+1
!                                       ---------------
!                                       MSB     LSB
!                                       ---------------
!
one_byte:
124E 7D 13 2C           tst     number
1251 27 02              beq     clearc
one_1:
1253 0C                 clc                     ! clear carry -> 16 bit #, or -ve
1254 39                 rts                     ! number is no good anyway.
clearc:
1255 7D 13 2D           tst     number+1
1258 2B F9              bmi     one_1
125A 0D                 sec                     ! is a one byte number & +ve
125B 39                 rts
!
!-------------------------------------------------------!
!                                                       !
!       COLLECT the data FROM the TMS_320_10            !
!                                                       !
!-------------------------------------------------------!
!
!       Have the Eurocard Controller ( EM 550 ) store the 16 bit words
!       from the TMS_320_10 starting at the TMSmem address.
!
!                                       ---------------
!       Format:         TMSmem  +0:     MS byte  LS byte
!                                       ---------------
!                         "     +2:     MS byte  LS byte
!                                       ---------------
!                       etc.
!
collect:
125C 4F                 clra                    ! set up PIA for input
125D B7 E4 01           staa    cra             ! access data direction register
1260 B7 E4 03           staa    crb
1263 B7 E4 00           staa    ddra            ! all pins = input
1266 B7 E4 02           staa    ddrb
1269 86 04              ldaa    #4
126B B7 E4 01           staa    cra             ! access output register
126E B7 E4 03           staa    crb

1271 B6 13 2B           ldaa    nwords
1274 B7 13 19           staa    count


1277 CE 17 00           ldx     #TMSmem         ! define start of input buffer

                !               poll bit 7 in CRA for CA1 active
                !               transition, i.e. a word has been
                !               latched in the bi-diretional buffers.
```



```
pollread:
127A B6 E4 01           ldaa    cra
127D 49                 rola
127E 24 FA              bcc     pollread
!                       bpl     pollread        ! get another 16 - bit word.

getw:
1280 86 30              ldaa    #_cx2_lo        ! enable latch output for EC to read.
1282 B7 E4 01           staa    cra
1285 B6 E4 02           ldaa    ddrb            ! get the word
1288 A7 00              staa    0,x             ! compiler won't allow ',x': wants '0,x
128A B6 E4 00           ldaa    ddra
128D A7 01              staa    1,x
128F 08                 inx                     ! 2 bytes inc for TMSmem table
1290 08                 inx

1291 86 38              ldaa    #_cx2_hi        ! tri-state o/p & 'send ACK' to '320.
1293 B7 E4 01           staa    cra

1296 7A 13 19           dec     count
1299 2E DF              bgt     pollread
129B 39                 rts
!-------------------------------------------------------!
!                                                       !
!       MODIFY  DATA    MEMORY  IN THE TMS_320_10       !
!                                                       !
!-------------------------------------------------------!
!
!       Change the 'nwords' consecutive memory contents
!       starting from 'addr'.   Also checks if  user tries
!       to change more than the end of memory.
!
!
!       change format:
!                               c   address   data[1]   data[2]   ...   data[5]
!       eg
!                               c 15 82         "dma 15 now to equal 82 decimal
!
change:
129C 86 03              ldaa    #addrdgts
129E B7 13 2A           staa    ndgts

12A1 BD 13 54           bsr     get_number      ! get the ADDRESS
12A4 81 0D              cmpa    #newline
12A6 27 16              beq     ch_4
12A8 8D A4              bsr     one_byte
12AA 25 18              bcs     ch_0            ! comments as for 'read'

!
!       err msgs
!
ch_1:
12AC CE 10 58           ldx     #errmsg
12AF 7E F8 0F           bra     print           ! go back to main
ch_2:
12B2 CE 10 BB           ldx     #too_many
12B5 7E F8 0F           bra     print
ch_3:
12B8 CE 10 74           ldx     #out_of_range
12BB 7E F8 0F           bra     print
```

```
                ch_4:
12BE CE 10 CE           ldx     #eoln
12C1 7E F8 0F           bra     print
                ch_0:
12C4 B6 13 2D           ldaa    number+1        ! define the address
12C7 B7 13 18           staa    addr
12CA CE 13 1A           ldx     #databuf        ! set up output buffer for 'send_data'
12CD FF 13 30           stx     tempx
12D0 7F 13 2B           clr     nwords
12D3 86 05              ldaa    #datadgts       ! set max on # of digits to get.
12D5 B7 13 2A           staa    ndgts

                !                               ----------------
                !       number organisation:    number, number+1
                !                               ----------------
                !                               MSB   LSB
                !                               ----------------
                !
                get_dat:
12D8 BD 13 54           bsr     get_number      ! get data( nwords++ )

12DB FE 13 30           ldx     tempx
12DE F6 13 2C           ldab    number
12E1 E7 00              stab    0,x             ! xfr ms byte into reserved buffer
12E3 08                 inx
12E4 F6 13 2D           ldab    number+1
12E7 E7 00              stab    0,x             ! xfr ls byte
12E9 08                 inx
12EA FF 13 30           stx     tempx

12ED 7C 13 2B           inc     nwords          ! nwords <-- nwords + 1

12F0 F6 13 2B           ldab    nwords
12F3 C1 05              cmpb    #maxwords
12F5 2E BB              bgt     ch_2            ! too many

12F7 FB 13 18           addb    addr
12FA 2B BC              bmi     ch_3            ! in range ( addr + nwds < 128 ) ??

12FC 81 0D              cmpa    #newline        ! insert number just read in then
12FE 27 02              beq     ch_continue     ! tst for newline

1300 20 D6              bra     get_dat


                ch_continue:
1302 7F 13 29           clr     remode          ! mode = change
1305 BD 12 02           bsr     send_crtl
1308 BD 13 34           bsr     send_data

130B CE 10 E9           ldx     #c_finish
130E 7E F8 0F           bra     print
                !       rts
```



```
                !
                !************    to be put into RAM ************
                !
                !       VARIABLES in units of bytes.
                !
                        .data
1318-1318       addr:   .space 1
1319-1319       count:  .space 1
131A-1323       databuf:        .space 10       ! five - 16 bits words
1324-1327       dblock: .space  4
1328-1328       minus:  .space 1
1329-1329       remode: .space 1                ! one byte of memory
132A-132A       ndgts:  .space 1
132B-132B       nwords: .space 1
132C-132D       number: .space 2
132E-132F       stacksave:      .space 2
1330-1331       tempx:  .space 2                ! one word = 16 bits.
1332-1333       tmp_no: .space 2                ! for the multiply routine
                !
                !************    to be put into RAM ************
                !
                !-------------------------------------------------!
                !                                                 !
                !       PRINT a MESSAGE                           !
                !                                                 !
                !-------------------------------------------------!
                !prmsg:
                !       jmp     print
                !       rts
                !
                !-------------------------------------------------!
                !                                                 !
                !       SEND the DATA out to the TMS_320_10       !
                !                                                 !
                !-------------------------------------------------!
                send_data:
1334 B6 13 2B           ldaa    nwords
1337 B7 13 19           staa    count

                pollwrite:
133A B6 E4 01           ldaa    cra             ! wait for ACK from TMS_320_10
133D 2A FB              bpl     pollwrite

                inject:
133F CE 13 1A           ldx     #databuf
1342 E6 00              ldab    0,x
1344 F7 E4 00           stab    ddra            ! can use breakpoints
1347 08                 inx
1348 E6 00              ldab    0,x
134A F7 E4 02           stab    ddrb            ! cause cb2 to go low after data
134D 08                 inx                     !       is put on the bus.
134E 7A 13 19           dec     count
1351 26 E7              bne     pollwrite
1353 39                 rts
```

text

After the main program is in the EPROM.

