

https://dspace.org/

Development of a Mil-std-1553b Time Division Data Bus Terminal

Holt, Geoffrey Anthony

2015-01-27
http://hdl.handle.net/10539/16800

Downloaded from a DSpace repository. https://dspace.org/

DEVELOPMENT OF A MIL-STD-1553B TIME DIVISION DATA BUS TERMINAL

Geoffrey Anthony Holt

A project report submitted to the Faculty of Engineering, University of the Witwatersrand, Johannesburg, in partial fulfilment of the requirements for the degree of Master of Science in Engineering.

Johannesburg, 1985

DECLARATION

I declare that this project report is my own, unaided work. It is being submitted for the Degree of Master of Science in Engineering in the University of the Witwatersrand, Johannesburg. It has not been submitted before for any degree or examination in any other University.

C Hall

27 day of DECEMBER 1985

ABSTRACT

A flexible, general purpose MIL-STD-1553B network terminal is developed to enable evaluation and possible later use of this standard in distributed processing applications. The terminal is capable of operating in any of the defined modes and supports all optional features of the standard.

A survey of components available for interfacing to MIL-STD-1553B is presented. This leads to a choice of the Marconi range of components as being best suited for use in this terminal.

A specification for the terminal is formulated. The terminal hardware is designed and a general approach to software for the terminal is discussed. An Intel 80186 microprocessor is used to interface a pair of 1553B busses to Multibus, via a dual port memory which provides message buffering.

A prototype version of the terminal has been constructed, and lessons learned in the testing and debugging of this hardware are discussed.

ACKNOWLEDGEMENTS

The author wishes to acknowledge the following assistance in the execution of this project.

- Mr. G.T. Gray of the Department of Electrical Engineering, University of the Witwatersrand, for supervision of the project.
- Mr. H. Roos of Teklogic, for advice and constructive criticism.
- Tony Kempe, for his advice, help with Forth, and the use of his software utilities.
- Teklogic (Pty) Limited, for the sponsoring of all the components used in the project.
- The Council for Scientific and Industrial Research for personal financial assistance.

CONTENTS

		Page
DECLARATION		2
ABSTRACT		3
ACKNOWLEDGEMENTS		4
CONTENTS		5
LIST OF FIGURES		9
LIST OF TABLES		10
NOMENCLATURE		11
1	INTRODUCTION	1.1
1.1	Problem Statement	1.2
1.2	Background to Problem	1.3
1.2.1	Network Requirements of Distributed Systems	1.3
1.2.2	MIL-STD-1553B Overview	1.6
1.2.3	Existing 1553B Systems	1.13
1.3	Relevance of Project	1.15
1.4	Scope of the Project	1.16
1.5	Report Layout	1.16
2	SURVEY AND SELECTION OF MIL-STD-1553B BUS INTERFACE COMPONENTS	2.1
2.1	Introduction	2.2
2.2	Selection Criteria	2.2
2.3	Parts of the Bus Interface	2.3
2.4	Survey of Components	2.3
2.5	Validation of 1553B Interface Components	2.6
2.6	Component Selection	2.7

CONTENTS

		Page
3	TERMINAL SPECIFICATION	3.1
3.1	Influencing Factors	3.2
3.2	Functional Specification	3.3
3.2.1	MIL-STD-1553B Interface	3.4
3.2.2	Multibus Interface	3.5
3.2.3	Generalised Input/Output Interface	3.6
3.2.4	Embedding of the Terminal in the Subsystem	3.6
3.2.5	System Configurations	3.6
3.3	Electrical Specification	3.9
3.4	Mechanical Specification	3.9
3.5	Compliance with Specification	3.10
3.5.1	Hardware Design	3.10
3.5.2	Prototype	3.10
3.5.3	Final Version	3.11
4	TERMINAL DESIGN OVERVIEW	4.1
4.1	Hardware Design Overview	4.2
4.1.1	The Local Processor	4.4
4.1.2	Local Memory	4.4
4.1.3	Dual Redundant MIL-STD-1553B Interface	4.5
4.1.4	Generalised I/O Interface	4.6
4.1.5	Multibus Interface	4.7
4.1.6	Dual Port Memory	4.7
4.1.7	Local Peripherals	4.8
4.2	Software Design Overview	4.9
4.2.1	Range of Complexity of Terminal Software	4.10
4.2.2	Proposed Software Structure	4.11
4.2.3	Implementation of Software Scheme	4.16

CONTENTS

		Page
5	PROTOTYPING AND TESTING	5.1
5.1	Prototype Construction	5.2
5.2	Prototype/Design Differences	5.3
5.2.1	Generalised I/O Interface	5.3
5.2.2	Multibus Master Interface	5.4
5.2.3	Bus Interface Busy Latch	5.4
5.2.4	Dual Port RAM	5.4
5.2.5	Interrupt Jumper Matrix	5.5
5.3	Debugging and Testing Methods	5.5
5.3.1	Initial Debugging	5.5
5.3.2	Use of Forth as a Development Tool	5.6
5.4	Design Testing and Results	5.8
5.4.1	Dual Port Memory	5.8
5.4.2	1553B Bus Interface	5.10
6	RECOMMENDATIONS & CONCLUSIONS	6.1
6.1	Design Review	6.2
6.1.1	Possible Design Improvements	6.3
6.2	Future Work on Terminal	6.4
6.3	Conclusions	6.5

APPENDICES

A	MIL-STD-1553B BUS INTERFACE COMPONENTS	A.1
A.1	STC Range	A.1
A.2	Marconi/CTI Range	A.2
A.3	Grumman/SMC Bus Interface Unit	A.4
A.4	Harris Range	A.5
A.5	ILC Data Device Corporation Hybrid Set	A.6
A.6	Rockwell-Collins 1553 Interface Device	A.8

CONTENTS

		Page
B	MRTU 53045-SUBSYSTEM INTERFACE AND OPERATION	B.1
B.1	Remote Terminal Operation	B.1
B.2	Bus Controller Operation	B.9
B.3	Bus Monitor Operation	B.11
C	HARDWARE DESIGN	C.1
C.1	Local Processor	C.1
C.2	Local Memory	C.6
C.3	MIL-STD-1553B Bus Interface	C.7
C.4	Generalised I/O Interface	C.26
C.5	Multibus Interface	C.28
C.6	Dual Port Memory	C.31
C.7	Local Peripherals	C.36
C.8	Processor Address Space Allocation	C.39
D	HARDWARE SCHEMATICS AND CONFIGURATION TABLES	D.1
D.1	Pin to Pin Schematics	D.1
D.2	Component List	D.17
D.3	Prototype Board Layout	D.18
D.4	Jumper Configuration	D.21
E	PAL DESIGN AND PROGRAMMING	E.1
E.1	Design Procedure	E.1
E.2	Bus Interface PALS	E.3
E.3	Dual Port Arbitration and Control PALS	E.17
F	*TEST SOFTWARE LISTINGS*	F.1
F.1	Assembler Hybrid Test	F.1
F.2	Forth Primitives	F.9
F.3	Dual Port Memory Lock Test	F.14
F.4	Forth Bus Interface Drivers	F.16
F.5	Command Interrupt Service Routine	F.19

REFERENCES

LIST OF FIGURES

		Page
3.1(A)	Line Replaceable Unit	3.7
3.1(B)	Partially On Board Subsystem	3.7
3.1(C)	Fully On Board Subsystem	3.8
3.1(D)	Stand Alone	3.8
4.1	Hardware Block Diagram	4.3
4.2(A)	Dual Port Memory Map	4.13
4.2(B)	Buffer Structure	4.13
C.1	The Local 186 Processor	C.3
C.2(A)	Template for Bus Interface Schematic	C.24
C.2(B)	MIL-STD-1553B Bus Interface	C.25
C.3	SBX Address Maps	C.27
C.4	Multibus Master Interface	C.30
C.5	Dual Port Memory	C.32
C.6	Interrupt Jumper Matrix	C.38
D1-14	Pin-to-Pin Schematics	D.2
D15	Prototype Board Layout	D.19
E.1	Handshaking and DMA Request Lines	E.10
E.2	RTSYNC* Line	E.11
E.3	ENREQBUS Lines	E.12
E.4	RX/INCMDINT Line	E.13
E.5	DBC/RTOINT Line	E.14
E.6	PAL 1 Version 0	E.15
E.7	PAL 2 Version 2	E.16
E.8	DMCS* Line	E.18
E.9	ARBIP Line	E.20
E.10	Control Logic	E.22
E.11	PAL 3 Version 1	E.25
E.12	PAL 4 Version 1	E.26

LIST OF TABLES

		Page
2.1	Comparison of Components	2.4
3.1	Built In Test Word Bit Assignments	3.5
C.1	Programmable Chip Selects and Wait States	C.40
D.1	Component List	D.17
D.2	ROM Jumpers	D.21
D.3	RAM Jumpers	D.21
D.4	SBX Jumpers	D.22
D.5	Multibus Slave Decoding Jumper	D.22
E.1	LBEN* AND HBEN	E.5
E.2	LBSTJ* AND HBSTB*	E.7
E.3	Arbitration Truth Table	E.19

NOMENCLATURE

The following conventions have been adopted for the name of signal lines:

All signal lines are written in upper case (eg. CLKOUT). If the signal is active low, the name is followed by an asterisk (eg. IUSTB).

Abreviations used in the text have the following meanings:

1553B	MIL-STD-1553B
BC	Bus Controller
BIT	Built-In-Test (word)
BM	Bus Monitor
CMOS	Complimentary Metal Oxide Semiconductor (logic)
CPU	Central Processing Unit
DMA	Direct Memory Access
EPROM	Erasible Programmable Read Only Memory
FIFO	First In First Out
IEEE	Institute of Electrical and Electronic Engineers (standardising body)
I/O	Input/Output
LED	Light Emitting Diode
LRU	Line Replacable Unit
LSTTL	Low power Schottky Transistor-Transistor Logic
MSI	Medium Scale Integration
PAL	Programmable Array Logic
PIC	Programmable Interrupt Logic
PIT	Programmable Interval Timer
RAM	Random Access Memory
ROM	Read Only Memory
RT	Remoter Terminal
RX	Receive(r)

NOMENCLATURE

SBX	iSBX I/O bus (IEEE P959 I/O bus)
SSI	Small Scale Integration
TTL	Transistor Transistor Logic
TX	Transmit(ter)
USART	Universal Synchronous Asynchronous Receiver Transmitter

CHAPTER 1

INTRODUCTION

The problem that is addressed by this project is stated. This is followed by the relevant background material to the problem leading to a justification of the project. The scope of the project work is defined and finally there is a guide to the layout of this report.

1.1 PROBLEM STATEMENT

Teklogic has a need for a fast, reliable and robust network to be used in distributed processing, and remote sensing and actuation applications.

These applications may include avionics, "fly-by-wire", and armament functions in aircraft, and similar operations in other vehicles such as tanks, armoured cars and small naval vessels.

Such a network could also find application in the industrial field of process control where again reliablilty and robustness (but not so much speed) are of prime importance.

A potentially suitable network is defined by MIL-STD-1553B. The function of this project is to provide a flexible, general purpose MIL-STD-1553B data bus terminal for the evaluation and possible later use of this network in distributed processing applications.

The terminal is required to make use of one of the standard components available for interfacing to the bus, rather than attempting a "from scratch" design, while still maintaining flexibility and generality. It must interface to Multibus and if possible not occupy more than one board. A further requirement is that Intel microprocessor products be used where necessary, as support for these is available.

1.2 BACKGROUND TO PROBLEM

1.2.1 Network requirements of Distributed Systems

The required characteristics of the communication channel used by a distributed computer system are influenced by the application of the system.

Typical applications in this case are aircraft systems such as:

- Instrumentation
- Navigation
- Flight safety
- Remote control or "fly-by-wire"
- Weapons control
- Flight recording

These systems may also be used in other types of vehicles, both military and civil.

A second class of application to be considered is industrial process control.

1.2.1.1 Speed

The speed requirements vary from high speed in data processing (because of the large amounts of data typically involved) to relatively low speeds in the case of process control if the plant time constants involved are long. The speed in any system also obviously depends on the number of units that are connected to the network.

The aircraft or vehicle type application lies between the two. A relatively high speed is required as the time

constants are fairly short (in the case of remote control) and a system may need to respond rapidly to an event (in the case of flight safety and weapons control). On the other hand there are usually few units in the system.

1.2.1.2 Extent

For use in a vehicle, the network need only extend a few tens of metres and support a few units. For use in process control on the other hand, in a large system the extent may be a few kilometres and the number of units a few hundred.

1.2.1.3 Integrity

Both classes of application demand secure [illegible]ation. This is particularly true when messages (for example triggered by events) may occur only once and must not be lost.

This may of course be ensured at a level above the communication medium at the price of incurring an overhead. It is, however, more desirable that the network have a low error rate, and some method of error detection built into it.

Considering that the electrical environment is potentially noisy, the network should have good noise immunity in order to have a low error rate.

1.2.1.4 Robustness

In both classes, the environment in which the network has

to operate is potentially harsh from an electrical, mechanical and temperature viewpoints. Further, in a military application there is the possibility of ballistic damage.

The network hardware must thus be hardened for use in such an environment to ensure a reasonable robustness. The effect of ballistic damage can be reduced by the inclusion of redundancy at the network level, specifically, redundancy of the actual communication medium (cabling).

1.2.1.5 Fault Tolerance

In all distributed applications, it is a general requirement that a failure of one unit connected to the network does not cause the failure of the entire network.

Such a failure must thus neither render the communication medium unuseable nor leave the system without overall control.

1.2.1.6 Protocol

The type of protocol required arises partly from the above points.The protocol must support any error detection and redundancy that is available. It must also have a suitable mechanism for fault recovery.

Further, the network should be deterministic, that is; the maximum delay in the delivery of a message to its destination must be calculable. This is again important when critical messages are to be passed.

1.2.2 MIL-STD-1553B Overview

The U.S. Department of Defence sets standards to be used and applied by the military services and their contractors. A military standard is a document that establishes engineering and technical requirements for procedures, practices, and methods that have been adopted as standard. One of these is MIL-STD-1553B "Aircraft Internal Time Division Command/Response Multiplex Data Bus" [3].

1.2.2.1 Origins

The steadily increasing complexity of military systems, particularly in avionics, has made it no longer practical to use independent and self-sufficient units to meet all requirements [5]. Thus integration of such systems has been taking place to share information among the units in the system with the following advantages:

- Elimination of unnecessary duplication of information sensing and display.
- Performance gains.
- Reliability gains.
- Cost reduction.
- Space saving.

This integration was however initially carried out with little thought to the interconnection between units; wiring was generally dedicated point-to-point. As the above approach was refined, this wiring method became impractical [4] as:

- The weight and space taken up by the wires was becoming excessive.

- Modifications to the system were difficult.
- Access to all parts of the system for test purposes was difficult.

The solution was to turn to data bus techniques which were also beginning to emerge in commercial applications.

The U.S. military realised the need for data bus techniques and their standardisation as far back as 1968 [5]. At that time other methods failed to meet the high integrity, reliability and flexibility requirements necessary in the military environment. It was therefore necessary to review the characteristics of a data bus. The following factors were considered:

- Modulation and coding techniques.
- Signalling methods and signal detection techniques.
- Transmission media considerations.

Eventually MIL-STD-1553(USAF) was published in August 1973 and found its first full application in the F-16 aircraft. After input from the Army, Navy, and industry, MIL-STD-1553A was released for use by all the armed forces in 1975. As this was applied to more vehicles and systems, certain difficulties were recognised [5]. The standard was revised with more input from suppliers and users, and MIL-STD-1553B was released on 21 September 1978. This final version has now also been adopted in the United Kingdom as DEF.STAN.00-18 Part 2 and by NATO as STANAG 3838.

1.2.2.2 Scope of Standard

The standard establishes requirements for information transfer formats and electrical interface characteristics.

It can be considered as defining a standard for information transfer formats or protocol, while laying down a specification for electrical interface characteristics.

1.2.2.3 General Architecture

MIL-STD-1553B defines a serial, time division multiplexed data bus operating at one Mbit/sec data rate. The communication medium is a shielded twisted pair. Provision is made for the use of more than one such cable for redundancy purposes. The length of the bus is limited to about 100m by the propagation delay of the cable and the response times required by the standard.

A command/response or "speak only when spoken to" type of protocol is used under which all transfers are initiated and controlled by a single bus controller. Up to 31 individually addressable remote terminals can be connected to the bus, each of which can interface with up to 30 subsystems if necessary. Further non-transmitting bus monitors can be connected to the bus to "...receive bus traffic and extract selected information to be used at a later time". These are the three possible modes of terminal operation.

Provision is made to allow terminals to change their mode of operation and for the bus controller to broadcast messages to all remote terminals. Bus controllership can be offered to and accepted by a remote terminal by means of a mode command. The protocol thus has a bus token passing type mechanism built into it.

1.2.2.4 Information Transfer Formats

Also referred to as "message formats" in 1553B, these define the protocol that is used. The exchange of messages is precisely described and there are only ten allowable formats.

They can be divided into two groups:

Data Transfers. These are essentially used to "...extract data from and feed data to a functional subsystem...". They may transfer up to 32 data words at a time.

Mode Commands. These are essentially reserved "...to communicate with the multiplex bus related hardware, and to assist in the management of information flow...". There is provision for 32 mode commands, 15 of these are defined, the rest being reserved. The use of any or all defined mode commands is optional.

1.2.2.5 Modulation and Coding Technique

The technique used is baseband Manchester II biphase encoding at 1 Mbit/second. This method defines bits by a transition from one voltage level during the first half bit time to the opposite level for the second half bit time. The average voltage is thus zero.

1.2.2.6 Word Formats

Each word consists of 16 bits preceded by sync and followed by parity. The sync is three bit-times long, and consists of one and a half bit-times at one level and one and a half bit-times at the other level. The parity (one

bit) is odd. Thus a word is 20 bit-times long, that is 20 microseconds.

MIL-STD-1553B defines three types of words:

Command Word. This has a positive followed by a negative sync. It is transmitted only by a bus controller terminal. It initiates transmission and defines the data transfer format that is to be used and contains the address of the remote terminal that is to respond.

Status Word. This word also has a positive followed by a negative sync. It is transmitted by remote terminals at the beginning of their response to a command. The status word identifies the remote terminal and passes certain status information back to the bus controller.

Data Word. This word has a negative followed by a positive sync. It is always transmitted contiguously with a command word, status word, or other data words.

1.2.2.7 Electrical Characteristics

This section of the standard specifies the characteristics of the cable (its impedance, shielding, attenuation and termination), terminal input and output characteristics (waveform rise times, noise, common mode rejection, and impedance) and cable stubs.

Terminals may be connected anywhere on the bus in one of two ways: Direct coupled stub connections which should not be longer than one foot (300 mm), and Transformer coupled stubs which may be up to 20 feet (6m)

1.2.2.8 Error Rate and Error Detection

The maximum word error rate that a terminal may exhibit under specified test conditions is one part in 10**7. Theoretical and experimental results indicate that an undetected bit error rate of 10**-12 can be expected from a practical system using the built-in mechanisms (parity, Manchester, and sync validation).

1.2.2.9 Options in the Standard

There are many parts of the standard which are optional. These are summarised below:

Mode of terminal operation. Terminals may or may not support more than one of the three modes of operation.

Subaddresses. Remote terminals may have provision for any number of subaddresses, between one and thirty, within their subsystems.

Embedding of subsystems. A remote terminal may be embedded in, or separate from but communicating with, the subsystem(s) that it serves.

Broadcast. Terminals may or may not support the use of broadcast commands.

Mode commands. These are all optional as to whether or not they are supported by a terminal. Non-implemented mode commands should simply be ignored.

Built-in-test word. One of the mode commands is for a remote terminal to transmit a built-in-test word. This

word is intended to supplement the available bits in the status word when a terminal is sufficiently complex to warrant its use. There is no specification defining what the bits will be used for, except that they shall not be used to transfer subsystem information, and their definition is left up to the designer.

Stub coupling. The terminal may use a transformer coupled stub or direct coupled stub or have provision for both.

Redundant busses. The terminal may support extra bus cables to be used as redundant standbys.

1.2.2.10 MIL-STD-1553B and the OSI/RM

MIL-STD-1553B has not been related to the OSI reference model in any of the references used in this project. This is probably due to the fact that 1553B was defined before the work on the reference model was completed. This is a brief attempt to show what parts of the model are defined by 1553B.

Level 1, the Physical Control Level comprises the physical, mechanical, functional and procedural characteristics required to connect, maintain, and disconnect the physical circuits between equipments [9]. This layer is defined by the MIL-STD-1553B standard with the exception of the mechanical characteristics as no specific connector configuration is laid down.

Level 2, the Link Control Level is concerned with reliable interchange of data connected by level 1 facilities and includes address differentiation, message identification, error detection and response, and flow control. All these features are built into the standard, but since some of

them are partially optional (built-in-test word and vector word), they will vary from terminal to terminal.

Levels 3 upward are concerned with more complex networking operations and support transmission across more than one network. MIL-STD-1553B does not extend to any of these layers.

1.2.2.11 Signal Suitability

The appendix to the MIL-STD-1553B document, which gives some design guidelines suggests, that signals of bandwidth 400 Hz or less are the most suitable for inclusion on the bus, while signals in the range 400 Hz to 3 kHz may be accommodated if the loading on the bus is low enough. High bandwidth signals (eg. video) must obviously be excluded [3].

Signals that have a low rate but possess a high urgency *(event type messages) are also considered to be generally* unsuitable because of bus latency.

1.2.3 Existing 1553B Systems

The existing applications of the standard are mainly in avionics. This is mainly because all new U.S. military aircraft requiring a data bus are obliged to use 1553B. The specific application examples found in the references are all aircraft systems, but mention is made of ground and naval systems that are operating, as well as one on the space shuttle [5]. Although the suitability of the bus to industrial applications is often mentioned [11,17,4], there is no reference to specific cases of such systems.

From the examples it is clear that in the past at least, a 1553B terminal was seldom designed in isolation. Generally an entire multiplex system was designed and then the terminals were designed to meet specific requirements in the system. This would often resulted in several different types of terminals in the system, each possessing differing capabilities.

Finally there are some variations of the standard that have been suggested and in some cases are in prototype form:

Transmission method. Fibre optic [5] and current mode transmission [17]. Although non-standard, these could be useful in certain applications and could form the basis of new standards with only the relevant sections altered.

Bus length. This could be extended by increasing the no response time out [11]. This does not violate the standard, as only a minimum no response time out is specified, but the minimum is invariably used in aircraft applications.

Bus speed. There is a design supporting a 2.5 MHz bit rate [17]. This is obviously non-standard.

Use of reserved parts of standard. The reserved mode codes and reserved bits in the status word could be used to convey application specific subsystem information, thus increasing efficiency [15]. Again this is in obvious violation of the standard.

1.3 RELEVANCE OF PROJECT

From the background study, it is clear that MIL-STD-1553B is potentially well suited for the type of distributed processing applications outlined in section 1.2.1. In order to evaluate, and possibly later use the standard in such applications, a 1553B terminal design is required.

In contrast to the usual approach, in which the system is designed around 1553B and the function of each terminal is fixed, the specific requirements of this terminal will change from one application to the next, or from one function within an application to the next.

In the case that the standard is put into use in a system, it will save considerable design time to have a configurable terminal that can be used in every connection to the bus. It will also make the system far more maintainable, particularly if the terminal can be made to be an easily replaceable module (hence the requirement that it occupy only one board).

The terminal must thus be as flexible and general as possible. All optional portions of the standard should be implemented and where there is a choice, both possibilities should be accommodated. If the terminal could support some of the non-standard variations this might be an advantage, but such variations would have to be used with great discretion. If the terminal could use the status bits and mode codes that are currently reserved it would be able to support possible future versions of the standard which might define functions for the these status bits and mode codes.

The requirements that the terminal use Intel products and interface to Multibus are limiting but practical in terms

of the support which is available, and equipment which is in general use by Teklogic.

Finally this project is intended to add to the local knowledge and expertise in MIL-STD-1553B.

1.4 SCOPE OF THE PROJECT

The scope of this project is the specification, hardware design and verification of a suitable MIL-STD-1553B data bus terminal to meet the stated requirements of the problem.

This also involves a survey of commercially available 1553B bus interface components (since a "from scratch" bus interface design has been ruled out), formulation of an overall approach to the design of the terminal (including a software scheme), and the construction of a prototype terminal with which to verify the design.

1.5 REPORT LAYOUT

The remainder of this report is arranged as follows:

The available bus interface components are surveyed in chapter 2. With this information it is possible to draw up the functional specification in chapter 3. The overall design of the terminal is presented down to block diagram level in chapter 4. Chapter 5 deals with the prototype hardware construction and testing. Finally chapter 6 contains conclusions and suggestions for further work.

Appendix A is survey of 1553B bus interface components. The interface to, and operation of, the components that were selected is summarised in appendix B. Appendix C

describes the detailed implementation of the hardware blocks defined in chapter 4. Appendix D contains the full circuit schematic diagrams, component list, configuration information and board layout of the prototype. The method, and details, of the design of the PAL components used is in appendix E, and listings of software used to test the prototype are in appendix F.

CHAPTER 2

SURVEY AND SELECTION OF MIL-STD-1553B BUS INTERFACE COMPONENTS

The criteria for the choice of bus interface components are discussed. The components that are available are then surveyed. A selection of bus interface components is then made and justified.

2.1 INTRODUCTION

Before a realistic functional specification for the MIL-STD-1553B terminal can be defined, it is necessary to review the capabilities of the various components that are available to interface to 1553B. The level to which the standard is implemented, the options that are supported, and the restrictions that are imposed by the components that are finally selected, will affect what the terminal is functionally capable of doing. The selection is thus made before the functional specification is presented.

2.2 SELECTION CRITERIA

The criteria on which a selection is made are as follows:

Minimisation of external hardware and software required to complete the full protocol. The more that is done in this regard by the bus interface components themselves, the less the burden on hardware external to the components (thus saving board space) and software (thus freeing processor time for other functions).

Simplicity of interfacing to the components. For example: Do the components have an 8 or 16 bit bus? Are commands specified by a pattern of input lines or simply by writing to a control block in memory? Is there a DMA type interface or must transfers be handled externally? Again the more simple this interface, the less board space will be occupied.

Ability to implement optional features of 1553B. The components must allow the support (either directly or with external hardware and software) of as many of the options in the standard as possible. Of most importance is that

bus control and bus monitor modes are possible besides the usual remote terminal mode.

Flexibility in terms of reserved parts of 1553B. Will the components allow the use of the reserved mode codes and status bits or not?

Small size. Again, a board space consideration.

The components to be chosen must thus be the ones which best fulfil the requirements of "generalness" and size, while not being so elementary that it will be necessary to design a great deal of hardware and incur a large software overhead to implement the full protocol.

2.3 PARTS OF THE BUS INTERFACE

The bus interface can be separated into three main parts:

- The isolation/coupling transformers.
- The analog driver/receiver section.
- The digital section.

It is to the last of these that most of the above criteria apply. Hence the following survey covers mainly the components for the digital section.

2.4 SURVEY OF COMPONENTS

The components surveyed fall roughly into two groups:

Full implementation components. These are usually in the form of a chip set or hybrid. They tend to implement the whole standard, and nothing but the standard.

More general components. These are generally single chips performing some of the low level operations necessary in a 1553B terminal (eg. Manchester encoding/decoding and recognition of sync type, address, broadcast and mode code) but not enforcing any particular response or action by the terminal.

Appendix A is a detailed survey of features of components from the main MIL-STD-1553B manufacturers. Of these the most common appear to be the first four namely Marconi, STC (Smith), Harris, and Grumman/SMC [11,15]. The features of the main components of these manufacturers are summarised in table 2.1

TABLE 2.1 COMPARISON OF COMPONENTS

FEATURE	GRUMMAN	HARRIS	SMITH	MARCONI
Subsystem data I/F	16 bit parallel	serial	16 bit parallel	8 bit parallel
Data transfer control	DMA async handshake	sync	sync	DMA sync
Use of reserved features allowed	yes	yes	no	no
Buffering of data	double	none	one message length FIFO	double

TABLE 2.1 Continued

FEATURE	GRUMMAN	HARRIS	SMITH	MARCONI
Error checks	Manchester parity word count response sync >16 bits	Manchester parity sync	Manchester parity sync bit count word count	Manchester parity sync bit count word count loop test RT address parity no response tx timeout subsystem handshake
provision for other bit rates	no	2,5 Mb/sec	no	no
Bus controller and monitor capability	yes	yes, not directly available	no	yes
Redundancy capability	no	no	yes, with addition of second enc/dec	yes, in hybrid or with second receiver
Packaging	40 pin chip	40 pin chip	3 chip set 48, 40 and 53 pins or hybrid	4 chip set 40, 40, 48 and 40 pins or hybrid
Mode codes handled directly	7 only	none	all	all

TABLE 2.1 Continued

FEATURE	GRUMMAN	HARRIS	SMITH	MARCONI
Remote terminal address	s/w loaded can be read back	must be externally implemented	hardwired with parity bit	hardwired with parity bit
Technology	NMOS	Junction isolated CMOS	CMOS	Iso-CMOS

2.5 VALIDATION OF 1553B INTERFACE COMPONENTS

All MIL-STD-1553B components to be used by the U.S. armed forces have to pass a validation test at the Air Force's Systems Engineering Avionics Facility (SEAFAC) [1]. The tests cover six basic areas:

- Misinterpretations and deviations from the standard.
- Bus configuration.
- Test conditions at manufacturer's facility.
- Basic design flaws.
- Periodic and non-periodic error conditions.
- Terminal response to induced errors.

This gives rise to an argument against the use of software to implement the full 1553B protocol. It is argued that during slight software modifications to suit a particular application, the 1553B protocol may be corrupted, thus requiring re-approval by SEAFAC for every implementation.

This problem, which is not an important consideration in this design, could partly explain the fact that the two British designs from Marconi and Smith Industries have

secured a large portion of the mainly American market [11].

2.6 COMPONENT SELECTION

The interface components chosen are:

- Marconi MRTU 53045 Remote Terminal Bus Control Thick Film Hybrid.
- Two Marconi MCT 3231 Driver/Receivers.
- Two Pascall P1264 coupling transformers for use with MCT 3231s.

The reasons for the choice of the MRTU 53045 are as follows:

- The MRTU 53045 is a very compact version of the Marconi chip set.
- It supports bus controller and bus monitor modes as well as the usual remote terminal mode.
- The full 1553B protocol is implemented and no external hardware or software is required for this purpose.
- The subsystem interface is relatively simple. For bus control in particular it is very easy to use. Almost all internal conditions can be monitored or used to interrupt the subsystem.
- The no response time out can be stretched.
- The device has been used before in South Africa and thus there is some local experience of the device.
- The design has been tried and proven overseas. (This applies to the chip set as such, the hybrid implementation did prove to have some flaws.)

- The documentation is fairly complete, but a little difficult to read due to awkward layout.
- The component was readily available.

The main disadvantanges of this component are:

- Inflexibility from the point of view of reserved features of the standard.
- Built-in-test word is predefined and fixed.
- It is very expensive.

The choice of the MCT 3231 analog section and the P1264 transformer is due purely to the fact that they are directly compatible, both with the MRTU 53045 and with each other.

CHAPTER 3

TERMINAL SPECIFICATION

The factors influencing the specification are reviewed. The detailed specification is then presented. Finally the level to which the design, prototype, and final versions are expected to comply with the specification are discussed.

3.1 INFLUENCING FACTORS

The main factors determining and influencing the specification have been covered in the previous two chapters and are reviewed here.

They are:

- The requirements and
- the constraints as given in the original problem statement.
- The capabilities of the bus interface components that have been chosen.

The requirements are:

- The terminal must implement the MIL-STD-1553B data bus standard.
- It must be general, supporting as many of the optional portions of the standard as possible.
- It must be flexible, allowing its use in a variety of applications.
- It must be able to interface to Multibus so as to enable its use with existing equipment.

The constraints are:

- The terminal must make use of existing components to interface to the bus and not attempt a "from scratch" design.
- The terminal should not occupy more than one Multibus card.
- Microprocessor products to be used are restricted to the Intel range.

The bus interface components selected influenced the final concise specification of the terminal. This does not imply that the components were selected without regard for the specification and then the specification built around the components. Rather the selection was the result of some iterations of terminal specification and reveiw of component capablities. Parts of the final specification however, came about directly as a result of the components ultimately chosen. For this reason the selection of the components is covered before the specification is presented.

The features of the components that affect the specification are:

- The MRTU 53045 supports the full protocol with all mode codes in remote terminal mode.
- The MRTU 53045 may easily be configured for bus control and bus monitor modes.
- The MRTU 53045 does not allow the use of reserved features of the standard.
- The contents of built-in-test word is defined and fixed.
- The MRTU 53045 supports dual redundant busses.
- The Pascall P1264 transformer will support both direct and transformer coupled stubs.

3.2 FUNCTIONAL SPECIFICATION

The terminal may be viewed as a 'black box' which has, as means of communicating with the outside world, a dual redundant MII-STD-1553B bus interface, a Multibus interface, and a generalised I/O interface.

3.2.1 MIL-STD-1553B Interface

The MIL-STD-1553B interface is dual redundant, that is connections are possible to two busses which are regarded as a dual redundant pair as defined in the standard.

The terminal is capable of operating in all three defined modes namely; remote terminal (RT) mode, bus controller (BC) mode and bus monitor (BM) mode. It is capable of changing from one mode to another during operation as allowed by the standard. It may also change modes in special circumstances (such as failure of the bus controller to initiate any communication for longer than a certain time) not normally allowed by MIL-STD-1553B. It is capable of powering up into any one of these modes.

In all three modes the terminal operates in full accordance with the entire standard. In addition all optional features of the standard are implemented. This means that in bus monitor mode, all defined information transfer formats may be intiated and that any command may be sent. In remote terminal mode this means that all mode codes are implemented, and that the terminal may respond to broadcast commands if desired.

When the mode command to transmit built-in-test (BIT) word is received by the terminal in RT mode, the terminal will respond with its status followed by a BIT word where the bits have the significance as given in table 3.1. The full explanation as to the meaning and use of the various bits is given in the MEDL 1553 LSI Chip Set - Remote terminal specification [18].

TABLE 3.1 BUILT IN TEST WORD BIT ASSIGNMENTS

BIT NO.	ASSIGNMENT
0 lsb	Transmitter timeout flag
1	Subsystem handshake failure
2	Loop test failure
3	Mode T/R bit wrong
4	Illegal mode command
5	Word count low
6	Word count high
7	Broadcast transmit data received
8	Bus 0 shutdown
9	Bus 1 shutdown
10	Bus 2 shutdown
11	Bus 3 shutdown
12	Transmitter timeout on bus 0
13	Transmitter timeout on bus 1
14	Transmitter timeout on bus 2
15 msb	Transmitter timeout on bus 3

3.2.2 Multibus interface

The terminal has an interface functionally conforming to the IEEE 796 bus or Multibus specification [24]. The interface has two parts; a slave interface and a master interface.

The slave interface appears as a block in the Multibus memory address space. The size of this block is sufficient to queue messages to and from the network in the event of the maximum of 30 subaddresses being used in a remote terminal subsystem. The memory is dual ported and may be transparently accessed by the terminal internally. Both Multibus and the terminal are able to "lock" a series of transactions to this memory to prevent the other

gaining access. The base address of this memory in the Multibus address space is hardware configurable.

The master interface allows the terminal access to the Multibus in the same way as any other master. The block in the memory address space occupied by the slave interface specified above is not necessarily accessible to the terminal via its master interface.

3.2.3 Generalised Input/Output Interface

This interface is functionally compatible with the IEEE P959 or Intel iSBX I/O bus [24]. It is provided to enable the terminal to communicate with a variety of Input/Output (I/O) devices having this interface, without the need to use Multibus. For prototyping, this interface will be useful to monitor the internal state of the terminal.

3.2.4 Embedding of the terminal in the subsystem

The terminal has the capability of supporting the software for a small subsystem or part of a larger subsystem on-board. Any I/O required by this subsystem could be accessible either via the Multibus interface or the generalised I/O interface. Such a configuration is equivalent to the subsystem with embedded remote terminal as defined in MIL-STD-1553B.

3.2.5 System Configurations

In this context the system refers to the *combination of* terminal and subsystem. The network will have several such systems connected to it.

The system may thus be configured in four main ways (some other combinations are obviously possible) as illustrated in figure 3.1 . The position of the terminal/subsystem functional interface is shown in each case.