```
                        !
                        !-----------------------------------------!
                        !                                         !
                        !       GET     a decimal       NUMBER    !
                        !                                         !
                        !-----------------------------------------!
                        !
                        !       returns the number in 'number, number+1'
                        !       and a space or newline in ACCA.
                        !
                        get_number:
1354 7F 13 2C                   clr     number
1357 7F 13 2D                   clr     number+1
135A 7F 13 28                   clr     minus
135D 5F                         clrb

135E BD F8 06           gn_0:   bsr     getchar         ! sample first character
1361 81 20                      cmpa    #' '            ! skip leading blanks
1363 27 F9                      beq     gn_0
1365 81 2D                      cmpa    #'-'            ! '-' ??
1367 26 1B                      bne     gn_1
1369 7C 13 28                   inc     minus
136C 20 F0                      bra     gn_0

                        panic1:
136E CE 10 A7                   ldx     #too_big
1371 7E F8 0F                   bra     print
                        error:
1374 81 0D                      cmpa    #newline
1376 26 06                      bne     gn_err
1378 CE 10 CE                   ldx     #eoln
137B 7E F8 0F                   bra     print
                        gn_err:
137E CE 10 90                   ldx     #panic
1381 7E F8 0F                   bra     print
                        gn_1:
1384 81 30                      cmpa    #'0'            ! first character must be a digit
1386 2D EC                      blt     error
1388 81 39                      cmpa    #'9'
138A 2E E8                      bgt     error

138C 20 03                      bra     gn_3            ! valid character is a digit
                        gn_2:
138E BD F8 06                   bsr     getchar         ! get another character
                        gn_3:
1391 81 30                      cmpa    #'0'
1393 2D 1D                      blt     gn_rtn          ! error, so retore the char.
1395 81 39                      cmpa    #'9'
1397 2E 19                      bgt     gn_rtn          ! fix it up, return.
                        valid:
1399 80 30                      suba    #'0'            ! remove the offset of ascii zero.
139B 8D 3B                      bsr     mul10           ! multiply the number by 10 as soon
                                                        ! as the next char is a digit.
139D 5C                         incb

139E BB 13 2D                   adda    number+1        ! update LS byte
13A1 B7 13 2D                   staa    number+1
13A4 24 05                      bcc     gn_4
13A6 7C 13 2C                   inc     number          ! inc MS byte
13A9 29 C3                      bvs     panic1          ! if ovrlfw, print msg & abort
```



```
                        gn_4:
13AB F1 13 2A                   cmpb    ndgts
13AE 2E BE                      bgt     panic1
13B0 20 DC                      bra     gn_2
                        gn_rtn:
13B2 81 20                      cmpa    #' '            ! space or newline terminating the numl
13B4 27 06                      beq     ok
13B6 81 0D                      cmpa    #newline
13B8 27 02                      beq     ok
13BA 20 C2                      bra     gn_err          ! 'bad data found on integer read'
                        ok:
13BC 36                         psha
13BD 37                         pshb
13BE 7D 13 28                   tst     minus           ! minus sign ?
13C1 27 12                      beq     gn_home

13C3 B6 13 2C                   ldaa    number          ! negate the number
13C6 F6 13 2D                   ldab    number+1
13C9 43                         coma
13CA 53                         comb
13CB CB 01                      addb    #1
13CD 89 00                      adca    #0
13CF B7 13 2C                   staa    number
13D2 F7 13 2D                   stab    number+1
                        gn_home:                        ! returns with a space or newline
13D5 33                         pulb
13D6 32                         pula
13D7 39                         rts
                        !
                        !-----------------------------------------!
                        !                                         !
                        !       MULTIPLY        BY      10        !
                        !                                         !
                        !-----------------------------------------!
                        !
                        mul10:
13D8 36                         psha                    ! save both accumulators
13D9 37                         pshb
13DA F6 13 2D                   ldab    number+1        ! ls byte
13DD B6 13 2C                   ldaa    number          !       ms byte
13E0 58                         aslb                    ! 2N
13E1 49                         rola                    ! move carry of lsbyte into
                                                        ! bit 0 of ms byte.
13E2 B7 13 32                   staa    tmp_no
13E5 F7 13 33                   stab    tmp_no+1
13E8 58                         aslb                    ! 4N
13E9 49                         rola
13EA 58                         aslb                    ! 8N
13EB FB 13 33                   addb    tmp_no+1                ! 10N
13EE B9 13 32                   adca    tmp_no          ! add with carry ( A <- tmp + C + A )

13F1 F7 13 2D                   stab    number+1
13F4 B7 13 2C                   staa    number

13F7 33                         pulb                    ! restore accumulators
13F8 32                         pula
13F9 39                         rts                     ! home
```

```
!
! equiv if accd accessible
!       ldd     number
!       asld
!       std     tmp_no
!       asld
!       asld
!       add     tmp_no
!       std     tmp_no
!       rts
```

```
*** symbol table for "main.s"
 abs 0004 EOT        abs 1700 TMSmem     abs 0038 _cx2_hi    abs 0030 _cx2_lo
 dat 1318 addr       abs 0003 addrdgts   org 10E9 c_finish   org 12C4 ch_0
 org 12AC ch_1       org 12B2 ch_2       org 12B8 ch_3       org 12BE ch_4
 org 1302 ch_conti   org 129C change     org 1255 clearc     org 125C collect
 dat 1319 count      abs E401 cra        abs E403 crb        dat 131A databuf
 abs 0005 datadgts   dat 1324 dblock     abs E400 ddra       abs E402 ddrb
 org 10CE eoln       org 1058 errmsg     dat 1374 error      org 1012 fred0
 org 1028 fred1      org 1031 fred2      org 1036 fred3      org 12D8 get_dat
 dat 1354 get_numb   abs F806 getchar    org 1280 getw       dat 135E gn_0
 dat 1384 gn_1       dat 138E gn_2       dat 1391 gn_3       dat 13AB gn_4
 dat 137E gn_err     dat 13D5 gn_home    dat 13B2 gn_rtn     org 1050 illegal
 dat 133F inject     abs F8DF io         org 1000 main       org 1223 make
 abs 0005 maxwords   dat 1328 minus      dat 13D8 mul10      dat 132A ndgts
 abs 000D newline    dat 132C number     abs 0003 nwdgts     dat 132B nwords
 dat 13BC ok         org 1253 one_1      org 124E one_byte   org 1074 out_of_r
 org 1090 panic      dat 136E panic1     org 127A pollread   dat 133A pollwrit
 abs F80F print      org 10FB prompt     org 11CB r_er1      org 11D1 r_er2
 org 11D7 r_err3     org 10FE r_finish   org 11DD r_ok       dat 1329 remode
 org 11BA read       org 1242 s_ack1     org 122E send       org 1202 send_crt
 dat 1334 send_dat   dat 132E stacksav   org 1009 start      dat 1330 tempx
 dat 1332 tmp_no     org 10A7 too_big    org 10BB too_many   dat 1399 valid
 org 110E words      org 115D wrfmt
```

```
File : lp2nd.t
   1    >
   2    *       2nd order lowpass filter in TMS_320_10 code.
   3    *       Butterworth type: Cut-off Frequency = 1 kHz.
   4    *       Sampling frequency = 20 kHz.
   5    *
   6    *                                       Written by
   7    *                                               Tom Millett (before Jan '85)
   8    *
   9    *       Commented by                            Bob Smith (5 Sep '85)
  10    *       Modified for use as a test program,     Bob Smith (16 Sep '85)
  11    *       demonstrating the developed protocol
  12    *       with the DMR's Eurocard (EM 550)
  13    *       and the TMS_320_10 Digital Signal Processor.
  14    *
  15    admask  equ     0               * RESERVED data memory for protocol.
  16    nwmask  equ     1               * temporary stack for saving environment
  17    TAR0    equ     2               * once protocol is executing.
  18    TAR1    equ     3
  19    TACCH   equ     4
  20    TACCL   equ     5
  21    *
  22    ADDR    equ     6               * words for the address, control and data.
  23    CRTL    equ     7
  24    DATA    equ     8
  25    NWORDS  equ     9
  26    UNITY   equ     10
  27    *
  28    AR0     equ     0               * define LABELS for readability
  29    AR1     equ     1
  30    PORT0   equ     0
  31    PORT7   equ     7
  32    ONE     equ     1
  33    ZERO    equ     0
  34    *
  35    *
  36    *                               On chip RAM locations (data memory)
  37    x0      equ     11              * used for the program.
  38    x1      equ     12
  39    x2      equ     13
  40    y0      equ     14
  41    y1      equ     15
  42    y2      equ     16
  43    ovflw   equ     17              * for safety from the 'dmov' instruction.
  44    a0      equ     18
  45    a1      equ     19
  46    a2      equ     20
  47    b1      equ     21
  48    b2      equ     22
  49    *
  50    *
  51    *
  52            aorg    2000
  53    da0     data    82              * coefficients set up in prog. mem.
  54    da1     data    164             *       they are Q12 numbers
  55    da2     data    82
  56    db1     data    6394
  57    db2     data    -2627
```