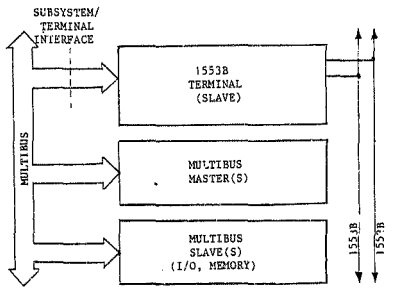


FIGURE 3.1(A) LINE REPLACEABLE UNIT

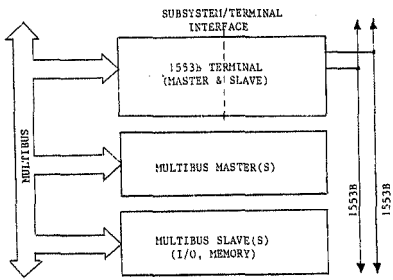


FIGURE 3.1(B) PARTIALLY ON BOARD SUBSYSTEM

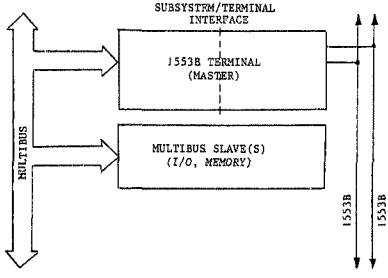


FIGURE 3.1(C) FULLY ON BOARD SUBSYSTEM

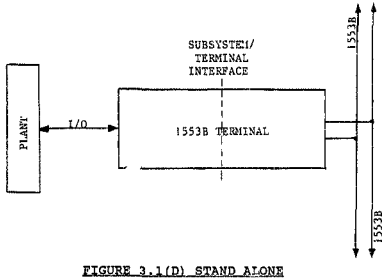


FIGURE 3.1(D) STAND ALONE

In Figure 3.1(A) the terminal can be regarded as a line replaceable unit (LRU) which can be replaced by an "off the shelf" unit in the event of a failure, without regard for the *specific subsystem* it must serve. It is

anticipated that the terminal will be used mainly in this configuration.

The configuration in figure 3.1(B) could be used in medium sized systems where one external processor on Multibus is just insufficient for the application. A small amount of the load could be shifted to the terminal's local processor.

The figure 3.1(C) and (D) configurations will be useful in the case of a small subsystem load. The stand alone mode also has the advantage that it frees the terminal from Multibus.

3.3 ELECTRICAL SPECIFICATION

The Multibus and SBX interfaces conform electrically with the relevent sections of the respective standards.

The *MIL-STD-1553B interfaces conform to the specification* in all respects for both direct and stub coupling.

The components all conform to the military temperature ratings or are directly replaceable with components which do.

The terminal requires power supplies at +5 volts +12 volts and -12 volts.

3.4 MECHANICAL SPECIFICATION

The terminal occupies a single Multibus board.

The Multibus and SBX interfaces once again conform fully

with the mechanical portions of their specifications.

There is no specific portion in MIL-STD-1553B determining how the twisted pair stubs are to be connected, except that the connection must be as close as possible to the coupling transformer. This is adhered to and suitable connectors are used. Separate pins are provided for the direct and transformer coupled stub options on each of the two busses.

3.5 COMPLIANCE WITH SPECIFICATION

3.5.1 Hardware Design

The hardware design of the terminal presented in the following chapters complies fully with the functional and electrical specifications presented above, as far as is possible in the hardware design phase.

Some aspects of the functional specification which relate to software, such as the ability of the onboard processor to be able to support an application as well as control the bus interface components during time critical operations, could not be guaranteed at the time of this design. It was also not possible to predict with certainty that the hardware would fit onto the single Multibus card.

3.5.2 Prototype

The prototype, seeks to verify the the most important aspects of the hardware design and test its abilty to implement the functional specification.

The prototype adheres to the constraint of board space imposed by the single Multibus card. In some cases availability and cost of components were limitations. For these reasons, some parts of the hardware (that in any case are fairly standard designs) were not implemented in the prototype, and certain sections were substituted by other simplified or less general designs.

3.5.3 Final Version

The experience of the prototype shows how much of the original hardware design can be implemented on a single card with all the design flaws eliminated. It also shows what parts of the functional specification cannot be met.

The final product of this design, if it were to be carried through to that stage, would thus best be a result of at least another iteration of functional specification and prototype (using a printed circuit board this time).

CHAPTER 4

TERMINAL DESIGN OVERVIEW

An overview of the design to meet the functional and electrical specifications is presented. It is divided into hardware and software aspects of the design. The hardware is presented as a block diagram design. The functions and requirements of each block are discussed. The proposed software design is then overviewed.

4.1 HARDWARE DESIGN OVERVIEW

The hardware required to meet the design specifications can be broken down into 7 main blocks as shown in figure 4.1:

The three interfaces required by the functional specification:

- Dual redundant MIL-STD-1553B interface.
- Multibus interface.
- Generalised I/O interface.

The medium for message buffering and communication between the terminal and Multibus:

- Dual port memory.

The on board microprocessor system required to control these resources:

- Local processor.
- Local memory.
- Local peripherals.

Figure 4.1 also shows the principal parts internal to each block and the interconnections between each block.

The following paragraphs cover the purpose, requirements and functional design of each block. The actual implementation of each block in hardware is covered in the appendix C while pin-to-pin connection diagrams and configuration information are given in appendix D.

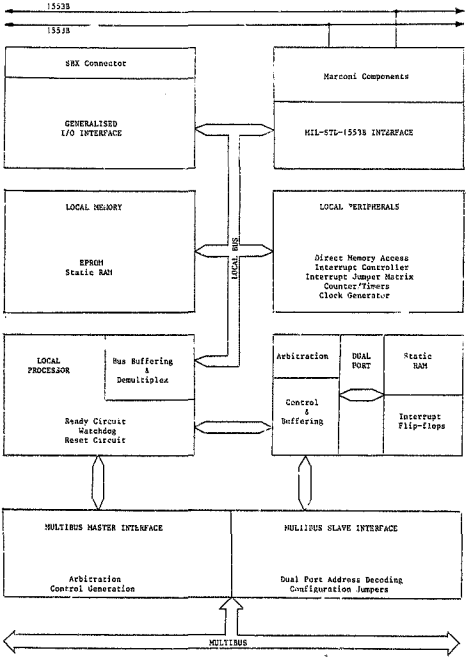


FIGURE 4.1 HARDWARE BLOCK DIAGRAM

4.1.1 The Local Processor

This will be responsible for controlling the MRTU 53045 chip set hybrid, as well as possibly forming part of the subsystem. The response time demanded by the hybrid in certain cases is fairly short. The processor should be able to respond to an interrupt and run the necessary code in time to satisfy this requirement (see section 4.2.3). The processor chosen must therefore be suitably powerful both in terms of speed and instruction set.

Since the data quantities handled by 1553B are intrinsically 16 bit, it makes sense to use at least a 16 bit processor.

A watchdog circuit is provided to restart the processor in the event of it "hanging". This may be either a hardware watchdog, which will only detect that the processor has stopped running, or a software watchdog that, together with suitably written software, will also detect that the processor is hung in a software loop.

4.1.2 Local Memory

The processor will require some ROM for initialisation after power up and to implement the terminal function. If it is required to implement a part or all of the subsystem as well, considerably more program memory may be required (see section 4.2). The ROM memory must thus be expandable up to at least 32k. EPROMS are used at this stage.

Some local RAM is provided. It is anticipated that this will be used simply as scratch pad memory since all the data to be handled by the terminal will be stored in the dual port memory. However in some situations, notably in

prototyping, this memory could be useful as program storage for code which has been downloaded. Since it is necessary for all components that are used to be available in MIL versions, dynamic RAM cannot be used. At least 16k of static RAM is therefore required.

4.1.3 Dual Redundant MIL-STD-1553B Interface

This block represents the major portion of the work in the design, particularly as very few guide lines were available. Some simple examples are given in the Subsystem Interface For MEDL LSI 1553B Remote Terminals, [20 pp 28-30].

This provides the interface as specified to two 1553B busses (allowing both direct and transformer coupling on each) on one side, and clean interface to the local microprocessor system on the other.

It consists of the Marconi bus interface components and the required circuitry to interface these to the local bus, and to provide suitable signals for interrupting the local processor. The interface to, and operation of, the Marconi 53045 chip set hybrid is summarised in Appendix B.

The 8 bit hybrid data bus must be interfaced to the 16 bit processor data bus with a suitable buffering to ease the response times required of the processor. Handshaking must be supplied for these data transfers. A method of writing to and latching the control input lines is required, as well as a means of reading the status output lines. Those output lines which indicate events (rather than states), must either be latched to provide status information, or modified (if necessary) to provide interrupts.

Since the terminal has to be flexible and the exact function of some of the lines on the MRTU 53045 is not clear, the interface is more complex than is strictly necessary. For instance, several output lines have been made suitable for both monitoring as status, and causing interrupts as it is not known at this stage which will be more useful. After the prototype has been fully tested in all modes it may be possible to simplify the supporting circuitry.

The hybrid has, by means of the current word count (CWC) status lines and certain strobe lines, the ability to implement a DMA type interface. While this could be useful for a simple application, it is not a viable method of transfering data in this case for the following reason:

The use of this feature of the hybrid will necessitate either the use of a triple port memory (since both Multibus and the local processor also need access to the data), or the design of handshaking and isolation to enable the hybrid to gain control of the local bus. The first method is obviously impracticable, while the second will be expensive on board area and will tie up the local bus for approximatly two microseconds for each transfer. Transfer of data can be achieved more efficiently by using a normal DMA channel.

4.1.4 Generalised I/O Interface

This interfaces the local bus to the SBX I/O bus. In order to support all possible modules that may be attached, two interrupt lines and one DMA channel are required. It is hardware configurable for different modules.

4.1.5 Multibus Interface

This consists of a master and a slave interface. The master interface allows the local processor access to the Multibus while the slave interface allows Multibus access to the dual port memory.

These could be totally independent of each other, however it is convenient (from a space point of view) to combine certain lines which are common to both. In particular this could be done with the data and address busses. The local busses could be interfaced to the Multibus busses via the dual port busses, th[illegible] duplication of bus isolation and byte swap compo[illegible]

The slave interface also provides hardware selection and decoding of the Multibus dual port address.

The Multibus interrupt lines are available for use either to or from the terminal, the master interface does not support bus vectored interrupts.

4.1.6 Dual Port Memory

This block is composed of three sections; the memory itself, interrupt flip-flops, and the arbitration logic and bus buffering. These are linked by the dual port data, address, and control busses.

The memory has to be large enough to queue messages to and from the network in the event that the full house of 30 subaddresses are used in a remote terminal (see section 4.2). The memory must thus be expandable to 32k. The memory must be static RAM for the same reason as the local memory.

The interrupt flip-flops are provided as a means for the local processor to generate interrupt for another processor on Multibus. These flip-flops are necessary because Multibus does not allow the use of edge triggered interrupts. They are used to cause an interrupting condition which must be reset by the service routine of the interrupted processor. The latches are set and cleared from within the dual port address space.

The arbitration logic manages the bus buffer and RAM control lines to effectively couple either the local processor bus, or Multibus, to the RAM, based on which has asserted a select line. The arbitration logic also allows each bus to lock itself to the memory by means of a lock line. The arbitration logic further provides for the coupling of the local data and/or address busses and the corresponding Multibus busses via the dual port address bus when the local processor is accessing Multibus, if the scheme outlined in 4.1.5 above is implemented.

4.1.7 Local Peripherals

Based on the above discussion, certain peripheral devices are required.

One direct memory access channel is needed to service the 1553B bus interface and another for the SBX interface.

At least one interrupt controller device is required to service the 1553B interface, the SBX interface and Multibus interrupts. Not all of the possible sources of interrupt will be required for a given application. Particularly this is true of the 1553B interface, where many signals have been designated as interrupt sources in

the prototype design, in case they are needed. A second interrupt controller however may be required if the minimum number of lines that may need service in any application exceeds the number provided by one controller.

An interrupt jumper matrix is provided to enable selective strapping of optional interrupts to the controller(s).

An event counter is useful in the case of a terminal which is to be used in bus monitor mode. Conditions occuring on the bus (such as parity errors) can be counted by connecting the parity error interrupt line to the event counter. Thus the input to this device must also go to the above interrupt jumper matrix.

An interval timer is required to provide a regular interrupt (tick) to the processor if a multitasking executive is to be used (see section 4.2).

Sundry clock signals are generated for use by various devices.

4.2 SOFTWARE DESIGN OVERVIEW

The software design presented here is not implemented in this project. The scope of the project covers only test software to verify the operation of the prototype hardware, and to check the ablility of the local processor to respond to the network within the time constraints imposed by the protocol.

This section is provided to show the motivation for some of the features that have been included in the hardware, and to suggest an approach for software for the terminal to be written at a later date. This approach will

accommodate both simple and complex applications in its structure.

The various strategies for program control of the hardware itself (mainly the MRTU 53045 hybrid) are covered in the appendix C under the hardware design as, in some sections, the hardware design is a result of hardware/software interface considerations.

4.2.1 Range of Complexity of Terminal Software

The terminal software could range in complexity from a very simple remote terminal application to a bus controller/remote terminal which supports the full protocol and many subaddresses.

An example of a simple application is a remote terminal that interfaces an instrument on a plant to a 1553B network. If the output of the instrument is a single 16 bit value, then only a single type of message transfer format needs to be accommodated, namely a transmit command to transmit a single word. The bus controller should not send any other type of command to the RT. In this case the transmit/receive bit, subaddress and word count may be ignored by the RT. It does not need to respond to broadcasts or support the subsystem busy bit, the service request bit, the transmit vector word mode command, or the synchronise mode command. All that is required of the terminal software, in this case is, that it continuously reads the output of the instrument, check the MRTU 53045 hybrid for a word request and if true, supplies the value. Note that this is an example of the subsystem residing locally on the terminal board.

An example of a complex application is the case of a

remote terminal which supports the use of the full 30 subaddresses and is capable of accepting bus controllership. All information transfer formats, all status bits, and all mode commands are supported in both RT and BC mode. In RT mode, messages to and from the network must be buffered and queued. In addition part or all of the subsystem software resides locally.

Both extremes of application are accommodated in the following proposed software structure.

4.2.2 Proposed Software Structure

The software is separated into tasks. The bus interface or terminal software is considered as one task, while the subsystem is considered as consisting of at least one task but possibly several tasks (corresponding to different subaddresses). The bus interface task communicates with the subsystem tasks via a suitable data structure in the dual port memory.

The bus interface task always runs on the terminal's local processor, while the subsystem tasks may run locally or on other processors on Multibus. If one or more tasks besides the bus interface task reside locally, some form of multitasking executive must be provided.

4.2.2.1 Dual Port Memory Data Structure

Each of 30 potential subsystem tasks (one for each subaddress) is allocated a pair of buffers in the dual port memory, one for input from and the other for output to the network.

The base of the dual port memory contains a table of pointers (offsets relative to the base of the dual port memory) to the input and output buffers to be used for each subaddress. This allows more than one subaddress to be assigned to the same task by placing the same buffer addresses in the pointers for each subaddress.

Figure 4.2(a) is a diagram of the dual port memory using this scheme.

The buffers are managed as queues of optional size and also provide for the passing of control information. Figure 4.2(b) is a diagram of a possible buffer structure. The max queue size indicates the size of the queue area. The following three locations manage the queue. The next 12 locations could be used for passing control information (such as subsystem busy, subsystem status, service request synchronising word and vector word) or semaphores indicating the validity of data.

A dual port memory of 32k will allow an application to use a full house of subaddresses each having separate input and output buffers capable of holding seven full length (32 word) messages as follows:

Buffer pointers: 2 x 30 = 60 words
Each buffer: 32 + 7 x 32 = 256 words
Total size of data structure: 60 + 30 x 2 x 256
= 15420 words
= 30840 bytes
Dual port memory of 32k = 2**15 = 32768 bytes

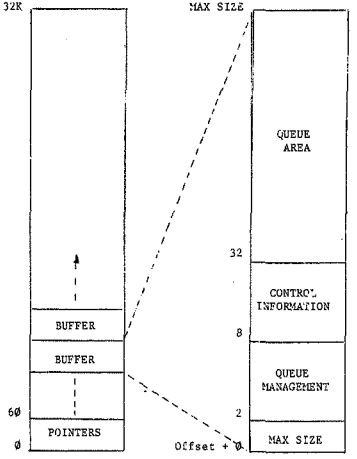


FIGURE 4.2(A)
DUAL PORT MEMORY MAP

FIGURE 4.2(B)
BUFFER STRUCTURE

4.2.2.2 Bus Interface Task

This task may be composed of up to three routines corresponding to the three possible modes; remote terminal, bus controller, and bus monitor. Control is passed from one routine to another (if present) when the terminal changes mode under the conditions specified in the standard.

The RT routine is application independent. It is responsible for:

- Setting up DMA transfers of the correct length to and from the correct location in the dual port memory in response to transmit and receive commands.
- Monitoring the service request subsystem status and busy bits of each subaddress and providing the logical OR of these bits to the hybrid.
- Providing the vector word in response to a transmit vector word mode command.
- Synchronising the subsystem in response to a synchronise mode command with or without an associated data word.
- Correctly setting the dynamic bus control acceptance bit and passing control to the BC routine if a dynamic bus control mode command is received and the terminal is able to assume bus control.

The BC routine will be application dependent. No specific functions are required of a terminal in BC mode by the standard. This routine may pass control to either the RT or BM routines after successfully passing bus control to another terminal on the network.

The BM routine is also application independent. No specific functions are required of a terminal in BM mode, but the terminal is restricted in that it cannot transmit and in that all information gathered is for off line use. The BM routine may pass control to the BC routine in the event that the bus has been sensed to be silent for longer than a specified period.

4.2.2.3 The Subsystem Tasks

These are obviously application dependent. They have access to the data structure in the dual port memory. Suitable primitives could be provided to allow these tasks to enqueue and dequeue messages, and to set and read control information from the buffers.

4.2.2.4 The Multitasking Executive

The requirements of this executive need further consideration, but a fairly simple scheme would probably suffice. Provision is made for a tick or regular interrupt in hardware on which to run the executive.

Since Intel components are being used, a possible choice for an executive could have been the RMX operating system. However, besides being far too complicated, this executive is unsuitable due to the length of time that it disables all interrupts.

A further point about interrupts is that there are certain conditions which will require immediate response from the local processor, such as the receipt of a command by the hybrid when in RT mode. These interrupts must be made high priority and must never be disabled. It will not be

necessary for the service routines of these interrupts to modify the state of any of the tasks, so they will not affect the operation of the executive if they occur while it is rescheduling tasks.

4.2.3 Implementation of Software Scheme

Certain aspects of this software design will have to be tested on the prototype board in order check the viability of the scheme.

In particular the amount of processing that can be done between an interrupt indicating the reception of a command in RT mode and the hybrid requiring a DMA transfer, is important. During this time the processor must determine what type of command it is and, if it is a transmit or receive command, initialise a DMA channel. The DMA initialization involves using the subaddress and transmit/receive bit to get the buffer pointer from t table in the base of the dual port RAM, and using this pointer to obtain a queue input or output pointer (depending on the command received) which is the address to or from which the DMA must be programmed to occur. The word count must also be programmed into the DMA channel.

If the processor proves unable to handle this manipulation in time, a higher level of protocol could be used to implement it. For example, a receive command could be sent to the remote terminal routine to tell it what type of command it will receive next, and to what subaddress it will refer, so that the routine can prepare the DMA channel.

The ability of the processor to handle extra tasks besides the bus interface task must also be tested.

CHAPTER 5

PROTOTYPING AND TESTING

The methods used in the construction and debugging of the prototype, and the differences between the hardware design and the actual prototype terminal are discussed and justified. The methods employed during debugging are covered. Finally, the testing of certain aspects of the design, and the results of these tests are discussed.

5.1 PROTOTYPE CONSTRUCTION

A small amount of initial prototyping was done on breadboard during the design phase, mainly of the dual port arbitration logic.

The main prototype, at the end of the design phase, was constructed using an incremental build-and-debug technique. This has the advantage that any error that occurs can be pin-pointed within the section that is currently being tested (providing that previous sections have themselves been fully debugged). It also builds confidence in the design and prevents repeated errors before construction.

Wire-wrap was used for the following reasons:

- It is well suited to the above build-and-debug technique.
- Wire-wrap is easy to modify. The complex nature of the 1553B bus interface in particular, and the doubt as to the exact behaviour of some of the pins on the Marconi components, made it likely that a fair number of changes would have to made in this area.
- It was felt that better board density could be achieved using wire-wrap than a double-sided p.c. board.

Power supply decoupling was included as follows:

- Two 22 uF capacitors across 5 and 0 volt supplies at their input connections to board.
- One 0.1 uF capacitor across the supply of each major component (processor, memories, PPIs, hybrids etc), for each supply voltage.

- One 0.1 uF capacitor across the 5 and 0 volt supplies of every two SSI and MSI TTL chips.

All wiring was continuity checked prior to the insertion of components. In the case of expensive devices (particularly the Marconi hybrids), all surrounding components were inserted, and the relevant socket was checked for output-to-output connections and correct power supplies.

5.2 PROTOTYPE / DESIGN DIFFERENCES

Due to the board space constraint and the expense and unavailability of certain components, some omissions, and simplifications were made to the full design as presented in the last chapter. The design principles of all the important areas of the terminal are, however, still represented and testable in the prototype. The changes are detailed below.

5.2.1 Generalised I/O Interface

An SBX I/O module having a standard serial interface was required for debugging and testing purposes in the prototype. Such a module was considered too expensive to justify its purchase for this limited use and none could be obtained on loan.

Secondly the motherboard side of the connector required by SBX was unobtainable.

For these reasons, a serial interface was substituted for the SBX interface design. The design of this interface is straightforward. An 8251A USART with standard line drivers

and receivers is used. It is enabled by PCS2*, the same chip select line that the SBX interface used. The transmit and receive baud rate clock is provided by the 186 internal timer 0.

The detailed pin-to-pin diagrams for both the SBX interface and the serial interface are included in appendix B, as it was only when construction of this block was about to begin that the change had to be made.

5.2.2 Multibus Master Interface

The master interface has been omitted for lack of space. Since the interface is of very standard design, and it is expected that most applications will not make use of it, this does not downgrade the prototype substantially.

There is thus no detailed pin-to-pin schematic for this part of the design in appendix D.

5.2.3 Bus Interface Busy Latch

The 16 bit busy latch has been replaced with an 8 bit latch, and a correspondingly smaller data selector is used, to save space. The principle of operation is identical. Few applications will require the use of more than 8 subaddresses.

5.2.4 Dual Port RAM

Only one pair of sockets is supplie. in the dual port memory. This allows a maximum si. i6k which will be sufficient in many applications.

5.2.5 Interrupt Jumper Matrix

The interrupt lines from the SBX interface are obviously not present, while the Transmitter and Receiver Ready lines from the USART are. These could be used as interrupts, or be connected to the unused DMA request line DRQ1 which is routed to the matrix in the absence of SBX.

5.3 DEBUGGING AND TESTING METHODS

The term debugging as used in this chapter refers to the identification and elimination of errors introduced at the schematic drawing, or construction stage, but does not refer to fundamental design errors. Testing refers to the verification of the design itself and its ability to meet the specifications.

A Nicolett Logic Analysis system was extensively used at all stages during construction and testing. This provides both a 16 channel timing analyser and a 48 channel state analyser, both with complex trigger modes which may be coupled together to give a common trigger.

5.3.1 Initial Debugging

The 186 processor together with ROM was the first part of the design to be constructed and debugged. No emulation facilities were available for the 186 processor. The method used to get the processor going, was to place a simple assembler program into EPROM, and then to examine the hardware with the logic analyser.

When the above section was working, it was possible to drive subsequent sections under software. Once the serial interface was added, some control was possible, and a certain amount of observation of the hardware could be done without the use of the logic analyser.

At this stage the method of putting every test program into EPROM was still employed. Assembler was being used rather than PASCAL or C (both of which were available), for its ease in driving the hardware directly. This method soon became cumbersome with longer programs, due to both the problems of debugging the test software itself (usually using the Nicolett's stat analyser), and the turn around time of changing the software. A ROM based monitor was therefore required nable downloading and running of software, setting of breakpoints, and examination of internal states, etcetera, via the serial port.

5.3.2 Use of Forth as a Development Tool

The above requirements were fulfilled with many additional advantages by the use of Forth.

A version of 8086 fig-Forth was used. This had to be *configured*, and a custom initialisation routine had to be provided, for the particular hardware. This version copies the Forth system from ROM into RAM where it can run, and then passes control to it. The Forth system requires approximately 6k in both ROM and RAM. However, in order to make any real use of it, at least twice the amount of RAM is needed.

The decision to use Forth was made after its use was demonstrated in a similar application, although its real

power was not appreciated until after some use. Forth, a threaded interpretive language, is both an operating system of sorts, and a language which has high level constructs but allows low level manipulation. It's main advantage however, is it's extensiblity. Extra words to perform a desired function can be added to the system using previously defined words. If the function cannot be achieved using previously defined words, or the function is time critical, a new primitive word written in assembler may be added to the system.

The construction of the required monitor functions is thus a straight-forward matter. The real power of Forth as a development tool is realised when these basic monitor functions are used to build more complex diagnostic features. Very little time is required to learn to use Forth effectively for this type of application, and having done so, such extended features can be created very rapidly.

The version of Forth used, had already had most of the required assembler primitives already built in. A few extra primitives were added to drive the 1553B bus interface in time critical situations. A Forth utility had also been written to download machine code in Intel hex format.

The use of Forth thus made the debugging during the final stages of the prototype construction, and the testing of the design, far quicker and less laborious than would have been possible with a simple monitor, or even an emulator.

5.4 DESIGN TESTING AND RESULTS

Only the testing of those areas of the design which are novel, is included in this section. These are the MIL-STD-1553B bus interface and the dual port memory. The SBX interface would also have been covered, had this been included in the prototype. However, this is a fairly simple circuit and is unlikely to contain any major errors.

5.4.1 Dual Port Memory

This was debugged and tested with the aid of a single board computer on Multibus.

One of the PALs had a design error in that the data transceiver on the local port was being enabled on all 186 bus cycles, not only those to the dual port memory. While this made no difference to a write cycle, bus contention would occur during a read. No testing was necessary to detect this fault, since as soon as the transceivers were inserted, the processor failed to run. The solution was to qualify the transceiver enable signal by the 186 dual port select line, which could fortunately be achieved temporarily, using external components, without having to reprogram the PAL before testing could continue.

With this problem solved, access to the dual port memory from both individual ports was tested in the same way that any memory test would be done; by writing to the memory and then checking that what is read back is correct. The same test was then done across the memory from one port to the other.

Apart from exposing any wiring bugs on the data and address busses, this checked that the arbitration logic was allowing both ports read and write access to the memory. Two bugs were found in the other PAL at this stage. These were also temporarily correctable by inverting two inputs. With these corrections the control logic could be assumed to be correct.

In order to test the arbitration logic fully, both processors were then set to perform continuous reads and write on the memory. This test was still running after a long period of time with neither processor hanging. The arbitration logic waveforms, as observed by the Nicolett timing analyser, also indicated that all was correct.

Finally, to test the locking feature designed into the arbitration, both processors were made to execute a simple routine which performed locked exchange transfers on a semaphore flag in the dual port memory (see Forth code in appendix F). With this test, both processors would hang every few minutes, indicating that each thought that the other had set the semaphore. The timing analyser showed all waveforms to be exactly as expected, and no fault could be found until the analyser was set to trigger on two access requests occurring very close together. If a Multibus request in one period of the 18 MHz arbitration clock was followed by a local request in the next, a race around condition would occur in which the arbitration logic oscillated between the ports (irrespective of the lock line settings), until one port removed its request (up to 150 microseconds later!).

The fault was due to the fact that two latches were used in the logic, one to synchronise and present a stable input to the other which latches the result of the arbitration cycle. The one clock period delay in the

signal fed back (via some combinational logic) from the second latch, was causing the condition to occur. The second latch was replaced by a Schmitt trigger to ensure that the arbitration output would assume one or the other state even if the setup and hold times on the remaining latch were violated.

With all errors found, the PALs were correctly reprogrammed.

5.4.2 1553B Bus Interface

After the circuitry surrounding the MRTU 53045 hybrid was debugged and working as designed, the hybrid was inserted, and bus interface tested as a whole.

Since only one prototype terminal had thus far been produced, these tests involved driving the hybrid in bus controller mode. Only bus control options 1 and 2, and (partially) option 0 could be tested at this stage. The hybrid performed as expected except that:

- When a receive command was initiated using option 0, the command would be transmitted correctly, but the hybrid would not assert its Data Transfer Request line or attempt to strobe in data from the TX latches. After carefully checking the signal timing against that given in the MRTU literature, it was concluded that this was a bug in the hybrid. Suggested solutions such as toggling the RT/BC* line a few times during initialisation made no difference, and this problem remained unresolved at this stage.

- As soon as the hybrid was powered up, it would begin to send pulses from its RESET line. These would prevent the processor from ever getting through the initialisation code. If this line was disconnected, however, and the processor was allowed to complete the initialisation code, the pulses would disappear. This is obviously another bug in the hybrid. The problem was resolved by masking off the RESET line and enabling it after initialisation.

Testing was done at this stage using assembler, an example of which is included in appendix F.

Since the problems at this stage were due to the hybrid rather that the design of its surrounding circuitry, and since an option 0 transmit command can be simulated (rather more tediously) by using option 1 and 2 commands, a second prototype was constructed in order to enable the testing of the terminal in remote terminal mode. The second prototype only includes those parts of the design necessary to implement the 1553B bus interface and run Forth. The dual port memory, Multibus slave interface, and external PIC are not included (but could be added).

The same two bugs were present in the second hybrid, however the first problem was solved as soon as the two prototypes were connected up together on a bus. It turned out that, for some reason, as soon as a hybrid had received a transmission from another terminal, it would work correctly.

It was thus possible to test the bus interface in remote terminal mode. Not every possible command combination was tried, but a representative selection was tried and all were found to work. Testing at this stage was done with

Forth using assembler primitives which were added to the end of the dictionary in ROM to drive the hybrid during time critical periods. These primitives and the Forth source used are included in appendix F.

A design error was found with the DMA request line. This was due to the speed with which the DMA channel, once programmed, is able to service requests even when programmed with the destination synchronisation option (where the bus is automatically relinquished and must be regained after every transfer). On requests for a word to be transmitted, the DMA request line DMARQ was not negated until the Data Transfer Request line was negated, during which time the DMA would transfer two or sometimes three words to the TX latch. The hybrid would only transmit the last of these on the network. The problem solved when the request was modified to be removed immediately a word was written to the TX latch.

With the 1553B bus interface design thus verified to be functionally correct, some tests were done on the ability of the processor to service, in the required time, an interrupt indicating a command had been received. Servicing of this interrupt is described in section C.3.13.1.

A simplified service routine was written first, to extract the required information from the MRTU 53045 and use it to determine the command type and, if necessary, fully program the DMA channel. No use of indirect access into a buffer structure, as proposed, was attempted initially. Several versions of the routine were tried using whatever features and tricks with the assembly language as could be found. An example of the assembler interrupt service routine is included in appendix F.

Forth using assembler primitives which were added to the end of the dictionary in ROM to drive the hybrid during time critical periods. These primitives and the Forth source used are included in appendix F.

A design error was found with the DMA request line. This was due to the speed with which the DMA channel, once programmed, is able to service requests even when programmed with the destination synchronisation option (where the bus is automatically relinquished and must be regained after every transfer). On requests for a word to be transmitted, the DMA request line DMARQ was not negated until the Data Transfer Request line was negated, during which time the DMA would transfer two or sometimes three words to the TX latch. The hybrid would only transmit the last of these on the network. The problem was solved when the request was modified to be removed immediately a word was written to the TX latch.

With the 1553B bus interface design thus verified to be functionally correct, some tests were done on the ability of the processor to service, in the required time, an interrupt indicating a command had been received. Servicing of this interrupt is described in section C.3.13.1.

A simplified service routine was written first, to extract the required information from the MRTU 53045 and use it to determine the command type and, if necessary, fully program the DMA channel. No use of indirect access into a buffer structure, as proposed, was attempted initially. Several versions of the routine were tried using whatever features and tricks with the assembly language as could be found. An example of the assembler interrupt service routine is included in appendix F.

With the hardware arrangement, the first word for a transmit command must be written to the TX latch within 20,4 microseconds of the INCMD interrupt line being asserted, and for a receive command, the first word must be fetched from the RX latch within 38,5 microseconds. The latter time could be achieved for the simplified scheme, but even with the most compact service routine was unable to meet the time for the transmit command on every occurrence (interrupt service latency played a part here). The situation is eased slightly by the use of the NBGT* line instead of INCMD* as a source of the interrupt, as it occurs approximately 1,5 microseconds earlier, but this does not solve the problem.

The results of these tests therefore show that the processor, driving the current hardware arrangement, is nowhere near fast enough to implement the software scheme outlined in the design overview without the addition of a higher level of protocol. Possible solutions to this problem are discussed in the conclusion.

CHAPTER 6

RECOMMENDATIONS AND CONCLUSIONS

The terminal design is critically reviewed. Possible alterations, additions and exclusions to improve the terminal design are discussed and a prediction is made as to the final form of the terminal. Suggestions are made for further work on the terminal. Finally, the main points arising from the design are summarised.

6.1 DESIGN REVIEW

The design provides a usable, and compact, general purpose MIL-STD-1553B data bus terminal that meets almost every aspect of the specification. However, in the light of the experience with the prototype, the terminal design falls short of that required of it in some respects. This may partly be ascribed to a specification which can now be seen to be unrealistic in terms of the constraints imposed on the terminal.

The main criticism of the terminal design and specification may be levelled at the following aspects:

Complexity of the proposed software scheme. In particular, the complexity of the dual port memory data structure. The overhead in getting data in and out of the buffer queues is substantial, thus making it impossible to transfer to or from the structure directly, in response to an interrupt from the network.

Bus interface software overhead. Because of the need to conserve space, some operations that could have been done in hardware were left to software. Examples of this are; the use of only one pair of latches for passing words between hybrid and subsystem, and the use of an external DMA channel rather than the hybrid's DMA type lines. Reasons for these decisions were given in the design. It should also be noted, that the interface between the 1553B bus and the subsystem must, of necessity, be a software one (otherwise the terminal hardware would be specific to a particular subsystem).

Required system configurations. The range of configurations in which the terminal can operate, as given in the functional specification, necessitates the

inclusion of both the Multibus master and slave interfaces, as well as the generalised I/O interface. Considering the space limitations, this is unrealistic.

Cost. Although this was never stated as a restriction, it should be noted that the terminal designed will be very expensive to produce.

With the exception of the cost, the above problems are probably due to two factors; underestimation of the space constraint, and failure to recognise the limitations of the 186 processor or to relate the hardware more closely to its capabilities.

6.1.1 Possible Design Improvements

As mentioned in the specification chapter, the next prototype, if one is built, will first require a revised specification taking into consideration the above points. Possible hardware improvements that could be implemented in a second prototype are:

Either the Multibus master interface, or generalised I/O interface should be omitted. They have a similar function, namely to allow on-board subaddress tasks to communicate with I/O devices. The Multibus interface the is more flexible of the two as it allows access to bigger address space, and a wider range of I/O devices.

If the Multibus interface is retained, the address bus could also be routed via the corresponding dual port bus, as has been done with the data bus. This would require that the address buffers on the Multibus port be substituted by transceivers, and that the appropriate controls be generated by the dual port abitration and

control logic.

If the generalised I/O interface in the form of SBX is retained, most of the circuitry involved could be implemented in a PAL.

The layout of the input lines on the bus interface PPIs could be improved upon. The difficulty in manipulating words read-in using the current arrangement, can be seen from the interrupt service routine in appendix F. A better layout would reduce the time taken to program the DMA channel.

Finally some extra LEDs (besides the four used to indicate what mode the terminal is in), could be added to indicate such things as error conditions in the terminal.

6.2 FUTURE WORK ON TERMINAL

Before the construction of a second prototype, or modification of the first one, the proposed terminal software design should be implemented and tested.

A method of overcoming the problem of access to the dual port memory data structure would have to be devised.