```
  58    *-------------------------------------------------------*
  59    *                                                       *
  60    *                    START OF PROGRAM                   *
  61    *                                                       *
  62    *-------------------------------------------------------*
  63    *
  64            aorg    0
  65    reset   b       setup           * VECTORS
  66    intr    b       getin
  67    *
  68    *
  69            aorg    10              * START of program.
  70    *
  71    setup   dint                    *       disable further interrupts
  72            lack    ONE             * define UNITY
  73            sacl    UNITY
  74            lack    >7f
  75            sacl    admask          * define the address and number-of-words
  76            lac     admask,8        * masks for the protocol
  77            sacl    nwmask
  78    *
  79            zac                     * initialise relevant RAM locations
  80            sacl    x0              * for application program.
  81            sacl    x1
  82            sacl    x2
  83            sacl    y0
  84            sacl    y1
  85            sacl    y2
  86    *
  87    *                                 Because of the Harvard architecture need
  88    *                                 to get coefficients from program memory
  89            lark    AR0,5           into data memory.
  90            lark    AR1,a0          * instead of 18, makes life easier.
  91            larp    ZERO            *               page zero of RAM
  92            lt      UNITY           * offset address for moving data
  93            mpyk    2000
  94            pac
  95    consin  larp    ONE
  96            tblr    *+,PORT0
  97            add     UNITY
  98            banz    consin          * loop until all coefficients are in.
  99    *
 100    start   in      x0,PORT0        *       kick off ADC for FIRST input.
 101    *
 102    rerun   zac                     * clear for processing.
 103            mpyk    ZERO
 104            ltd     y2              * unit delay implementation.
 105            mpy     b2
 106            ltd     y1
 107            mpy     b1
 108            ltd     x2
 109            mpy     a2
 110            ltd     x1
 111            mpy     a1
 112            eint                    * enable interrupts
 113    *
 114    wait    bioz    fetch           * get the control word from the EUROCARD
 115    *                                 CONTROLLER. Note BIO pin is pulled up by
 116    *                                 an internal resistor.
 117    *
 118            b       wait            * hang about until a/d complete,
 119    *                                 i.e. wait for an interrupt.
```

SLAVE's PROGRAM LISTING

```
*
*
        nop                     * some space for bkpts
        nop
        nop
        nop
getin   in      x0,PORT0        * get one sample from port 0.
        dint
*
cont    ltd     x0              * tack on the 'a0 * x0' term.
        mpy     a0
        apac
        sach    y0,4            * only have a 12-bit DAC chip, so throw
        dmov    y0              * away the 4 least significant bits.
        out     y0,PORT0        *       OUTPUT processed sample to port 0.
        b       rerun
*
*
*-------------------------------------------------------*
*                                                       *
*                START   OF      PROTOCOL               *
*                                                       *
*       between the TMS_320_10 and the EM_550           *
*       micro-processor controller card.                *
*                                                       *
*-------------------------------------------------------*
*
fetch   dint                    * disable the maskable interrupt
        in      CRTL,PORT7      *       get control word from port 7
*
        sar     AR0,TAR0        * save the environment, i.e the auxiliary
        sar     AR1,TAR1        * registers and the accumulator.
        sach    TACCH,0
        sacl    TACCL
*                                 pick out the address
        lac     CRTL,0
        and     admask
        sacl    ADDR
        lac     CRTL,0          * pick out nwords
        and     nwmask          * extract the number-of-words to transfer.
        sacl    NWORDS
*                                 test bit 15 : a 'read' or 'change' instruction.
        lac     UNITY,15
tstb15  and     CRTL
        bz      change          * is zero, go and change data memory
*
*-------------------------------------------------------*
*                                                       *
*                       READ                            *
*                                                       *
*-------------------------------------------------------*
*
*       Master wants to collect 'NWORDS' words starting from
*       address 'ADDR' in data memory.
*
READ    nop
        lar     AR0,NWORDS      * output NWORDS starting from address given.
        lar     AR1,ADDR
*
output  larp    1
        out     *+,PORT7,AR0
```



```
*
rpoll   bioz    dec             * wait for EC to collect the word.
        b       rpoll
*
dec     banz    output
        b       HOME            * pack up and go home
*
*-------------------------------------------------------*
*                                                       *
*                       CHANGE                          *
*                                                       *
*-------------------------------------------------------*
*
CHANGE  nop
cpoll   bioz    go              * wait for first data word to arrive
        b       cpoll
*
go      larp    1
        lac     nwords,0        * set up counter and start address
        lark    AR0,addr
*
getdat  in      data,port7      * collect data from port7
        larp    0
*       sacl    *+,AR1          * indirect storage with auto-increment, AR0
*                                 contains the data memory address
        sacl    *+
        larp    AR1             * this works but SACL   *+,AR1 doesn't
*
        banz    getdat
*
*
*       got all data    words   in to the dma
*
*-------------------------------------------------------*
*                                                       *
*                       HOME                            *
*                                                       *
*-------------------------------------------------------*
*
HOME    lar     AR0,TAR0        * restore aux. reg. & acc.
        lar     AR1,TAR1
        lac     TACCH,15        * the high 16-bit word in the acc.
        adds    TACCL
        eint                    * enable interrupts
*
        b       wait            * continue with the rest of the program.
        end                     * <for assembler>
<
```

APPENDIX 6

15.6 Test Programs

The following are some test programs. But first, a definition of the "S record" is presented. Then, a sample program to test the status lights on the Eurocard Micro processor Controller.

15.6.1 S record format

This is how the S record is constructed. Over the years in the course, nowhere has a clear-cut definition been stated, most people assumed you knew. The following are the components of the S record.

```
S1091010865ABDF8093FF9

S1      byte    start    instructions     checksum
        count   address

here there are nine bytes (09), start address = 1010 hex
and the checksum = F9
```

The program listing was:

```
*** module "outchtst.s"
                !       A short RAM testting program to Output ONE character.
                !       Uses the DMR's Eurocard Microprocessor Controller.
                !
                !       An attempt to use a Lilbug routine: 'outch1'
                !
                !                                              Written by
                !                                              Bob Smith (17 July '85)
    F809        outch1  =       0xf809
                        .org    0x1010          ! start of RAM program
1010 86 5A              ldaa    #'Z'
1012 BD F8 09           jsr     outch1
1015 3F                 swi
```

```
*** symbol table for "outchtst.s"
 abs F809 outch1
```

15.6.2 MC 6803 port 1 testing

This program allows the user if the MC 6803, and the decoding I.C.'s are operational.

P.T.O. for listing

```
*** module "p1countup"
                        !
                        !       Test program #2:
                        !
                        !   To count up and display on Status Lights.
                        !
                        !   Note: Lights are hardwired to port 1 in the MC_6803.
                        !
                        !                                   Written by
                        !                                   Bob Smith (30 July '85)
                        !
      0000      p1ddr   =       0x00        ! ddr
      0002      p1d     =       0x02        ! data register
      0000      zero    =       0           ! zero for the display (+)ve logic
      00FF      output  =       0xff        ! output state
                !
                        .org    0x1000      ! ram starting address
1000 C6 FF              ldab    #output
1002 F7 00 00           stab    p1ddr       !       all pins = output
1005 86 00      go:     ldaa    #zero       !       initialise to display zero
1007 B7 00 02           staa    p1d         ! start at 00

100A 4C         z:      inca                ! count from 00 to FF with a delay
100B BD 10 13           jsr     delay
100E B7 00 02           staa    p1d
1011 20 F7              bra     z           !       repeat count sequence
                !
1013 CE FF FF   delay:  ldx     #0xffff     ! of say 1 sec
1016 09         do:     dex
1017 8C 00 00           cpx     #0          ! waste time instr.
101A 26 FA              bne     do
101C 39                 rts
```

```
*** symbol table for "p1countup"
 org 1013 delay     org 1016 do        org 1005 go        abs 00FF output
 abs 0002 p1d       abs 0000 p1ddr     org 100A z         abs 0000 zero
```

15.6.3 Memory Test for TMS 320 10

This is a sample program that shows the user how the table read (TBLR) and table write (TBLW) instructions are used within the TMS_320_10 code.

P.T.O.

APPENDIX 6

File : testmem.t