One possibility is the use of a higher level of protocol. This was mentioned under the software design overview. It involves building one transfer out of two. The first transfer is always a receive command of specified length and format, containing the information required to access the relevant table and set up the DMA channel in preparation for the next transfer. This has a major disadvantage in that special provision would have to be made in order to use this terminal in any network in which

this protocol is not normally implemented.

There is a second possibile hardware method of solving this problem. That is by using separate DMA channels for transmitting and receiving. This would mean that both channels could be kept partially set-up for their respective operations, and the time to program them would be reduced. When PAL 2 on the bus interface was reprogrammed for the last time to eliminate some errors, an extra line TXDMARQ was added on an unused output to enable this to be done. (For an RX DMA request, simply use the RXINT line.)

Another point that should be investigated, is the writing of software that makes use of the software watchdog.

The results of these investigations will indicate any other hardware alterations that *should be made, besides* those suggested in the previous section. A final version of the software (probably more simple than the one proposed) could be designed, and implemented taking into consideration any hardware changes that were made.

6.3 CONCLUSIONS

Several conclusions and observations arise from the work done on this project.

There is a fair range of MIL-STD-1553B components available from British and American manufacturers, and the indications are that the standard is beginning to find industrial applications. Of the components surveyed, none was found that would have both completely implemented the 1553B protocol, and given access to the reserved parts of the standard.

The Marconi 1553B chip set, in particular the hybrid version thereof, provides a full implementation of the 1553B protocol, with all command servicing that does not involve the subsystem done internally. It is compact and has a reasonably straight-forward hardware interface. It proved therefore, to be almost ideal for the requirements of this terminal. It is not completely bug free, but those bugs that were found could be easily solved.

The Intel 80186 processor is a very useful device. It is a fully 8086 compatible microprocessor, while having the most common peripheral devices on-board. Considering the equivalent individual components, it is very reasonably priced. It is highly suited to medium complexity microprocessor systems, the development time of which should be significantly reduced. It was also ideally suited to the hardware requirements of this project.

The use of PALs allows the replacement of a large number of small TTL chips by a single component. Larger PALs than those used in this project will shortly be available and a further prototype may be able to achieve an even more compact implementation by using them. The PAL design method employed in this design, however, proved laborious and error prone. If any major use of PALs is planned, a *design aid is essential*.

Finally, a MIL-STD-1553B data bus terminal has been designed that should, with some further work on software aspects, satisfactorily meet the requirements of the problem statement. It is hoped that the component survey, design, and test results presented in this report, will form a useful contribution to the local knowledge of MIL-STD-1553B and the major components used.

APPENDIX A

MIL-STD-1553B BUS INTERFACE COMPONENTS

A.1 STC RANGE

These components are based on a chip set designed by Smith Industries and MCE.

A.1.1 FC15532 Remote Terminal Unit

This is a full protocol unit in a single hybrid package. It has a built in analog section and a dual redundant facility. The data bus is 16 bits wide and is bufferd by a FIFO memory large enough to hold an entire message. There are fourteen control and status lines to interface to the subsystem but all routine bus protocol (mode codes etc.) are handled internally. There is, however, no way of initializing transmission on the bus, thus the unit cannot be used as a bus controller.

A.1.2 FC15531 Remote Terminal Unit

As above, but with only one bus interface.

A.1.3 FC15533 Txrx Encoder/Decoder Module

This device is also housed in a single package.

On the receiver side, it performs the functions of receiving words from the bus (while checking Manchester,

parity and sync), indicating type of sync, RT address matching and broadcast detection.

On the transmitter side, it adds the required sync and parity to each word to be transmitted. Finally it has a transmission time out facility. Like the 15532, it has a built in analog section but only interfaces to one bus.

The device implements or enforces no protocol. It could be used in a bus controller since transmission of any word may be initiated. It has a 16 bit data bus with a single word buffer.

A.1.4 FC15535 Bus Driver/Receiver

Single driver/receiver with a transmission time out facility.

A.1.5 FC1553TI Transformer

Isolation transfromer with a turns ratio of 1,41:1 conforming to 1553B requirements.

A.2 MARCONI/CTI RANGE

A wide range of products is available based on an Iso-CMOS LSI chip set designed by MEDL (Marconi Electronics Devices Limited). CTI is a Marconi U.S. subsidiary.

A.2.1 CT1088 Chip Set

This set allows use as a passive monitor, remote terminal, and bus controller. It implements the digital part of the bus interface only and separate analog and transformer sections have to be added. The minimal configuration is four chips: an internal control logic unit, an interface unit, an encoder and a decoder. This interfaces to a single bus. Further decoder chips can be added, up to a maximum of four, to interface to extra busses.

These chips implement the full protocol and only the protocol. In addition the no response time-out and contents of the BIT (built-in-test) word are defined and fixed. All routine (mode code) processing is performed internally to the chip set. An extensive self-test and transmitter time-out are also included. The data bus is only 8 bits wide and is bufferd to the level of a single data word.

A.2.2 CT1089 Chip Set

This is the same as the above but can be used only for remote terminals.

A.2.3 Hybrid Versions

Both manufacturers have produced some hybrid versions of these chip sets. MEDL have the MRTU 53045, MRTU 53040, MRTU 53055 and MRTU 53050, while CTI sell the CT1602 and CT1610. Some of these incorporate some extra features such as: on board oscillator, bus select logic, TTL buffering of control and data lines and an extra decoder chip.

A.2.4 CT1555 Data Terminal Bit Processor

This device does not implement the 1553B protocol. It could be used for both bus monitor and bus controller in addition to the remote terminal mode.

On the receive side it performs address, broadcast, sync and mode code recognition, as well as Manchester and parity validation. All command, status, and data words are received. For transmission, no restriction is placed on the word type. The desired sync and correct parity will simply be added to any word to be transmitted.

All words are passed to and from the subsystem via an 8 or 16 bit bus with one level of word buffering. Self test and transmitter time-out are included.

A.2.5 Analog Sections And Transformers

A very large range of hybrid driver/receiver sections is manufactured. These are compatible with the chip sets but could be used with other digital sections. Suitable transformers allowing both direct and transformer coupled stubs are available.

A.3 GRUMMAN/SMC BUS INTERFACE UNIT

This is a single chip design by Grumman Aerospace and Standard Microsystems Corporations which form the digital section of an interface to a single bus.

The Bus Interface Unit (BIU) can be viewed as an I/O processor. Commands for the processor are written into the

correct place in memory by a host processor. A line is then strobed and the I/O processor will request the local parallel bus, fetch the command, and process it. The BIU thus operates exclusively through 16 bit word DMA transfers at the parallel interface and all communication with the host processor is via RAM.

The BIU can be used as a bus controller or remote terminal. The initial status and remote terminal address must be loaded from memory. Operating as a remote terminal, internal decoding and control is provided for only five mode codes. Reserved mode codes and status bits may be used and the contents of the BIT word is definable. The usual validity checks are performed. In bus controller mode there is an extra check done automatically that the remote terminal that replies to a command, is the one to which the command was addressed.

A.4 HARRIS RANGE

The Harris HD-6408 is a Manchester encoder/decoder which could form the nucleus of a bus interface. It recognises and indicates the type of sync, checks for Manchester and parity errors, and passes the word to external circuitry in serial form. In the case of transmission, data to be transmitted is input in serial form and the desired type of sync is added before transmission. No analog circuitry is included.

The HD-6408 has commercial temperature ratings. If military ratings are required, the HD-15530-8 is the functionally equivalent device. If a higher data rate is required, the HD-15531B will operate up to 2,5 Mbits/sec.

correct place in memory by a host processor. A line is then strobed and the I/O processor will request the local parallel bus, fetch the command, and process it. The BIU thus operates exclusively through 16 bit word DMA transfers at the parallel interface and all communication with the host processor is via RAM.

The BIU can be used as a bus controller or remote terminal. The initial status and remote terminal address must be loaded from memory. Operating as a remote terminal, internal decoding and control is provided for only five mode codes. Reserved mode codes and status bits may be used and the contents of the BIT word is definable. The usual validity checks are performed. In bus controller mode there is an extra check done automatically that the remote terminal that replies to a command, is the one to which the command was addressed.

A.4 HARRIS RANGE

The Harris HD-6408 is a Manchester encoder/decoder which could form the nucleus of a bus interface. It recognises and indicates the type of sync, checks for Manchester and parity errors, and passes the word to external circuitry in serial form. In the case of transmission, data to be transmitted is input in serial form and the desired type of sync is added before transmission. No analog circuitry is included.

The HD-6408 has commercial temperature ratings. If military ratings are required, the HD-15530-8 is the functionally equivalent device. If a higher data rate is required, the HD-15531B will operate up to 2,5 Mbits/sec.

These components are thus useful for building up simple terminals with limited functions.

A.5 ILC DATA DEVICE CORPORATION HYBRID SET

This is a group of several hybrids and major chips which together form the BUS-65500 bus interface unit. This unit interfaces to a Motorola 68000 VME Bus and can be obtained on a VME Double-sized Eurocard. The individual components however could be used to implement a terminal at a desired level, ranging from a simple serial Manchester to parallel data encoder/decoder (with some extra features), to the full terminal without being 68000 VME specific.

The components are:

A.5.1 BUS-25679 Isolation transformers

These allow both direct stub coupling and transformer stub coupling.

A.5.2 BUS-65101 Dumb Interface Hybrid

This is built up from the Harris HD-15530 (see above) and two DDC custom chips. This performs the basic Manchester encoding and decoding, word validation, broadcast recognition, address recognition, mode code recognition (but not decoding) and transmitter time out. There is an on board clock generator and the data I/O is 8 or 16 bit tri-state parallel or serial.

A.5.3 BUS-66101 Protocol 1 Hybrid

This is used if dual 1553B busses are required. It multiplexes two BUS-65101 hybrids. Both channels are monitored for valid command words containing the correct remote terminal address and the active channel is flagged.

A.5.4 BUS-66102 Protocol 2 Hybrid

This hybrid along with an external state sequencer implements the remote terminal function. It has the various status, command and last command registers, and buffers as well as a word counter which is automatically incremented with every word. It also offers a full set of handshake lines that can be used for direct data transfers. The protocol 2 hybrid responds to eleven of the fifteen defined MIL-STD-1553B mode codes.

A.5.5 BUS-66106 Protocol 3 Hybrid

This hybrid along with its own state sequencer performs the bus controller function. It will send out a command word with or without data as instructed in a special control word sent to it by an external processor. The returning status word is monitored and validated for response time, continuous message format, incorrect remote terminal address, and bits set in the status word. If an error is found, an interrupt is caused.

A.5.6 BUS-66103 DMA Controller Hybrid

This interfaces any of the three protocol hybrids that may be present with a 4K by 16 bit dual port RAM which must

be provided. Memory is allocated in blocks sufficient for storing two full length messages. Each block is managed as double buffered message location. The allocation of blocks is flexible and is set up by an external CPU.

A.5.7 BUS-66107 VME Programmed I/O Interface Hybrid

Interfaces the three protocol hybrids and the dual port memory with the VME bus signals.

A.6 ROCKWELL-COLLINS 1553 INTERFACE DEVICE

This design breaks a MIL-STD-1553B into 4 blocks two of which are application independant and are implemented by components designed by Rockwell-Collins (namely Analog Transmit/Receive and Digital Transmit/Receive), and two of which are application dependant and are left up to the designer (Subsystem Interface and Interface Controller).

These application dependant sections must provide the following:

Subsystem Interface. This must allow the interface controller access to the subsystem and tailor the controller's response to bus commands which change on an application to application basis (eg. legal/illegal command detection).

Interface Controller. The interface controller performs message validation and data routing to and from the subsystem via the subsystem interface.

The components implement a full protocol dual redundant remote terminal. No bus control capability is available. The two components are:

A.6.1 Analog Transmit/Receive

This performs the usual analog driver/receiver function with the addition of the feature that the transmitter circuit reclocks the data and control signals. This allows the digital section to be located some distance away, and still avoid problems of data pulse width and timing skew.

A.6.2 Digital Transmit/Receive

This contains completely redundant Manchester encoders, buffered input registers, buffered status register, buffered transmit register and serial Manchester encoder. It is LSTTL compatable and interfaces to both 8 and 16 bit processors. It is implemented in a 48 pin CMOS device.

APPENDIX B

MRTU 53045 - SUBSYSTEM INTERFACE AND OPERATION

The following summary of the operation of the MRTU 53045 hybrid is condensed from the two main descriptive references on the Marconi 1553B Chip Set [19,20].

B.1 REMOTE TERMINAL OPERATION

B.1.1 Receive Data Operation

All valid data words associated with a valid receive data command word for the RT are passed to the subsystem. The RT examines all command words from the bus and will respond to valid (ie. correct Manchester, parity coding etc.) commands which have the correct RT address (or broadcast address if the RT broadcast option is enabled). When the data words are received, they are decoded and checked by the RT and, if valid, passed to the subsystem on a word by word basis at 20 microsecond intervals. This applies to receive data words in both Bus Controller to RT, and RT to RT messages.

When the RT detects that the message has finished, it checks that the correct number of words have been received and if the message is fully valid, then a Good Block Received signal is sent to the subsystem, which must be used by the subsystem as permission to use the data just received. If a block of data is not validated, then Good Block Received will not be generated. This may be caused by any sort of message error or by a new valid command for the RT being received on another bus to which the RT must

switch. If no GBR* signal is generated, then an error has been detected by the RT and the entire data block is invalid and no data words in it may be used.

B.1.2 Transmit Data Operation

If the RT receives a valid transmit data command addressed to the RT, then the RT will request the data words from the subsystem for transmission on a word by word basis. To allow maximum time for the subsystem to collect each data word, the next word is requested by the RT as soon as the transmission of the current word has commenced.

It is essential that the subsystem should provide all the data words requested by the RT once a transmit sequence has been accepted. Failure to do so will be classed by the RT as a subsystem failure and reported as such to the Bus Controller.

B.1.3 Control of Data Transfers

This section describes the detailed operation of the data transfer mechanism between RT and subsystems. It covers the operations of the signals DTRQ*, DTAK*, IUSTB, H/L*, GBR*, NBGT*, TX/RX* during receive data and transmit data transfers.

B.1.3.1 Receive Data Transfers

When the RT has fully checked the command word, NBGT* is pulsed low, which can be used by the subsystem as an initialisation signal. TX/RX* will be set low indicating a receive command. When the first data word has been

fully validated, DTRQ* is set low. The subsystem must then reply within approximately 1.5 microseconds by setting DTAK* low. This indicates to the RT that the subsystem is ready to accept data The data word is then passed to the subsystem on the internal highway IH08-IH715 in two bytes using IUSTB as a strobe signal and H/L* as the byte indicator (high byte first followed by low byte). Data is valid about both edges of IUSTB.

If the RT is receiving data in an RT to RT transfer, the data handshaking signals will operate in an identical fashion but there will be a delay of approximately 70 microseconds between NBGT* going low and DTRQ* first going low.

B.1.3.2 Transmit Data Transfers

As with the receive command discussed previously, NBGT* is pulsed low if the command is valid and for the RT. TX/RX* will be set high indicating a transmit data command. While the RT is transmitting its status word, it requests the first data word from the subsystem by setting DTRQ* low. The subsystem must then reply within approximately 13.5 microseconds by setting DTAK* low. By setting DTAK* low, the subsystem is indicating that it has the data word ready to pass to the RT. Once DTAK* is set low by the subsystem, DTRQ* should be used together with H/L* and TX/RX* to enable first the high byte and then the low byte of the data word onto the internal highway IH08-IH715. The RT will latch the data bytes during IUSTB, and will then return DTRQ* high. Data for each byte must remain stable until IUSTB has returned low.

For both receive and transmit command processing, if the subsystem does not declare itself busy, then it must

respond to DTRQ* going low. Failure to do so will be classed by the RT as a subsystem failure and reported as such to the Bus Controller.

B.1.4 Additional Data Information Signals

At the same time as data transfers take place, a number of information signals are made available to the subsystem. These are INCMD, the subaddress lines SA0-4, the word count lines WC0-4 and current word count lines CWC0-4. Use of these signals is optional.

INCMD* will go active low while the RT is servicing a valid command for the RT. The subaddress, transmit/receive bit, and word count from the command word are all made available to the subsystem as SA0-4, TX/RX* and WC0-4 respectively. They may be sampled when INCMD goes low and will remain valid while INCMD is low.

The subaddress is intended to be used by the subsystem as an address pointer for the data block. Subaddress 0 and 31 are mode commands, and there can be no receive or transmit data blocks associated with these. (Any data word associated with a mode command uses different handshaking operations.)

The word count tells the subsystem the number of words to expact to receive or transmit in a message, up to 32 words. A word count of all 0s indicates a count of 32 words.

The current word count is set to 0 at the beginning of a message and is incremented following each data word transfer across the RT-subsystem interfact. (It is clocked on the falling edge of the second IUSTB pulse in

each word transfer). It should be noted that there is no need for the subsystem to compare the word count and current word count to validate the number of words in a message. This is done by the RT.

B.1.5 Subsystem Use of Status Bits and Mode Commands

Use of the status bits and the mode commands is one of the most confusing aspects of MIL-STD-1553B. This is because much of their use is optional and also because some involve only the RT while others involve both the RT and the subsystem. The MRTU 53045 allows full use to be made of all the status bits, and also implements all the mode commands. The subsystem is given the opportunity to make use of status bits, and is only involved in mode commands which have a direct impact on the subsystem.

The mode commands in which the subsystem may be involved are Synchronise, Synchronise with data word, Transmit Vector Word, Reset and Dynamic Bus Control Allocation. The status bits to which the subsystem has access are Service Request, Busy, Subsystem Flag and Dynamic Bus Control Acceptance. The subsystem designer should note that all other mode commands and status bits are serviced internally by the RT, and the subsystem has no access to them. In particular, the terminal flag and message error status bits and BIT word contents are all controlled internally by the RT.

B.1.5.1 Synchronise Mode Commands

Once the RT has validated the command word and checked for the correct address, the SYNC* line is set low. The signal WC4 will be set low for a Synchronise mode command

and high for a Synchronise with data word mode command. In a Synchronise with data word mode command, SYNC* remains low during the time that the data word is received. Once the data word has been validated, it is passed to the subsystem on the internal highway IH08-IH715 in two bytes using IUSTB as a strobe signal and H/L* as the byte indicator (high byte first followed by low byte). SYNC* being low should be used on the enable to allow IUSTB to clock synchronise mode data to the subsystem.

If the subsystem does not need to implement either of these mode commands, the SYNC* signal can be ignored, since the RT requires no response from the subsystem.

B.1.5.2 Transmit Vector Word Mode Command

The RT requests data by setting VECTEN* low. The subsystem should use H/L* to enable first the high byte and then the low byte of the Vector word onto the internal highway IH08-IH715.

It should be noted that the RT expects the Vector word contents to be already prepared in a latch ready for enabling onto the internal highway when VECTEN* goes low. If the subsystem has not been designed to handle the Vector word mode command, it will be the fault of the Bus Controller if the RT receives such a command. Since the subsystem is not required to acknowledge the mode command, the RT will not be affected in any way by Vector word circuitry not being implemented in the subsystem. It will however transmit a data word as the Vector word, but this word will have no meaning.

B.1.5.3 Reset Mode Command

Once the command word has been fully validated and serviced, the RESET* signal is pulsed low. This signal may be used as a reset function for subsystem interface circuitry.

B.1.5.4 Dynamic Bus Allocation

This mode command is intended for use with a terminal which has the capability of configuring itself into a bus controller on command from the bus. The line DBCREQ* cannot go true unless the DBCACC* line was true at the time of the valid command, ie. tied low.

B.1.5.5 Use of the Busy Status Bit

The Busy Bit is used by the subsystem to indicate that it *is not ready to handle data transfers* either to or from the RT.

The RT sets the bit to logic one if the BUSY* line from the subsystem is active low at the time of the second falling edge of INCLK after INCMD* goes low. Once the Busy Bit is set, the RT will stop all receive and transmit data word transfers to and from the subsystem. The data transfers in the Synchronise with data word and Transmit Vector word mode commands are not affected by the Busy Bit and will take place even if it has been set.

It should be noted that a minimum of 0.5 microseconds subaddress decoding time is given to the subsystem before setting of status bits. This allows the subsystem to selectively set the Busy Bit if for instance one

subaddress is busy but others are ready. This option will prove useful when an RT is interfacing with multiple subaddresses.

B.1.5.6 Use of the Service Request Status Bit

The Service Request bit is used by the subsystem to indicate to the Bus Controller that an asynchronous service is requested. The timing of the setting of this bit is the same as the Busy Bit. Use of SERVREQ* has no effect on the RT apart from setting the Service Request bit.

It should be noted that certain mode commands require that the last status word be transmitted by the RT instead of the current one, and therefore a currently set status bit will not be seen by the Bus Controller. Therefore the user is advised to hold SERVREQ* low until the requested service takes place.

B.1.5.7 Use of the Subsystem Status Bit

This status bit is used by the RT to indicate a subsystem fault condition. If the subsystem sets SSERR* low at any time, the subsystem fault condition in the RT will be set, and the Subsystem Flag status bit will subsequently be set. The fault condition will also be set if a handshaking failure takes place during a data transfer to or from the subsystem. The fault condition is cleared on power-up or by a Reset mode command.

B.1.5.8 Dynamic Bus Control Acceptance Status Bit

DBCACC*, when set true, enables an RT to configure itself into a Bus Controller, if the subsystem has the capability, by allowing DBCREQ* to pulse true and BIT TIME 18 to be set in the status response. DBCACC* tied high inhibits DBCREQ* and cleares BIT TIME 18 in the status response.

B.2 BUS CONTROLLER OPERATION

To enable its use in a bus controller each chip in the hybrid has additional logic within it. This logic can be enabled by pulling the pin labelled RT/BC* low. Once the hybrid is in bus control mode, all data transfers must be initiated by the bus control processor correctly commanding the hybrid via the subsystem interface. In bus control mode six inputs are activated which in RT mode are inoperative and four signals with dual functions exercise the second function (the first being for the RT operation).

To use the MRTU 53045 as a 1553B bus control interface, the bus control processor must be able to carry out four basic bus-related functions. Two inputs, BCOPA and BCOPB allow these four options to be selected. The option is then initiated by sending a negative-going strobe on the BCOPSTB* input. BCOPSTB* must only be strobed low when NDRQ* is high. This is particularly important when two options are required during a single transfer.

With these options all message types and lengths can be handled. Normal BC/RT* exchanges are carried out in the hybrid option zero. This is selected by setting BCOPA and BCOPB to a zero and strobing BCOPSTB*. On receipt of the

strobe, the hybrid loads the command word from an external latch using CWEN and H/L*. The command word is transmitted down the bus. The TX/RX* bit is, however, considered by the hybrid as being its inverse and so if a transmit command is sent to an RT, the hybrid in BC mode believes it has been given a receive command. As the RT returns the requested number of data words plus its status, the BC hybrid carries out a full validation check and passes the data into the subsystem using DTRQ*, DTAK*, H/L*, IUSTB and CWC as in RT operation. It also supplies GBR at the end of a valid transmission. Conversely, a receive command sent down the bus is interpreted by the BC hybrid as a transmit command, and so the requisite data words are added to the command word.

For mode commands, where a single command word is required, option one is selected by strobing BCOPSTB* when BCOPA is high and BCOPB is low. On receiving the strobe, the command word is loaded from the external latch using CWEN and H/L*, the correct sync and parity bits are added and the word transmitted. Mode commands followed by a data word requires option two. Option two, selected by strobing BCOPSTB* while BCOPA is low and BCOPB is high, loads a data word via DWEN* and H/L*, adds sync and parity and transmits them to the bus. If the mode code transmitted required the RT to return a data word, then selecting option three by strobing BCOPSTB* when BCOPA and BCOPB are both high will identify that data word and if validated, output it to the subsystem interface using RMDSTB and H/L*. This allows data words resulting from mode codes to be identified differently from ordinary data words and routed accordingly. All received status words are output to the subsystem interface using STATSTB and H/L*.

RT to RT transfers require the transmission of two command words. A receive command to one RT is contiguously followed by a transmit command to the other RT. This can be achieved by selecting option one followed by option zero for the second command. The strobe (BCOPSTB*) for option zero must be delayed until NDRQ* has gone low and returned high following the strobe for option one. The RT transmissions are checked and transferred in the subsystem interface to the bus control processor.

B.3 BUS MONITOR OPERATION

In BC option three, if the signal PASMON* is active, then all data appearing on the selected bus is output to the subsystem using STATSTB for command and status words or RMDSTB for data words.

APPENDIX C

HARDWARE DESIGN

The implementation of each of the blocks that was defined in the hardware design overview is now discussed. The design thus presented represents the latest version after the correction of all design errors discovered during prototyping. Where appropriate, the software approach to driving the hardware is explained.

C.1 LOCAL PROCESSOR

Considering the speed requirements, the fact that a 16 bit processor is desirable, and the restriction that Intel components must be used, a member of the 8086 microprocessor family is the obvious choice for the local processor. The 80186 has a slightly enhanced 8086 instruction set and is available in an 8 MHz version. (In fact 10 MHz versions are now available from second sources.)

The main reason for the choice of this processor, however, is the fact that it has, on board, peripherals which fulfill almost all the requirements of the design. It has two independent DMA channels, a programmable interrupt controller, three programmable interval timers, clock generator, and programmable chip select and ready generation logic. The operation of these on board peripherals is very similar to that of the equivalent Intel peripheral chips. For a useful description of the entire device, including information not found in the data sheet, the application note AP-186 [27] may be consulted.

Considerable space saving is thus achieved by the use of this micro'. Apart from that saved on the large devices (DMA, interrupt controller, interval timers), the programmable chip select and ready generation logic saves the use of a vast number of SSI and MSI TTL chips.

This logic also allows for a great deal of flexibility. The memory and I/O maps of the local processor system, and the speed of devices in it, can be configured under software control. This is particularly convenient where provision for more than one size or speed of device has been made, as in the case of the local memory (see section C.2).

Figure C.1 is a diagram of the 186 and surrounding circuitry.

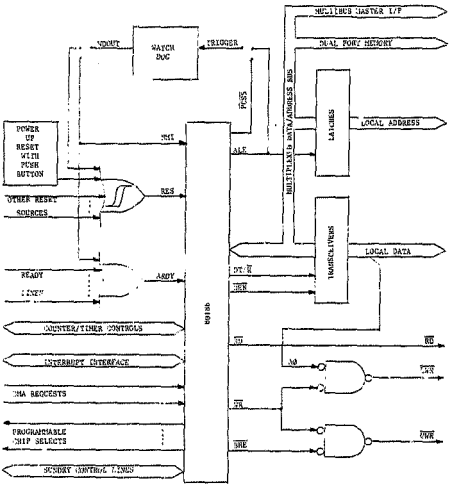


FIGURE C.1 THE LOCAL PROCESSOR AND ASSOCIATED CIRCUITRY

Elements of the surrounding circuitry are:

C.1.1 Address Latches and Data Bus Transceivers

These serve the dual purpose of demultiplexing and buffering the multiplexed data/address bus to form the local data and address busses. These busses serve the local memory, the 1553B bus interface, and the SBX interface.

Only the lower 16 address lines are latched since all decoding of the top 4 lines is done by the programmable chip select logic. The data transceivers are enabled by the chip select signal of any device which is on the local busses.

C.1.2 Reset Circuit

This consists of an RC power-up reset circuit having a time constant of 220 milliseconds with a reset switch. This and other lines capable of resetting the processor from the 1553B bus interface, multibus interface, and watchdog are combined, Schmitt triggered and connected to the processor RES* input.

C.1.3 Watchdog

The output of a retriggerable one shot is used to either cause a non-maskable interrupt (and a ready in case the processor has "hung" in an access to non-existant memory), or reset the processor (jumper selectable option) if it is not triggered at least once every 100 milliseconds. The one shot output is used directly in the case of the ed.

triggered non-maskable interrupt, while a shift register is used to produce a pulse off the output for the reset option.

The one shot is triggered either by the processor's Address Latch Enable line, for a hardware watchdog, or by one of the programmable chip select lines (PCS?*) if a software watchdog is required. (Also jumper selectable).

C.1.4 Odd/Even Byte Selection

Pairs of byte wide memories and peripherals are arranged with one device on the upper and one on the lower data bus. The address inputs use address lines A1 upwards, and both share the same chip select line. The device on the lower data bus thus occupies a group of consecutive odd byte addresses, and the other occupies the corresponding consecutive even addresses.

The processor is able to access either a byte or a word at a time. Whether a transfer is to word, odd byte, or even byte is indicated by the combination of the A0 address line and the BHE* line. These two lines are decoded and combined with the WR* line to produce separate write lines LWR* and UWR* for the lower and upper data busses respectively. The same RD* line, however, goes to all devices. Distinction between odd and even bytes is thus made only on write transfers.

C.1.5 Ready Input

The Ready lines of all devices in the local processor system which need to be able to vary the number of wait states that are inserted in an access to them, are ORed

together, and the resulting line connected to the Asynchronous Ready input (ARDY).

C.2 LOCAL MEMORY

C.2.1 ROM

The local ROM is selected by the Upper Memory Chip select (UCS*) line. This places it at the very top of the one Mbyte address range which includes the restart location. Several EPROMs types can be used. They are the 2732A (4k), 2764 (8k), 27128 (16k), and 27256 (32k). Devices with access time of 250 nanoseconds or faster may be used without the need to insert wait states. The pair of sockets is jumper configured for the type to be used (see figure in appendix D).

The ROM requires a Ready input to the processor. The UCS* line is simply used for this purpose thus giving no wait states. This is done because, after a reset, when the processor first begins fetching instructions, no automatic ready generation is in force on the UCS* line. This only becomes effective after it is specifically programmed into the control register for this chip select line.

C.2.2 RAM

The local RAM is selected by the Lower Memory Chip Select (LCS*) line. It is therefore always positioned at the base of memory. The devices that may be used in these sockets are the 6116 (2k), and 6264 (8k) static RAMs. The sockets must be configured for the correct device by installing jumpers (see figure in appendix D).

The local RAM does not need to provide a Ready line.

C.3 MIL-STD-1553B BUS INTERFACE

The dual redundant MIL-STD-1553B bus interface is shown in figure C.2(b) which folds out at the end of section C.3. It can be broken down into the following sub-blocks:

- The MRTU 53045.
- Data bus buffer/latches and enable/strobe logic.
- Data transfer handshaking and DMA request.
- Interrupt lines.
- Processor reset.
- Status lines.
- Command lines.
- Programmable peripheral interfaces.
- Busy latch and decoding.
- RT address switches.
- Analog section and isolation transformers.

Due to the large amount of combinational logic in this block, portions of it were implemented in PALs to save space (see appendix E).

Every line in the bus interface has been checked for current loading of the output by the inputs connected to it.

Figure C.2(a) indicates which areas of the diagram in figure C.2(b) correspond to these sub-blocks.

C.3.1 The MRTU 53045

This is the main component in the interface. Its

requirements dictate the form that the rest of the interface takes. Appendix B summarises the interface to and operation of this device.

C.3.2 Data Bus Buffer/Latches and Enable/Strobe Logic

The 16 bit 186 processor data bus is isolated from, and interfaced to, the 8 bit MRTU 53045 data highway by means of four edge-triggered, byte wide, tri-state latches; two for received data, and two for data to be transmitted. These also provide a one word buffer in each direction.

These four latches are mapped to a single word location in the local processor I/O address space, that can be both written to and read from (though the same data will not be read back). The location is enabled by the line MRTUSEL*.

This arrangement has the disadvantage that all types of words that are passed between the local processor and bus interface must be passed via these latches. This differs from the application example in reference 20 pp 31 which suggests that a separate pair of latches should be used for each different type of word that is passed, which is logical, since each type will be handled in a different way by the processor, and the transfer of each type is controlled by different lines on the MRTU 53045. That method is however very uneconomical on board space, particularly in this case where BC and BM modes are required and command and status word types must also be considered. The single location method employed in this design is very much more compact, but must pay the price of a higher processor software overhead, and more complicated decoding of the MRTU output lines.

C.3.2.1 Transmit Latch

The two transmit (TX) latches take their data inputs from the 16 bit local processor data bus, one from the high byte and one from the low byte. The data outputs of both latches are connected to the same 8 bit MRTU data highway.

The latches are always written together, as a word, by the local processor using a strobe derived from the WR* and MRTUSEL* lines.

The latches are read individually by the MRTU 53045, times when a word is required, by using two separa.. output enable lines; Low, and High Byte Enable (LBEN* and HBEN*). These lines are derived from various MRTU outputs, and enable the latches according to the truth table in appendix E. They are implemented in a PAL.

C.3.2.2 Receive Latch

The two receive (RX) latches are connected in exactly the opposite sense to the transmit latches: The inputs to the MRTU data highway, the outputs to the upper and lower bytes of the 186 data bus.

The latches are read together by the processor. The output enables are both driven by a line derived from the MRTUSEL* and processor RD* lines.

The latches are written individually using two strobes; Low and High Byte Strobe (LBSTB* and HBSTB*). These strobes clock the latches according to the truth table in appendix E. They are also implemented as PAL circuits.

C.3.3 Data Transfer Handshaking and DMA Requests

This section of the circuit is concerned with satisfying the requirements of the MRTU asynchronous data transfer handshake, and the provision of a line that can be used as a DMA request. As by-products, it also provides lines that indicate the status of the transmit and receive latches, and could be used as interrupts.

The status of each latch, that is, if data has been written to it but not yet read on the other side, is indicated [illegible] outputs of positive edge sensitive RS flip-flop[illegible]ad to be synthesised).

These outputs enable a simple circuit (see figure in appendix E) to be designed and implemented in a PAL, which returns a Data Transfer Acknowledge only under the *following conditions:*

- If there is new data not previously read in the TX latch, in the case of a request for data to be transmitted, or
- if no word will be over-written in the RX latch, in the case of a request to pass received data to the subsystem.

A problem with this method arises when a word in the TX latch becomes invalid due to a change in circumstances. An example of this is the case of vector word handling as follows: because the MRTU 53045 (in RT mode) expects a vector word to be present in the TX latch for immediate access if a mode command to transmit the vector word is received, the processor must ensure that this word is present in the latches between command servicing. If a transmit command then occurs, the MRTU will take the vector word as the first word to be transmitted.

To avoid this problem, a further D type flip-flop has to be used, the output of which is only allowed to go true during a TX Data Transfer Request. The output of this flip-flop is used as the latch status for the purposes of providing a DTAK*, while the original RS flip-flop is still retained to indicate the true TX latch status.

A similar problem occurs with the RX latch, but it is easily solved under software, simply by the processor reading and discarding the value in the latch.

A DMA request is simply generated any time there is a valid word in the RX latch or there is a TX Data Transfer Request. A DMA transfer will obviously only take place if the channel is enabled.

C.3.4 Interrupt lines

There are a large number of signals which may be required to cause interrupts in possible applications. Most signals are in the correct form to be used as interrupts directly. An exception is the SYNC* line, this is validated by a simple circuit to give VALIDSYNC (for use in RT mode), as suggested in reference 18 pp. 43, part of which is implemented in a PAL (see appendix E).

Because of the large number of lines, interrupts have been combined, where possible so that one line can be used for two interrupts.

For example, one interrupt may occur, or have meaning only in RT mode, while another only occurs or has meaning in BC mode. Both can be qualified by the hybrid being in the correct mode and then ORed onto one line. Lines that have

been combined thus are INCMD* in RT mode with RTO* in BC mode, and DBCREQ* in RT mode with RXINT* (derived from the RX latch status line above) in BC mode.

All other interrupt lines may be used in all modes if required.

C.3.5 Processor Reset

The MRTU's RESET* line pulse low for approximately 500 nanoseconds on receipt of a valid Reset mode command. This unfortunately is not long enough to guarantee the processor resetting properly. The pulse is therefore stretched using a shift register.

C.3.6 Status Lines

Apart from the usual MRTU status lines, that is outputs whose level indicates a state of the hybrid, two extra status lines have been synthesised from lines which carry pulses (which indicate events), using simple RS flip-flops. They are:

LASTRXOK*. This is set by GBR* and reset by NBGT*. It indicates that the last message transmission received passed all validation tests.

LASTCW/DW*. In BC or RM modes this indicates whether the word in the RX latch is a command or data word.

C.3.7 Control lines

Most of the control input lines on the MRTU 53045 require to be driven to a desired level at all times. Three of these also drive LEDs to indicate in what mode the terminal is operating in (RT/BC* and PASMON*), and if it will accept bus control in the event of it being offered (DBCACC*).

There are three cases where some extra circuitry for a control line is required.