```
   1    >
   2    *                    Test Program for Understanding
   3    *
   4    *        To put a number into a memory location and then try to
   5    *        alter it via the monitor. Note, this demonstrates both data
   6    *        and program memory access. When READing INto the data memory, a
   7    *        'tblr' is done. When WRITing OUT, a 'tblw'. Note, the tblw is
   8    *        analogous to the out instruction.
   9    *
  10    *
  11    *                                          Written by
  12    *                                          Bob Smith (12 Sep '85)
  13    *
  14    *        MUST use 'EX' with breakpoints for the TMS_9995's copy
  15    *        of the TMS_320_10 data memory to be updated.
  16    *                                  OR
  17    *        use 'RUN'. This will automatically update the TMS_9995's copy of
  18    *        the data memory. After this you can't use breakpoints: unless you
  19    *        do a 'hard reset' and  assemble the source program again.
  20    *
  21    *        Now it is possible to INSPECT the data and program memories,
  22    *        via,    1: DDM or MDM
  23    *                2: DPM or MPM
  24    *
  25    *        WHAT TO EXPECT:         top     middle  bottom  >64     >C8
  26    *                memory:         dma3    dma5    dma7    pma100  pma200
  27    *                contents:       7       6       4       7       6
  28    *
  29    ARO     equ     0               * data memory ( TMS_320_10 internal RAM )
  30    top     equ     3               * dummy locations
  31    middle  equ     5
  32    bottom  equ     7
  33    *
  34    *
  35            aorg    0
  36    reset   b       fill
  37    *
  38            aorg    10              * Philosophically:
  39    *                                       first 7 locations of program memory
  40    *                                       are reserved for internal use:
  41    *                                       e.g.    interrupts.
  42    *
  43    fill    larp    0               * PUT A NUMBER INTO THE DATA MEMORY
  44            lark    AR0,7           *       here, put 7 into dma"top"
  45            sar     AR0,top         *       via the auxiliary register.
  46    *
  47    *
  48            lack    4
  49            sacl    bottom          *       via the accumulator
  50    *
  51    *                               COPY CONTENTS OF DATA MEM. INTO PROGRAM MEM.
  52            lack    100             *       put contents of dma5
  53            tblw    top             *       into pma100 ( i.e >64 ).
  54            nop                     * for the brkpt.
  55    *                               COPY CONTENTS OF PROGRAM MEM. INTO DATA MEM.
```

P.T.O

```
56    * Can you do an 'lack da0' somehow ??
57            lack    200             * i.e. >C8
58            lark    AR0,middle      *       copy pma200 contents into dma"middle"
59            tblr    *               *       via the auxiliary register.
60            nop                     * for the brkpt.
61    wait    b       wait
62    *
63            aorg    200
64    da0     data    6
65            end
66    <
```

P.T.O. for more test programs:

e : get16+bio.t

```
1    >
2    *               Test program 320.1
3    *
4    *       to read in a 16-bit control word from the '320 data bus.
5    *       also uses the bio pin for the read.
6    *
7    *                                       Written by
8    *                                       Bob Smith (28 Oct '85)
9    crtl    equ     7
0    port7   equ     7
1    *
2    reset   b       main
3    int     b       main
4    *
5    main    bioz    get
6            b       main
7    *
8    get     in      crtl,port7
9            lac     crtl,0
0    ever    nop
1            nop
2            b       ever
3    *
4            end
5    <
```

e : getput16.t

```
1    >
2    *               Test program 320.3
3    *
4    *       To read IN a 16-bit control word from the '320 data bus
5    *       and the write it back OUT to the master (EM 550).
6    *               Also uses the bio pin for the read.
7    *
8    *                                       Written by
9    *                                       Bob Smith (7 Nov '85)
0    crtl    equ     7
1    port7   equ     7
2    *
3    reset   b       main
4    int     b       main
5    *
6    main    bioz    get
7            b       main
8    *
9    get     in      crtl,port7
0            lac     crtl,0          * reflect the result immediately.
1    *
2            nop
3            out     crtl,port7      * throw it back out
4    ever    nop
5            nop
6            b       ever
7    *
8            end
9    <
```


File : intodma.t

```
1    >
2            aorg  20
3    start   lack    22              * put the number 22 into (dma=1)
4            sacl    1
5            b       start
6            end
7    <
```

File : intopma.t

```
1    >
2    *                                       Test Program
3    *
4    *       To write out the contents of data memory 7
5    *       into program memory 40 forever.
6    *               Contents of dma7 = 9. Also 40 is >28.
7    *
8    *       Convention:     for     tblw    from data to program memory.
9    *                       for     tblr    from program to data memory.
10   *
11   *                                               Written by
12   *                                               Bob Smith (12 Sep '85)
13           aorg    20
14   start   lack    9
15           sacl    7
16   go      lack    40
17           tblw    7
18           b       go
19           end
20   <
```

File : lbltest.t

```
1    >
2    *       To test the duality of the 'equ' statement with the TMS_320_10
3    *       instructions.
4    *                                               Written by
5    *                                               Bob Smith (16 Sep '85)
6    *
7    * COMPILES OK
8    *                                       define as
9    port0   equ     0               *       a label
10   x0      equ     0               *       data memory location
11   *
12   go      in      x0,port0
13           nop
14   *
15           end
16   <
```

File : put16.t

```
1    >
2    *               Test program 320.2
3    *
4    *       to output in a 16-bit control word to the '320 data bus.
5    *       IT WORKS
6    *                                       Written by
7    *                                       Bob Smith (29 Oct '85)
8    crtl    equ     7
9    port7   equ     7
10   *
11   reset   b       start
12   int     b       start
```

```
*
*
start   lack    1
        sacl    crtl,0
*
        lac     crtl,3          * change bit position determined by
*                                 the shift (bit 8 = 1  here)
        sacl    crtl,0
*
put     out     crtl,port7
forever nop
        nop                     * room for a breakpoint
        b       forever
*
        end
<
```

```
: put8cts.t
   >
   *            Test program 320.2
   *
   *       to output in a control word to the '320 data bus forever.
   *       IT WORKS
   *                                    Written by
   *                                    Bob Smith (28 Oct '85).
   crtl    equ     7
   port7   equ     7
   *
   reset   b       start
   int     b       start
   *
   *
   start   lack    >FF
           sacl    crtl,0
   *
   put     out     crtl,port7
   ever    nop
           nop                  * room for a breakpoint
           b       put
   *
           end
   <
```



```
File : t1.out.t
  1    >
  2    *                              Test Program #1.320
  3    *
  4    *       To output a 16-bit word on the TMS_320_10 data bus forever.
  5    *       Also to observe the timing signals being fed into the interface.
  6    *
  7    *                                                  Written by
  8    *                                                  Bob Smith (11 Sep '85)
  9    *       IT WORKS
 10    *
 11    dummy   equ     127
 12    *
 13    reset   b       start
 14    *
 15    start   lack    255
 16            sacl    dummy
 17    cont    out     dummy,7         * port 7
 18            nop
 19            nop
 20            nop
 21            nop
 22            nop
 23            nop
 24            nop
 25            nop
 26            nop
 27            nop
 28            b       cont
 29    *
 30            end
 31    <
```

```
*** module "init.ex"
                                .data
0000 01                 datbuf: .byte   1
0001 02                         .byte   2
0002 03                         .byte   3
0003 04                         .byte   4
0004 05                         .byte   5
0005 06 06 06 06 06 06 06 06 \
                                .byte   6 : 8           ! initialise 8 bytes to the value of 6
000D 07                         .byte   7

*** symbol table for "init.ex"
 dat 0000 datbuf
```

```
*** module "tst04.s"
                        !       to test the 'fcb 4' on end of ascii string.     15 Oct '85.
     F80C               pdata1  =       0xf80c

                                .org    0x1000

                                .text
1000 CE 10 08           main:   ldx     #halt
1003 BD F8 0C                   jsr     pdata1
1006 3F                         swi

                                .data
1008-1017               halt:   .ascii  "ERROR:  halt:\n\r\04"

*** symbol table for "tst04.s"
 dat 1008 halt      txt 1000 main      abs F80C pdata1
```

APPENDIX 7

15.7 Buses

"Buses, which tie boards to data, control, and power lines, form the backbone of computer systems." [Conolly] This appendix will outline the standard buses used and the new Versa Module Europa bus. The following material has been summarised from Connolly and the VME specification manual.

15.7.1 in General

Firstly, the RS-232C bus. Binary serial data is transferred between data processing equipment. The maximum data transfer rate is 20 kbits per second at a maximum cable length of 20 metres. It has 4 control lines: Request to send (RTS), Clear to send (CTS), Data set ready (DSR) and Data terminsl ready (DTR).

Secondly, the IEEE-488 or the General Purpose Interface Bus (GPIB), is used for interfacing remotely programmable and non-programmble test intruments. It can be implemented for 'talking', 'listening' and 'controlling'. That is, it can support devices that do the talking or listening or controlling. Also this bus can support a maximum of 15 devices at a transfer rate of 1 M bit per second.

The asynchronous transfers are coordinated by 3 handshake lines: the Data Available (DAV), Not ready for data (NRFD) and the Not data accepted (NDAC). In total, there are 24 lines on this bus. Eight of them are for data transfer, allowing byte transfers. Eight for control signals and the rest are for the power and shielding.

15.7.2 VME Overview