C.3.7.1 BCOPSTB* input

The BCOPSTB* input is not a true strobe input. Control over this input, which is normally inactive, is required in two ways [19]:

For bus control options zero, one, and two, the input must be set active until the MRTU replies with NDRQ* active. BCOPSTB* must not go active while NDRQ* is asserted. This is achieved by using a JK flip-flop which asserts BCOPSTB* when clocked (by simply writing to a location in the 186 I/O address space with dummy data), and is cleared by NDRQ* active.

It must be possible to set the input active for a period of time, in order to use bus control option three. Two methods have been designed to do this as it is not obvious from the documentation what the NDRQ* line will be doing in option three. One method is to provide another location in the 186 address space which clears the flip-flop when written to. The other method is to drive it active as with any other control input.

C.3.7.2 BUSREQ Lines

The BUSREQA and BUSREQB lines are the only lines on the hybrid, besides the data highway, that are bidirectional. They are inputs in BC mode and outputs in RT mode. They are driven, in BC mode, by open collector gates which are floated high in RT mode. As a further precaution (as it is not known if there is a delay between changing the hybrid to BC mode and these lines being floated by the hybrid, or if they become outputs when the PASMON* line is asserted), another control line, ENREQ, is provided which must be specifically asserted in order to control the open collector gates.

The circuit which drives the open collector gates (see diagram in appendix E) has been "PALed".

C.3.7.3 SSERR* Line

This line is set active either by a watchdog timeout, or manually by a control line.

C.3.8 Programmable Peripheral Interfaces

Two 8255A-5 PPIs are used to read status lines, and to write to and latch control lines on the MRTU 53045. They are arranged together as a single word wide device in the processor's I/O address space, but are individually addressable, and are enabled by the PCS0* line.

Connecting these to the 186 processor poses one problem in that the data hold time after a write, required by the 8255A-5, is not guaranteed by the 186. This was solved by

adding two wait states to accesses on PCS0* with the programmable wait state generator, and using a shift register delay and the WR* line to produce a Shortened Write line (SWR*) that gives the necessary hold time.

All three 8 bit output ports on each device are used in mode 0. Ports A and B are input, while C is output. Considering the two devices together, these may be regarded as 16 bit ports. The output bits in port C may be read back to the processor, and also individually accessed using a bit set/reset function.

The grouping and arrangement of the lines on the input and output ports has been done with a view to reducing the number of port accesses and amount of manipulation that has to be done in software as follows:

Port A inputs all the lines at once that need to be read in order to set up the DMA controller after receipt of a transmit or receive command in RT mode. The lower byte contains the combination of the TX/RX* and SA lines, starting from bit one, which will be used as the address in the dual port memory buffer pointer table (INCMD*, on bit zero, will always read low during command servicing). The upper byte reads in the number of words to be transferred in the message.

Port B inputs all the remaining status lines, the ones that are expected to be most used are on the lower byte.

Port C latches and drives all control lines. The ones that are expected to be frequently altered are placed on the lower byte, while those that will simply be set up and left are on the upper byte.

C.3.9 Busy Latch and Decoding

When a non-mode command is received by the terminal in RT mode, the terminal must indicate whether or not the particular subaddress referred to in the command is able to process the command, by means of setting the BUSY* line to the appropriate value. Only 500 nanoseconds are allowed to decode the sub-address and set the busy line. This operation could, therefore, be done in hardware as follows:

The subaddress is used to select or multiplex a particular line from a 30 bit latch onto the BUSY* input. The latch is written to by the processor at regular intervals, and contains the busy settings of the individual subaddresses.

The actual scheme employed uses only a sixteen bit latch and multiplexer to save space. In most cases, provision for sixteen subaddresses will be sufficient. In applications where more are required, some bits in the latch will have to represent the logical OR of the busy settings of two subaddresses.

C.3.10 RT Address Switches

Six DIP switches are provided to set up the address to be used by the terminal when operating in remote terminal mode. Five of the switches set the address itself, while the last is set for odd parity on the address.

If the address is set such that there is a parity error, an LED will be lit by the RTADPAR line.

C.3.11 Analog Section and Isolation Transformers

This section was not designed, but simply connected as indicated in the data sheet for the MCT 3231 driver/receiver, with the exception that transmitter inhibit decoding need not be provided as this is done by the hybrid. No external threshold setting resistors are used on the receivers.

C.3.12 Bus Interface Chip Selects

The bus interface is selected by the processor by using two Peripheral Chip Select lines.

PCS0* selects the two PPIs as detailed above.

PCS4* enables the decoding of address lines AD2, AD3 and AD4 to give 8 select lines, only 4 of which are used (the others were provided in case they were needed in the prototype). The four are:

- MRTUSEL* which enables the TX and RX latches.
- BUSYSEL* which enables the busy latch.
- BCOPPULSESEL* which enables the strobe to set the BCOPSTB* flip-flop.
- BCOPRSTSEL* which enables the clearing of this flip-f...

They are each qualified by either RD* or WR* to produce the desired signal.

C.3.13 Bus Interface Software

The methods devised, during the above bus interface

hardware design, for driving the bus interface may now be discussed.

C.3.13.1 Remote Terminal Mode

In RT mode, the bus interface is not so much driven as serviced. The scheme described here is approximately what is expected to be used in the RT routine and associated interrupt servers, described in the software design overview.

The scheme is composed of interrupt service routines and a main routine which runs continuously while the terminal is in RT mode.

Interrupt service routines. There are four interrupts in RT mode that, if used, must be the highest priority in the interrupt structure and must never be disabled. They will run in any context, and will not interfere with the multitasking executive if they occur while it is rescheduling tasks. They are:

- VALIDSYNC interrupt. This is required only if the terminal is required to support the two synchronise mode commands. If present it must be the highest priority interrupt on the terminal in order to obtain accurate synchronisation. It indicates the reception of a valid Synchronise Mode Command. The routine should perform the following:

  Read WC4 line via PPI to determine whether or not there is an associated data word.

  If not, perform synchronisation and return from interrupt.

  If there is an associated data word, poll

RXWORDWAITING line until data word is received, read it from the RX latch, use it to perform synchronisation, and return from interrupt.

The exact mechanism of performing synchronisation is not specified.

- DBCREQ interrupt. This occurs in response to a dynamic bus control mode command and is required only if this mode command must be supported by the terminal. It should also be highest priority, but because they will never occur simultaneously, it may be made one level lower priority than VALIDSYNC. The routine simply sets a flag (DBCflag) and returns from interrupt.

- INCMD interrupt. This occurs whenever any command is received by the terminal. It is required in all but the simplest RT application (where polling can be used). It is one lev of priority below the previous two cases, and must keep the higher priority interrupts enabled since they will always occur during this service routine. It must perform the following:

  PPI port A must be read.
  From this the value of the SA lines must be determined.
  If this value indicates a mode command then return from interrupt.
  If not, program the DMA channel as described in sections 4.2.3 and C.3.8.
  Enable the DMA channel and return from interrupt.

- End of DMA interrupt. The DMA channel can be made to cause an interrupt after the last

transfer in a message. This service routine must:

Test the LASTRXOK* line to determine if the message contained an error.
If it did, take appropriate action, (such as setting a flag or altering the queue pointers), and return from interrupt.

A particular application may require the use of some other interrupts, to indicate the occurrence of an error for instance.

It should be noted that interrupt service latency affects the processing time it takes to begin an interrupt service routine. Interrupts are only serviced between instructions. If an instruction such as a multiply or divide is being executed, the interrupt may not be serviced for several microseconds.

The RT routine. On entering RT mode the routine must perform the following initialization:

- The DBCACC*, BCSTEN0, and BCSTEN1 lines must be set to the desired values for the duration of operation in RT mode.
- The vector word and busy latch must set be to initial values, as must the SSERR* and SERVEREQ* lines.
- The interrupt controller must be set up to enable the above interrupts and give them the correct priority. Since two of the above interrupt lines are used for different interrupts in BC mode, the pointers in the table at the base of the processor memory must be set to point to the correct routines.

- The hybrid must be placed in RT mode by setting the RT/BC* line high.

After initialisation, if the hybrid's DBCACC* line is set active, the routine must repeat the following operations until the DBCflag is set, at which time it must pass control to the BC routine. If the DBCACC* line was not set active during initialisation, these steps must simply be repeated indefinitely. The steps are:

- Test the error flags of each subaddress and place the logical OR on the SSERR* line.
- Test the service request flags of each subaddress and place the logical OR on the SERVREQ* line.
- Place the vector word of the highest priority subaddress requesting service in the RX latches, in case a transmit vector word mode command should be received (as described in section C.3.3).
- Test the busy flags of each subaddress, combine to form a word and write this to the busy latch.

Only the steps which support functions which are required in a particular application need be included.

C.3.13.2 Bus Controller Mode

No definite scheme is given as the standard requires no specific functions of a terminal in BC mode.

The interrupts that will probably be required in this mode (among others) are:

- RXINT, which indicates that a word has been placed in the RX latches by the hybrid. The LASTCW/DW* line can be read to determine what type of sync the word had.
- RTOINT, which indicates that the time for a reply to be received from a remote terminal has been exceeded (approximately 14,5 microseconds after transmission of the command ended). If it is desired to streach the no-reponse timeout, this interrupt may, instead of taking some error recovery action, simply start a timer. This timer will run for a prescribed time, and then will itself cause an interrupt on which the error recovery action can be taken.

These two interrupts occur on the same lines as, and replace, the INCMD and DBCREQ interrupts, which are not available in BC mode. Care must be taken, as mentioned above for RT mode, to point the interrupts to the correct routines.

On entering the BC routine, and after suitable initialization, the hybrid must be changed to BC mode by setting the RT/BC* input low, and the BUSREQ lines must be enabled by setting the ENRQ line high.

On leaving BC mode these lines must be toggled, but in the reverse order.

Within the BC routine, commands are initiated as described in appendix B, by setting up the appropriate lines and strobing the chip set by writing to the location in the I/O space, as described in section C.3.7.1. The DMA channel obviously may be used to achieve data transfers as in RT mode.

C.3.13.3 Bus Monitor Mode

Again no specific scheme is presented. It is expected that in bus monitor mode, a fair amount of information about what is happening on the bus will be collected by means of interrupt service routines which increment counters on the occurrence of certain events. In order to aid the counting of events, two of the 186 internal counter/timers can be used. If the terminal is required to monitor the bus for activity, and take over control of the 1553B bus in the event that none is detected for longer than a given time, then one of these two timers can be used to measure the time since the last activity on the bus.

The hybrid is placed in BC mode by setting the RT/BC* line low, initiating an option three BC command, then setting the PASMON* input active (see appendix B).

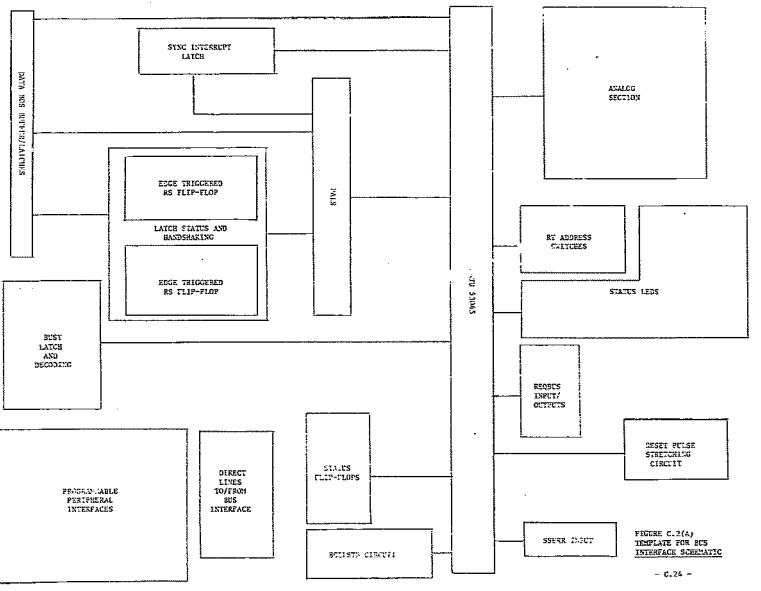
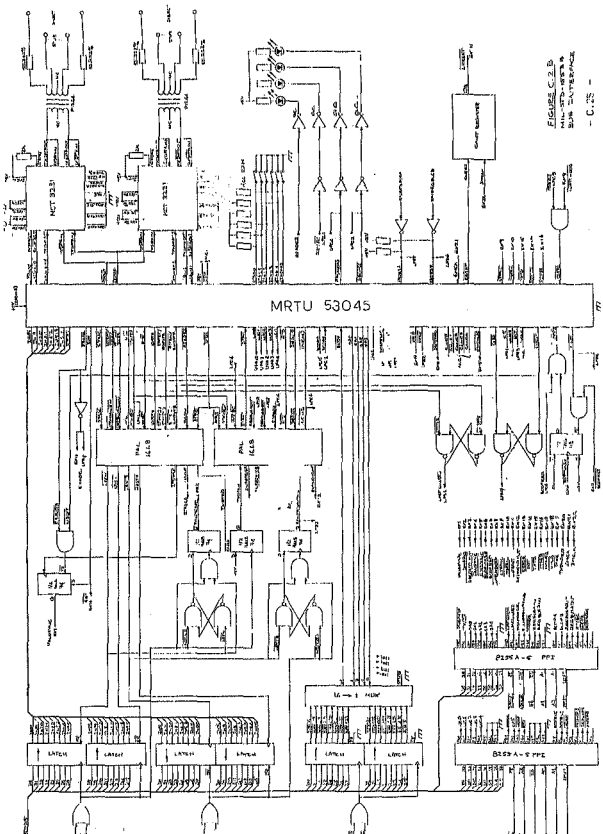


FIGURE C.2(A)
TEMPLATE FOR BUS
INTERFACE SCHEMATIC


MRTU 53045

C.4 GENERALISED I/O INTERFACE

The generalised I/O interface is required to connect the processor bus to the SBX I/O bus [24]. Figure D.3(A) in appendix D is the detailed diagram of the interface.

Most lines in the interface are straight-forward connections from one bus to the other. There are certain lines which require some explanation.

C.4.1 IOWRT* Line

As with the 8255A-5 PPIs in the 15F3B bus interface, there is a violation of the SBX data hold time after write by the normal 186 cycle. The Shortened Write line (SWR) devised for the PPIs is used again here to drive the IOWRT* line. All access to the SBX interface must thus have a minimum of two added wait states.

C.4.2 Chip Select Lines

The whole interface is enabled by the PCS2* line from which three sub chip selects must be provided:

- The MDACK* line which is used during DMA transfers.
- The MCS0* and MCS1* lines which enable either 8 word, or 8 byte locations each. This is done according to two schemes described in the SBX standard. This design allows for both by using jumpers (see diagram in appendix D).

Figure C.4 shows the two possible address maps that may be obtained for the interface depending on which scheme is

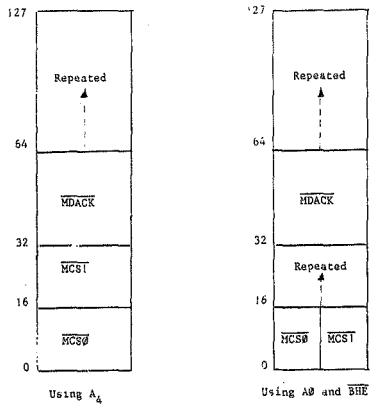


FIGURE C.3 SBX ADDRESS MAPS

C.4.3 TDMA Line

This is used to signify the end of a DMA transfer. It may be either an input or an output depending on the I/O module that is being used. Jumper selection is employed to select connection to either:

- a programmable chip select line, PCS3*, if the line must be an input, or
- an interrupt line if it must be an output.

C.4.4 RESET Line

This cannot simply be directly connected to the RESET output of the 186 processor as, although the power up reset of 50 milliseconds is sufficient for SBX, the in-operation reset that may originate from the 1553B bus interface is less than one microsecond which is too short for SBX. A one shot, triggered by the 186 RESET line, is thus used to provide the required 50 microsecond SBX reset pulse.

C.4.5 SBXREADY* line

Since the number of wait states required by an access to the SBX interface (that is required by the I/O module connected to it) may be more than three, the programmable wait state logic may not be usable. The SBX MWAIT* line qualified by the chip select is thus connected to the processors ready logic. The programmable wait state logic must still be used to ensure that at least two wait states are added as specified in C.4.1.

C.5 MULTIBUS INTERFACE

The design of both the master and slave interfaces were aided by the use of the SBC 86/12A board as an example [21,22].

C.5.1 Slave Interface

This is so closely coupled to the dual port memory arbitration logic and buffering that it is covered under

that section.

C.5.2 Master Interface

The Multibus master interface in figure C.5 is of a fairly standard design. It is based on the application example in the 186 data sheet [39].

The arbitration lines are interfaced to by an 8289 bus arbiter, while the command lines are driven by an 8288 bus controller. A Multibus transaction is requested whenever a bus cycle occurs that does not cause one of the Programmable Chip Select lines to go active.

The processor multiplexed data and address lines are isolated from the corresponding Multibus lines via transceivers and latches respectively. These are controlled by the bus controller and arbiter. However, many components could be saved if these lines could be combined with those on the slave interface via the dual port memory. This is usual on Multibus single board computers. In this design this has been done with the data bus, and is covered along with the slave interface under the dual port memory in the next section.

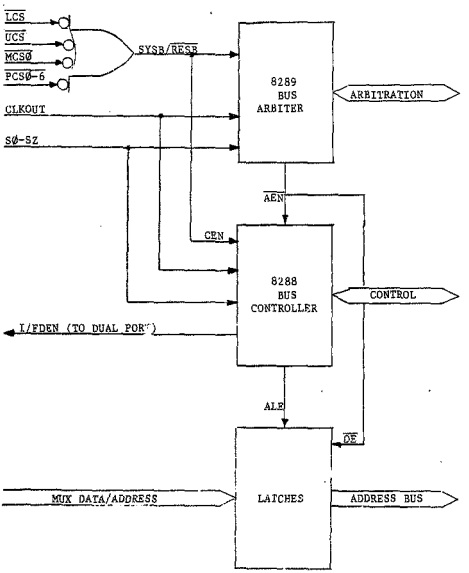


FIGURE C.4 MULTIBUS MASTER INTERFACE

C.6 DUAL PORT MEMORY

Figure C.6 is a diagram of the dual port memory. It can be treated in the following parts:

- The RAM itself.
- The Interrupt flip-flops.
- Dual port address decoding.
- Bus buffering.
- Multibus address decoding.
- Arbitration and control logic.