In 1981 three companies, Motorola, Signetics and Mostek wrote up the VMEbus specification to support the 68000 processor family. It is an aynchronous system and it uses a backplane for the boards instead of the cable connectors. Essentially, this is a multi_processor system where the bus can be shared between them.

The signal line length is stated to be 19 inches compared to the 20 metres for the RS-232 or the IEEE-488. However, an investigation revealed that there are three groups of bus lines in this VME system: the VME global bus, the VMX private parallel link, and the VMS serial link. Left over, are 64 pins on this backplane that can be used for input and output paths.

The global bus can be reconfigured to have 8, 16 or 32 data lines, and 16, 24 or 32 address lines. Depending on the number of lines needed, this system is easily adaptable. The VMX, being a private link, can access additional memory, handle higher priority traffic or even connect dedicated resources such as terminals or data acquisition systems into the system.

The serial bus allows rapid communication between the various modules plugged into the system. It can be used for broadcasting messages throughout the system. See figure 15.1 for use of this VMS bus. Only three lines, serial data (serdat), serial clock (serclk) and the system reset (sysreset) are provided.

If we define 'n' as being 8, for a byte, and 16 for a word length, it takes n + 2 clock cycles to transmit and n + 1 clock cycles to receive the serial bit stream. If the clock period is 200 ns, then from the timing diagram, on page B-26 of the TMS_320_20 Preliminary User's Guide, 8 bits will provide a rate of 500 kbps and 16 will give 280 kbps.

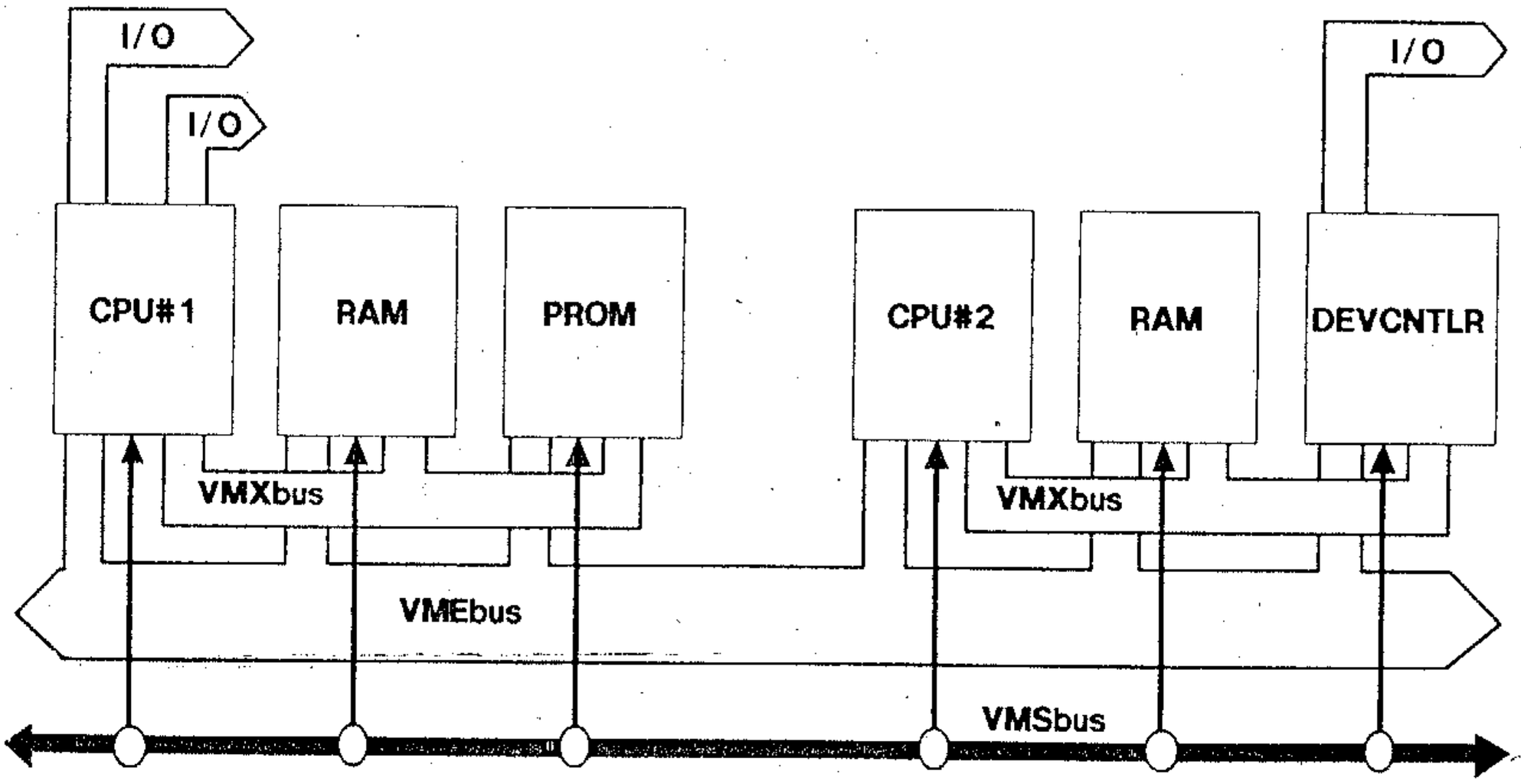


Figure 15.3. Global Serial Link

A figure quoted by the speaker at a Digital Signal Processing Seminar at Texas Instruments in Sydney (23 October 1985), was 2.5 Mbits per second. This implies there is a different frame of reference in calculating the maximum bit transfer rate. Perhaps, he was referring to the minimum serial port clock time of 8 * 0.25 * 200 ns (= 400 ns), giving 2.5 Mbps ?

The next two figures show how this bus can be used for serial and parallel bit stream communication. They are, the 'virtual signal line' and the 'virtual bus'. The diagrams are self evident.

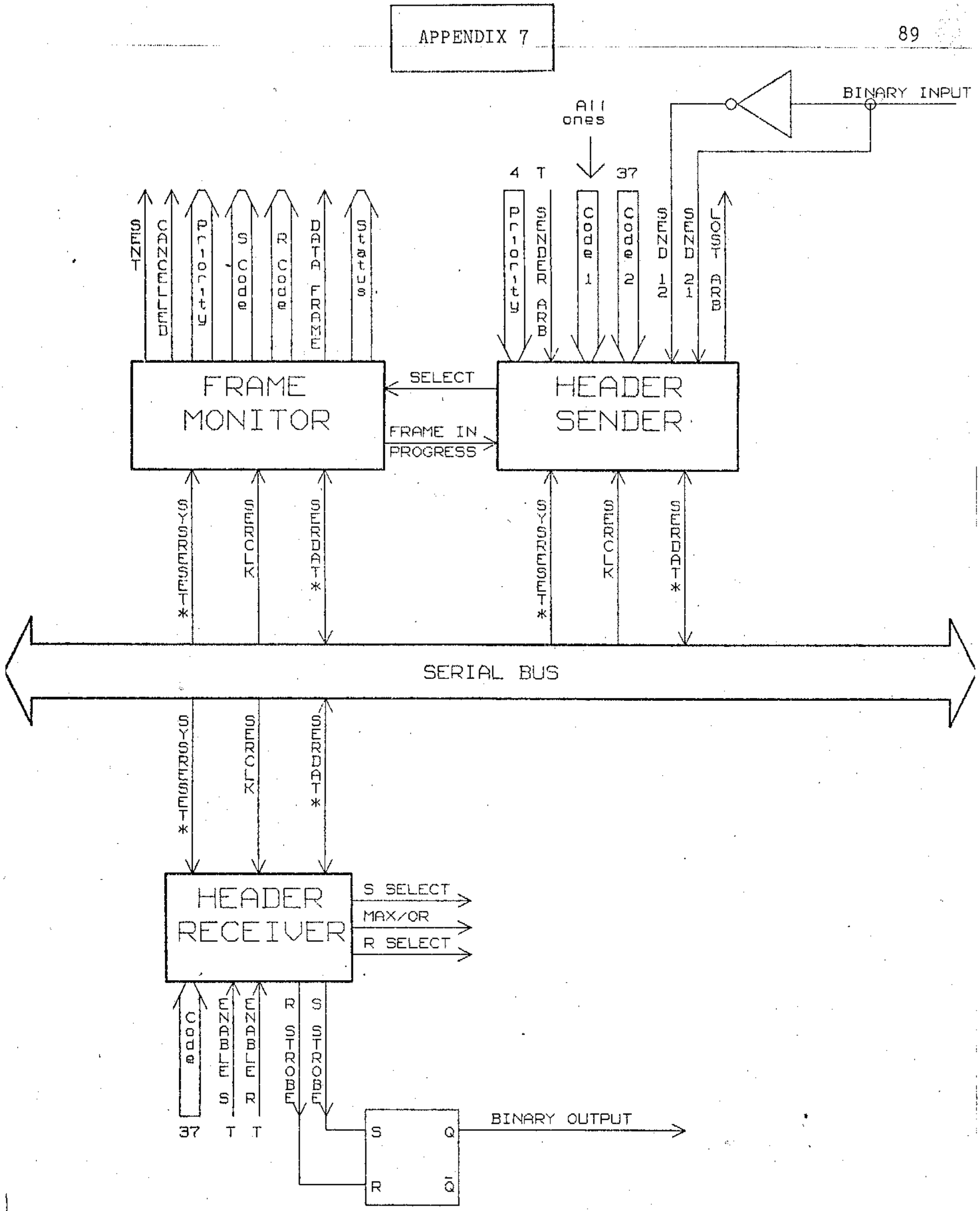


Figure 15.4. Virtual Signal Line example

APPENDIX 7

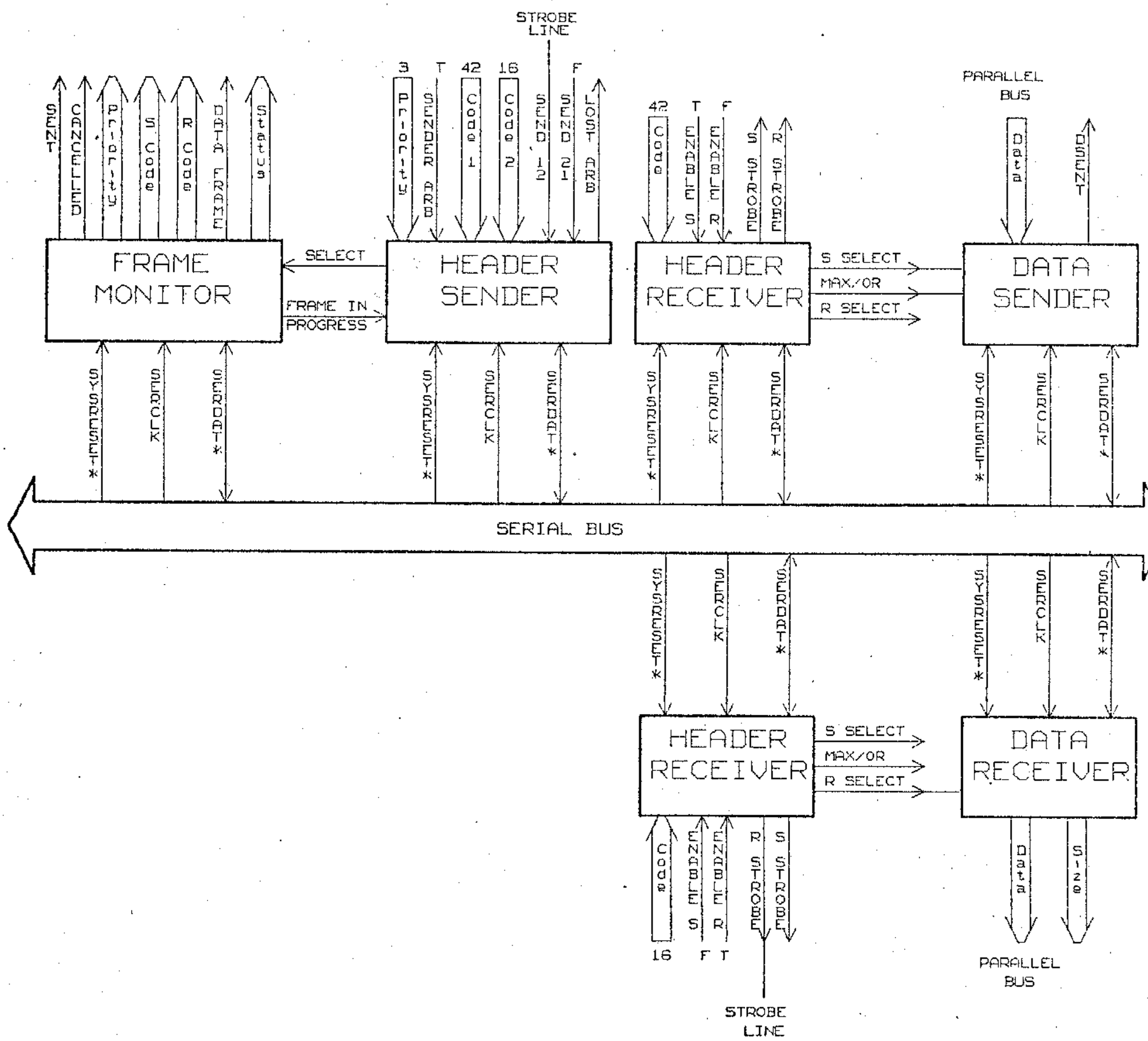
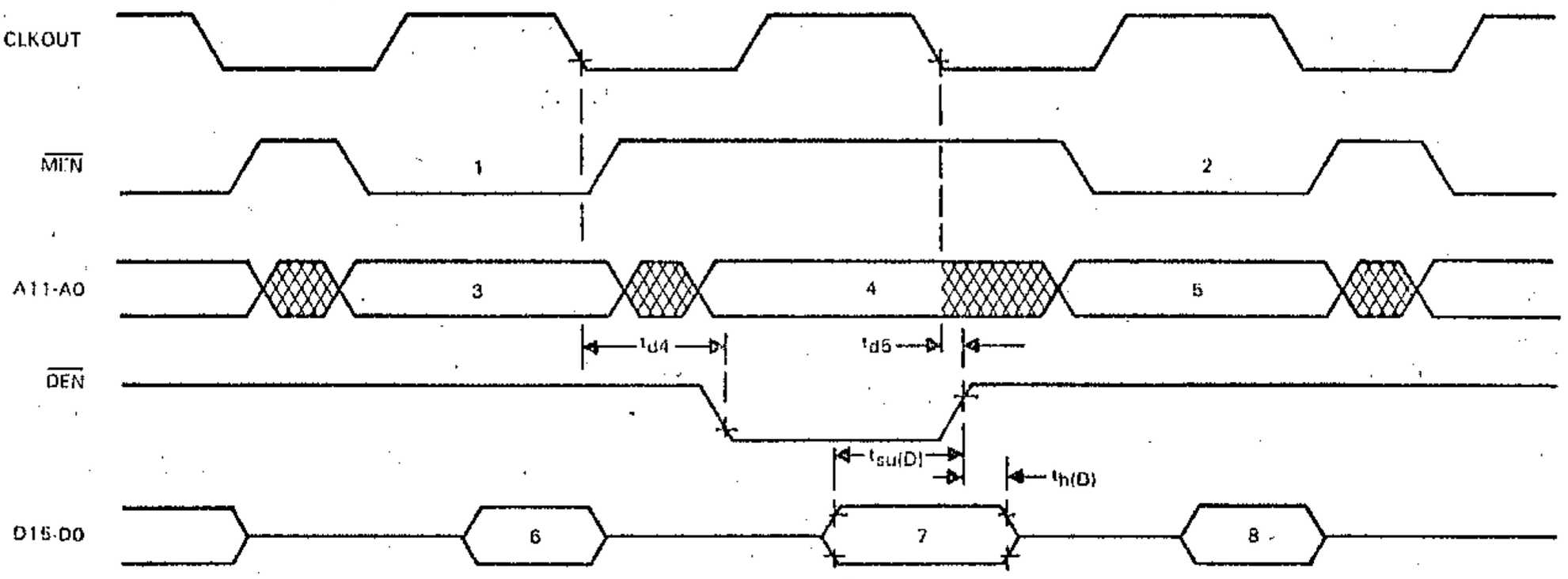


Figure 15.5. Virtual Bus example

APPENDIX 8

15.8 Timing Diagrams for the TMS 320 10

IN instruction timing (USER's GUIDE)

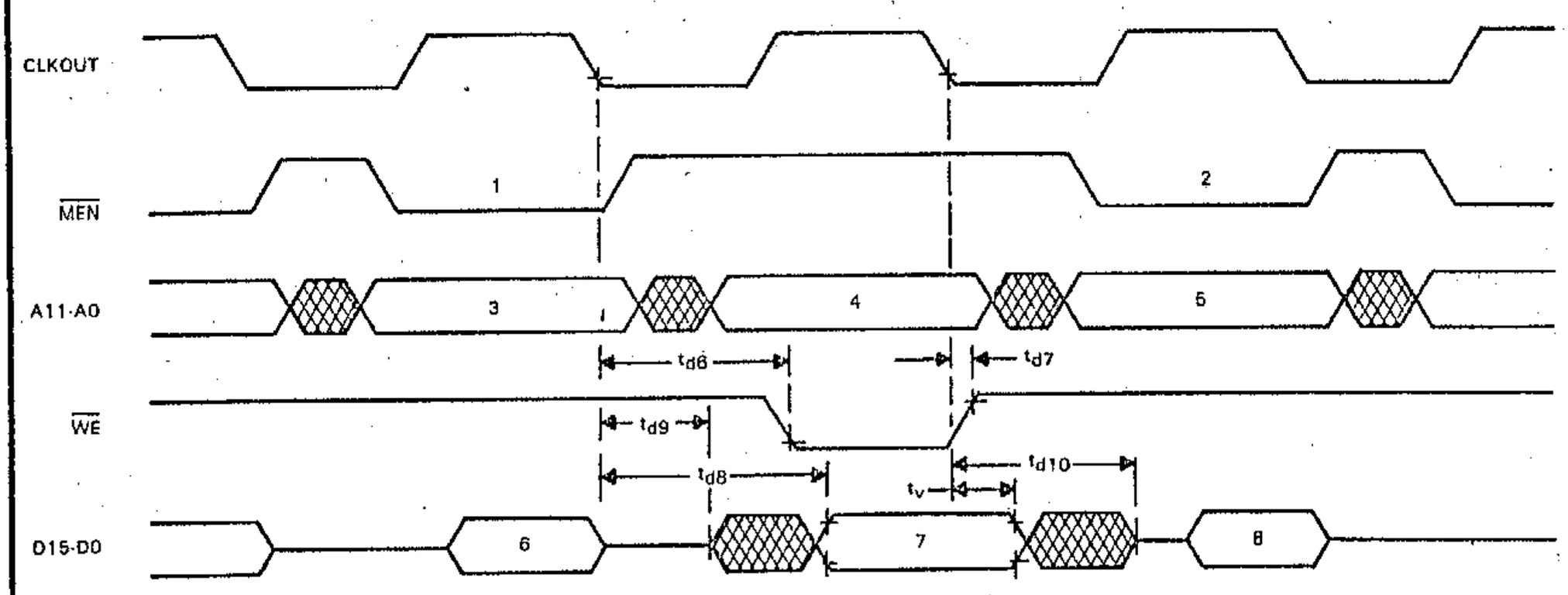


LEGEND:

1. IN INSTRUCTION PREFETCH
2. NEXT INSTRUCTION PREFETCH
3. ADDRESS BUS VALID
4. PERIPHERAL ADDRESS VALID
5. ADDRESS BUS VALID
6. INSTRUCTION IN VALID
7. DATA IN VALID
8. INSTRUCTION IN VALID

NOTE: Timing measurements are referenced to and from a low voltage of 0.8 volts and a high voltage of 2.0 volts, unless otherwise noted.

OUT instruction timing (USER's GUIDE)



LEGEND:

1. OUT INSTRUCTION PREFETCH
2. NEXT INSTRUCTION PREFETCH
3. ADDRESS BUS VALID
4. PERIPHERAL ADDRESS VALID
5. ADDRESS BUS VALID
6. INSTRUCTION IN VALID
7. DATA OUT VALID
8. INSTRUCTION IN VALID

Figure 15.6 IN/OUT INSTRUCTION TIMING

NOTE: Timing measurements are referenced to and from a low voltage of 0.8 volts and a high voltage of 2.0 volts, unless otherwise noted.

APPENDIX 9

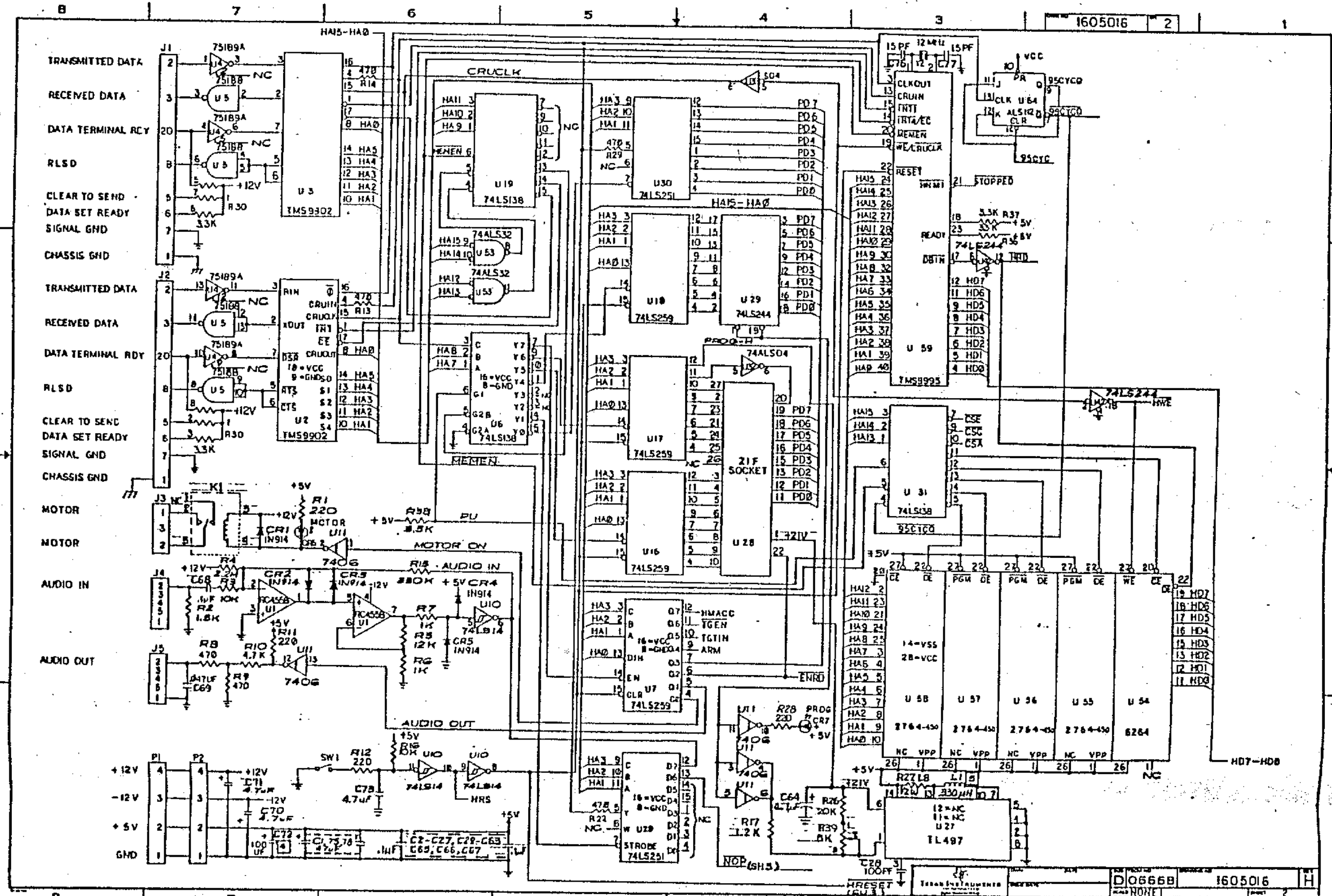
15.9 Worked example for Slave Program

The following is a worked example to indicate how the coefficients of the test program, in the slave processor, were calculated. Due to the volume of this work, and the thesis submission deadline, will be added to an appendix at a later date.

APPENDIX 10

15.10 Circuit Diagram

15.10.1 of the EVM board



1605016
2
TRANSMITTED DATA
RECEIVED DATA
DATA TERMINAL RDY
RLSD
CLEAR TO SEND
DATA SET READY
SIGNAL GND
CHASSIS GND
J1
J2
J3
J4
J5
75189A
75188
U5
U4
R30
3.3K
+12V
TMS9902
U3
U2
HA15-HA0
CRUCLK
HMACC
TGEN
TGTIN
ARM
MEMEN
U19
74LS138
74ALS32
U53
U6
U30
74LS251
U18
74LS259
U29
74LS244
PROG-H
74ALS04
U17
U16
21F SOCKET
U28
U7
U29
STROBE
NOP (SH3)
HRESET
CLKOUT
CRUIN
INT1
INT4/EC
MEMEN
WE/CRUCLK
RESET
NMI
STOPPED
READY
DBIN
U59
TMS9995
U64
ALS112
95CYCO
95CYC
U31
74LS138
CSE
CSC
CSA
WE
15 PF
12 MHz
14=VSS
28=VCC
U58
U57
U56
U55
U54
2764-450
6264
HD7-HD0
TL497
U27
330 µH
R28
PROG CR7
U11
7406
MOTOR
K1
CR1
1N914
R1
220
R58
3.5K
PU
MOTOR ON
AUDIO IN
AUDIO OUT
RC4558
U1
CR2
CR3
CR4
CR5
R15
330K
U10
74LS14
SW1
R12
220
HRS
+12V
-12V
+5V
GND
P1
P2
C71
C70
4.7uF
C2-C27, C29-C63
C65, C66, C67
ENRD
TEXAS INSTRUMENTS
D
06668
1605016
H
NONE

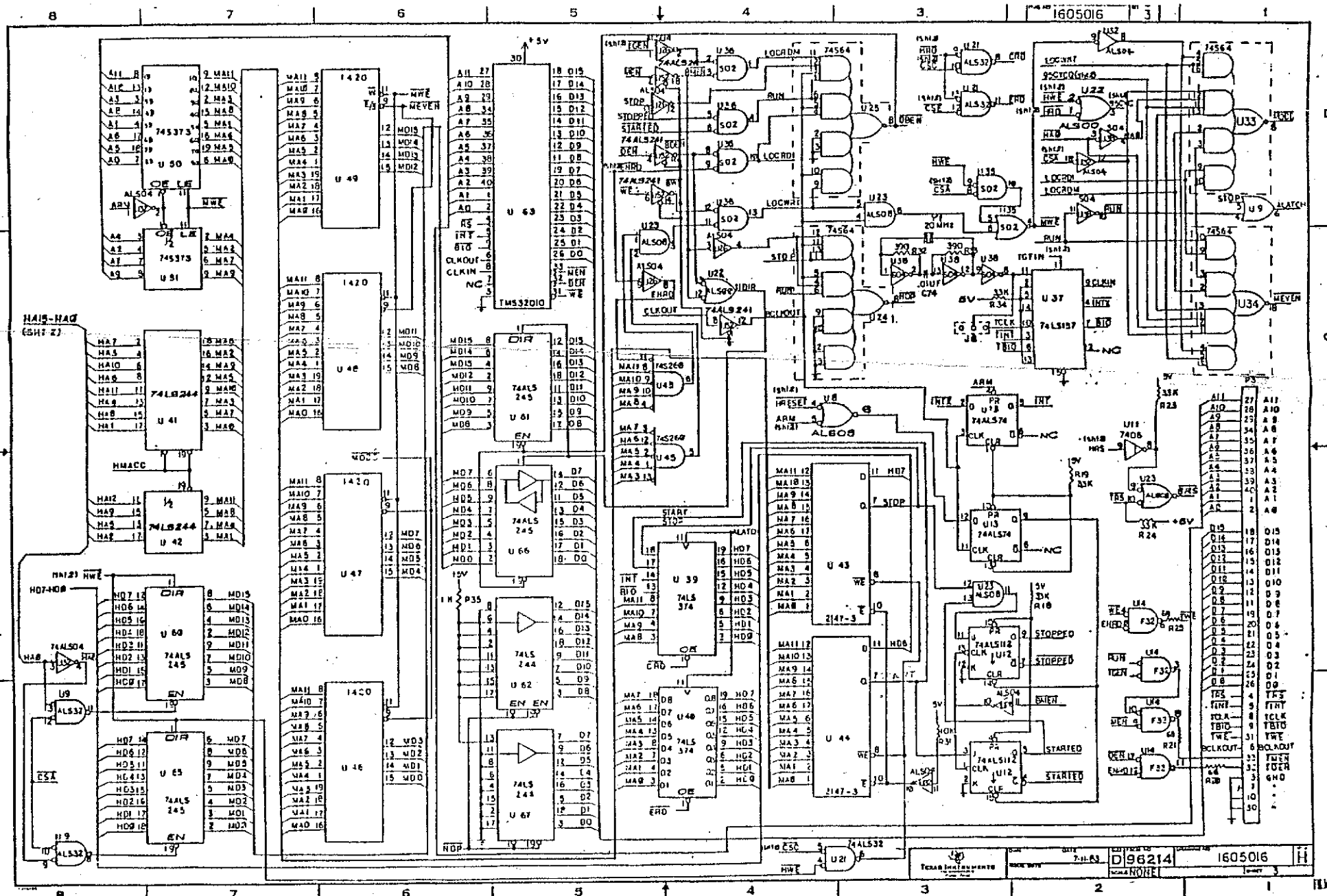


Fig 15.7 EVM CIRCUIT DIAGRAM

A-7

93a

15.10.2 of the Interface

CIRCUIT DIAGRAM OF INTERFACE

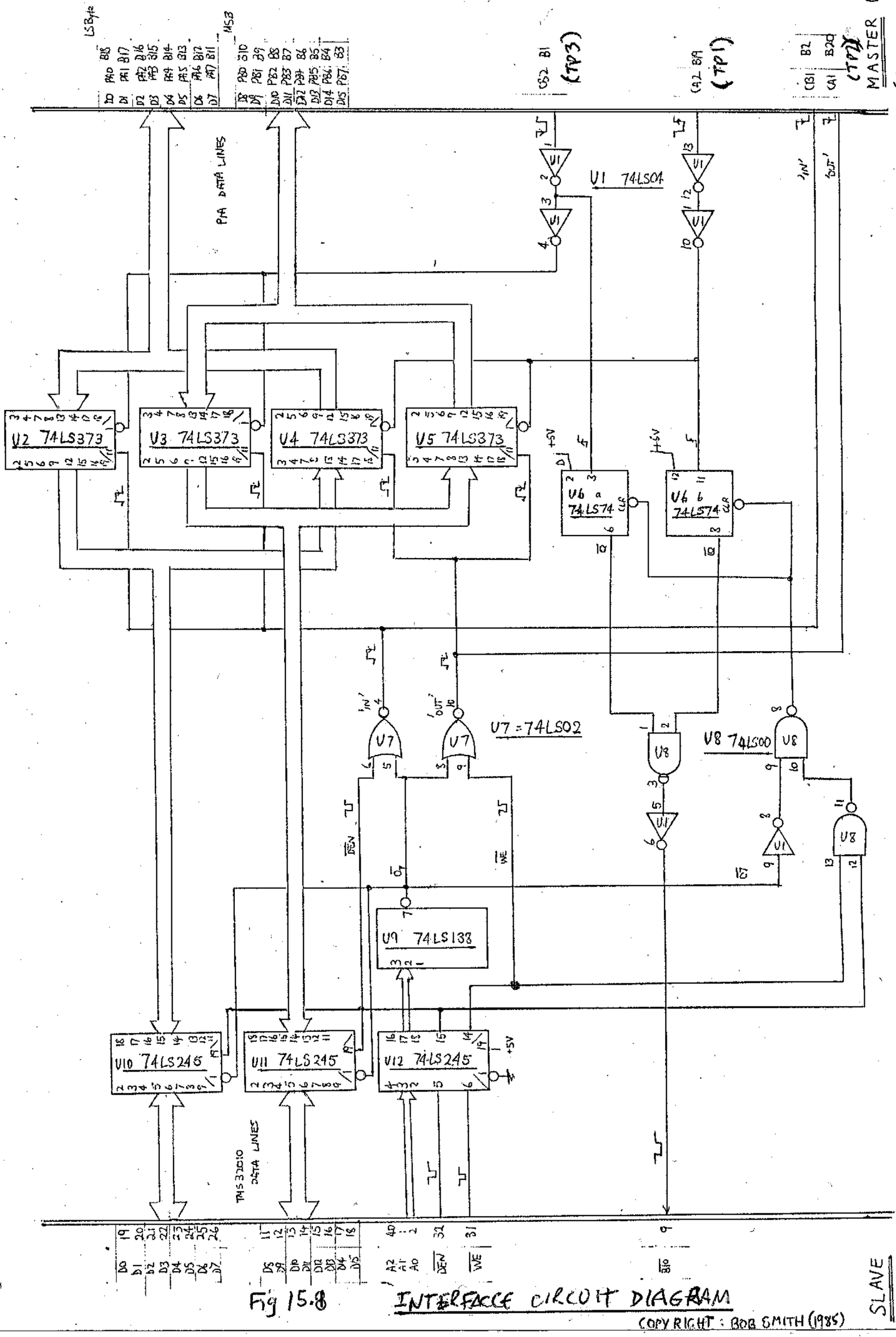


Fig 15.8 INTERFACE CIRCUIT DIAGRAM

COPYRIGHT : BOB SMITH (1985)

TABLE OF FIGURES

pg 95