Next to the 1553B bus interface, most design effort was put into this block, particularly into the arbitration and control logic.

C.6.1 RAM

Up to three pairs of sockets, connected to the dual port busses, are provided for static RAMs. Each pair is identical to the design of the local RAM. Each pair may take either 6116s (2k) or 6264s (8k) as selected by jumpers, according to the diagram in appendix D.

The maximum size of the memory is thus 48k. The amount required by the design overview, namely 32k, can be achieved using only two pairs.

C.6.2 Interrupt Flip-flops

Two lines, to be used to generate interrupts on the Multibus INT* lines, are driven by flip-flops. They may be set and reset by accessing four consecutive word locations in the dual port address space.

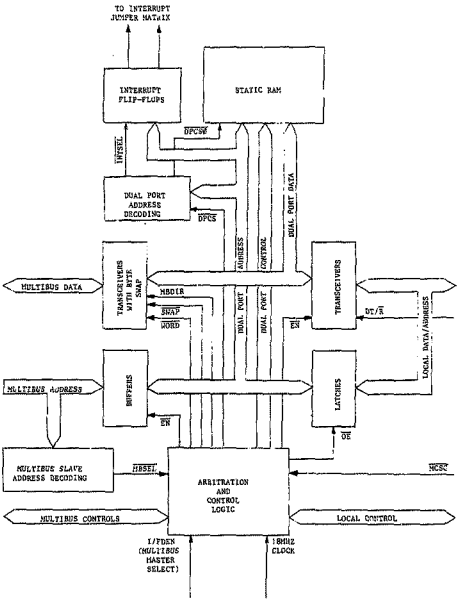


FIGURE C.5 DUAL PORT MEMORY

C.6.3 Dual Port Address Decoding

The dual port address space is decoded into four 16k blocks each with an enable line. One pair of RAMs is always connected to the lowest block, while two optional RAM pairs and the interrupt flip-flops may be connected on any of the other lines by means of jumpers.

The decoding logic is itself enabled by DPCS* whenever a dual port access occurs.

C.6.4 Bus Buffering

At the local port, the 186 multiplexed data address bus is demultiplexed by using transceivers and latches, in very much the same manner as is used to produce the local data and address busses. They are enabled onto the corresponding dual port during local accesses.

At the Multibus port, the address bus is buffered and inverted. The data bus is passed through three transceivers which implement the "byte swap" function that is required when Multibus is interfaced to a 16 bit device. These transceivers also invert the data lines. Both buffers and transceivers are enabled during a Multibus access to the dual port.

C.6.5 Multibus Address Decoding

This circuit decodes the Multibus address lines ADR7* through ADR13*, to produce a select line to the dual port memory (MBSEL*). This is done according to jumpers which determine the base address, and size, of the dual port memory in the Multibus memory address space (see figure

and table in appendix D). The top four address lines are not decoded.

C.6.6 Arbitration and Control Logic

A single component dual port memory controller is available for use with dynamic RAMs, but due to the requirement that only static RAM be used, this design could not take advantage of it to save space. However, even if it had been possible to use this device, it would almost certainly have been necessary to include some extra circuitry to suit the particular application.

A controller therefore had to be designed, which did have the advantage that it could be customised to the application. This controller, with the exception of two external latches, is entirely implemented in two 20 pin PALs making it very compact indeed. The PAL implementation is detailed in appendix E.

The design of this section went through several versions, to try and make it more compact, and efficient. The final one presented here was arrived at after a race around condition was found in the prototype terminal (see next chapter). The design was done by considering the logic itself, rather than using the more mechanical Asynchronous State Machine (ASM) techniques which would probably have arrived at a more correct design, but at the expense of complexity.

The logic is shown in appendix E. It is best described in its two parts.

C.6.6.1 Arbitration

The arbitration logic has as inputs the select and lock lines from each port. In the case of the Multibus select MBSEL*, qualification by one of the two memory operation command lines is necessary. In the case of the local port, the MCS0* line has to be delayed until the rising edge of the processor clock occurs, otherwise consecutive access to the memory may not be resolved by the logic which is clocked at 18 MHz.

The internal request lines LOCREQ and MBREQ are asserted if, either there is a select from a particular port, or the port asserts its lock line while the LOC/MB* line indicates that that port has the memory. In this way a port may retain the memory between access without maintaining a select.

By considering these internal request lines and the current internal state, the dual port memory is assigned to one or the other port. This is done such that the local port always has priority in the event of both ports requesting access simultaneously. If both ports request consecutive access, and neither asserts its lock line, alternate accesses will be granted. When neither port is requesting access, the memory will be assigned to the local port, so that no delay is experienced by the 186 processor if it does require access. This scheme is summarised by the truth table in appendix E and leads to a Boolean equation which describes the logic.

The result of the arbitration is indicated on two outputs to the control logic, one which signifies which port has the memory (LOC/MB*), and another which disables the subsequent logic while a change over is taking place (CYCEN).

C.6.6.2 Control

The control logic decodes these two outputs into LOCALCYCLE and MBCYCLE. These are used to effectively couple the controls of the RAMs to those of the appropriate port, enable the correct buffers, and send a ready signal back to the requesting processor.

After careful analysis of the timing requirements of both ports and the RAMs, it was found unnecessary to provide any relative timing between the various signals. This is mainly thanks to the speed of the RAMs.

Finally, this part of the circuit is also responsible for connecting the local data bus to the Multibus data bus during access by the local processor to Multibus via the master interface. Such an access is indicated by the line IFDEN* generated by the 8288 bus controller. All other controls for the buffers (byte swap controls etc.), are picked up from the respective port controls. Note that no arbitration for the dual port bus is required for this situation, since both ports are known to be occupied with transactions not involving the dual port memory. For this reason, the local processor is not allowed to try and access the dual port memory via Multibus.

C.7 LOCAL PERIPHERALS

C.7.1 186 On board Peripherals

As mentioned previously, nearly all the local peripheral requirements are fulfilled by devices integrated into the 186 processor itself. They are allocated as follows:

- DMA channel 0 serves the *MIL-STD-1553B* bus interface.
- DMA channel 1 serves the SBX interface.
- Timers 0 and 1 are available for counting events on the 1553B busses, or timing bus idle conditions when the terminal is in BM mode.
- Timer 2, to which there is no external access, can be used to provide a tick for a multitasking executive.
- The programmable interrupt controller is operated in non-RMX mode. This provides four external interrupt lines, which is insufficient. Two of these lines are thus operated in cascade mode, so that an external interrupt controller can be used as detailed below.

C.7.2 Programmable Interrupt Controller

An external programmable interrupt controller (PIC) is used to provide an extra eight interrupt lines. It is connected to the lower data bus and is enabled by PCS1*.

A point to note here is that the Interrupt Acknowledge line, INTA0* must be directed back to the processor ready logic since it is not catered for by the programmable wait state generators. This line must be pulled up since it is not an output after reset, but must be programmed to be an output during the 186 internal PIC initialisation routine.

C.7.3 Interrupt Jumper Matrix

This is not a peripheral as such, but it is best covered with the PICs. Figure C.8 summarises the lines into and

out of the matrix.

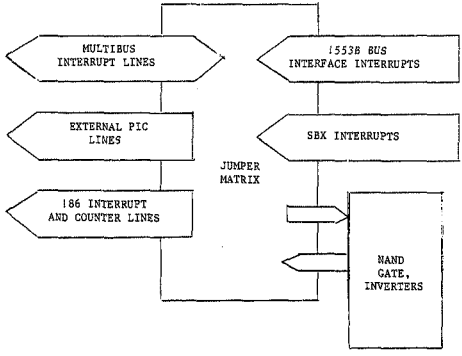


FIGURE C.6 INTERRUPT JUMPER MATRIX

The lines arriving at the matrix have been conditioned to be in the correct sense to cause an interrupt. Lines which carry pulses will have to be sensed by the PIC in edge triggered mode. In order to do this they must be active low. Lines on which a level must cause an interrupt must be active high and can be sensed by the PPI in either edge or level triggered mode as desired.

Three inverters and a three input NAND gate are available to combine interrupts from the 1553B bus interface. It may be convenient, for instance, to combine the *HSFAIL**, LTFAIL*, and TXTO* interrupts, to give one interrupt that will indicate a catastrophic failure of the bus interface hardware.

The Multibus interrupt lines can each be either an input to, or output from, the matrix. Only level triggered interrupts should use these lines.

C.7.4 Clock Generation

A 16 MHz crystal is provided for the 186 internal clock generator. The processor in turn provides its 8 MHz clock on the CLKOUT pin. The other frequencies that are required by certain components in the terminal are supplied by an 8284A clock generator with some extra circuitry.

Using an 18 MHz crystal, this device produces the following clock signals:

- 18 MHz for the dual port arbitration logic.
- 9 MHz, for Multibus master and SBX interfaces, produced by halving the 18 MHz signal with a flip-flop.
- 6 MHz, for the MRTU 53045 bus interface hybrid.
- 3 MHz, used for clocking sundry shift registers (and by the USART in the prototype).

C.8 PROCESSOR ADDRESS SPACE ALLOCATION

Because of the flexibility of the positioning of the Programmable Chip Select lines, no definite map of the

memory or I/O address spaces can be given. Table C.1 summarises the allocation of the chip select lines and the programming of their associated programmable wait state generators.

TABLE C.1 PROGRAMMABLE CHIP SELECTS AND WAIT STATES

LINE	DEVICE(S) ENABLED		WAITS	READY
UCS*	Local ROM	(top of memory)	0	Ignored
LCS*	Local RAM	(base of memory)	0	Ignored
MCS0*	Dual port memory			
MCS1*	-	(mid-range memory)	0	Used
MCS2*	-			
MCS3*	-			
PCS0*	Bus interface PPIs			
PCS1*	External PIC	(I/O or memory)	2	Used
PCS2*	SBX Interface chip selects			
PCS3*	SBX TDMA* line			
PCS4*	Bus interface RX/TX, busy, strobes			
PCS5*	Watchdog refresh	(I/O or memory)	0	Ignored
PCS6*	-			

The ready grouping PCS0*-PCS3* uses the ready input for the benefit of the SBX interface chip selects. The other three chip select lines must simply be routed back to the processor ready logic to provide an immediate ready.

The 186 internal control block must be placed somewhere in either memory or I/O space.

All address space that is not in the range of one of the Programmable Chip Select lines, maps to the Multibus master interface. Note that a problem may arise with Multibus being requested when an access to the internal control block is made. Although the processor ignores all usual inputs during such an access, the outputs indicate a normal bus cycle.

APPENDIX D

HARDWARE SCHEMATICS AND CONFIGURATION TABLES

D.1 PIN-TO-PIN SCHEMATICS

The following schematics show the full pin-to-pin connections of the terminal hardware design as implemented in the prototypes

Figure D.3(A) shows the SBX interface as it would have been implemented had this been possible, while figure D.3(B) shows the standard serial interface design that was used instead.

The convention adopted for indicating lines between figures is as follows:

The arrow on a line between figures indicates the direction of the signal. The letter(s) within the arrow specify a corresponding arrow on the figure indicated by the number after the slash. The combination of arrow direction, figure number and letter(s) specify a unique connection. For example: B/12> on figure D.1 would indicate connection to the line marked <B/1 on figure D.12.

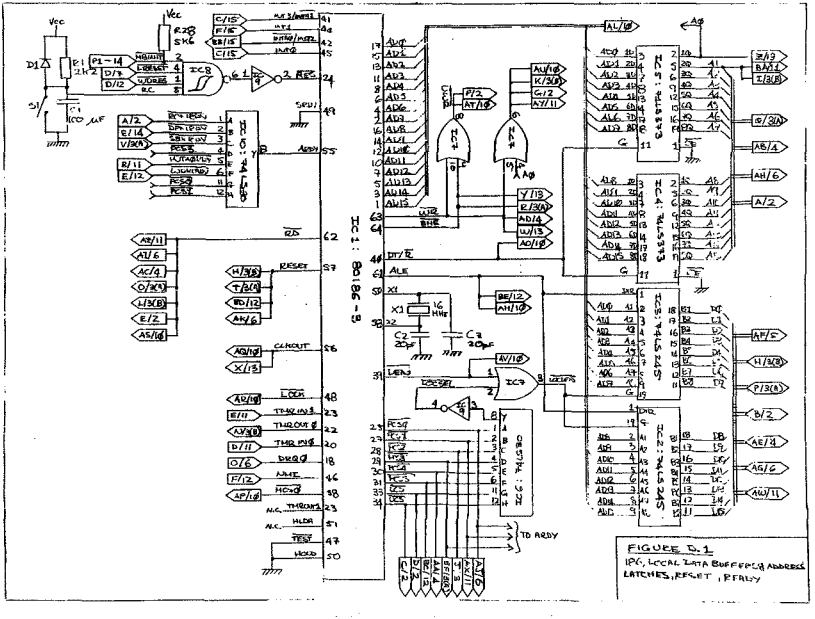
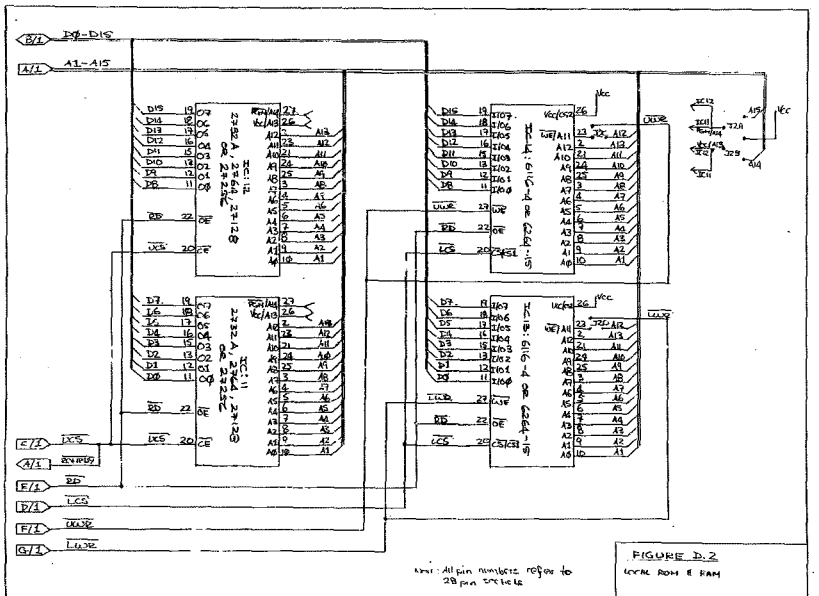


FIGURE D.1
IPG, LOCAL DATA BUFFERS & ADDRESS LATCHES, RESET, READY



Note: All pin numbers refer to 28 pin devices

FIGURE D.2
LOCAL ROM & RAM

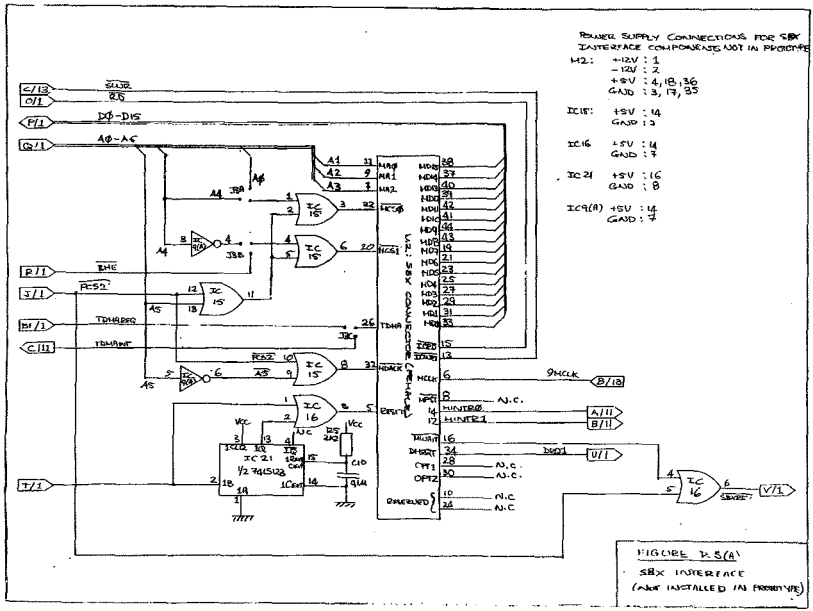

POWER SUPPLY CONNECTIONS FOR SBX INTERFACE COMPONENTS NOT IN PROTOTYPE
M2: +12V : 1
-12V : 2
+5V : 4, 18, 36
GND : 3, 17, 35
IC15: +5V : 14
GND : 7
IC16 +5V : 14
GND : 7
IC21 +5V : 16
GND : 8
IC9(A) +5V : 14
GND : 7
C/13
O/1
P/1
Q/1
R/1
J/1
M/1
C/11
T/1
SWR
RS
DØ-D15
AØ-A5
BHE
PCS2
TDMARQ
TDMAINT
IC 15
IC 16
IC 21
1/2 74LS123
MA0
MA1
MA2
MCS0
MCS1
TDMA
HDACK
RESET
MCLK
MINTR0
MINTR1
DMQ1
OPT1
OPT2
RESERVED
9MCLK
B/13
A/11
B/11
U/1
V/1
N.C.
SBX INTERFACE
(NOT INSTALLED IN PROTOTYPE)

FIGURE D.5(A)

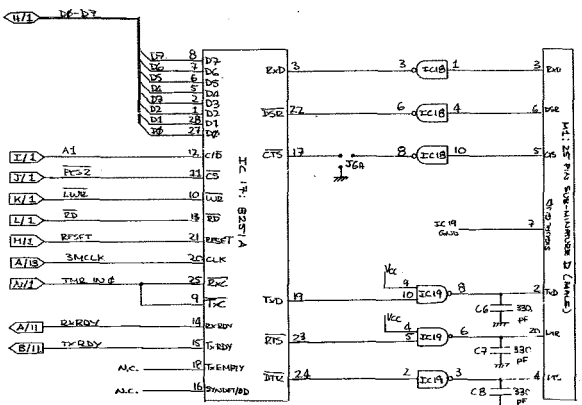


FIGURE D.3(B)

SERIAL INTERFACE

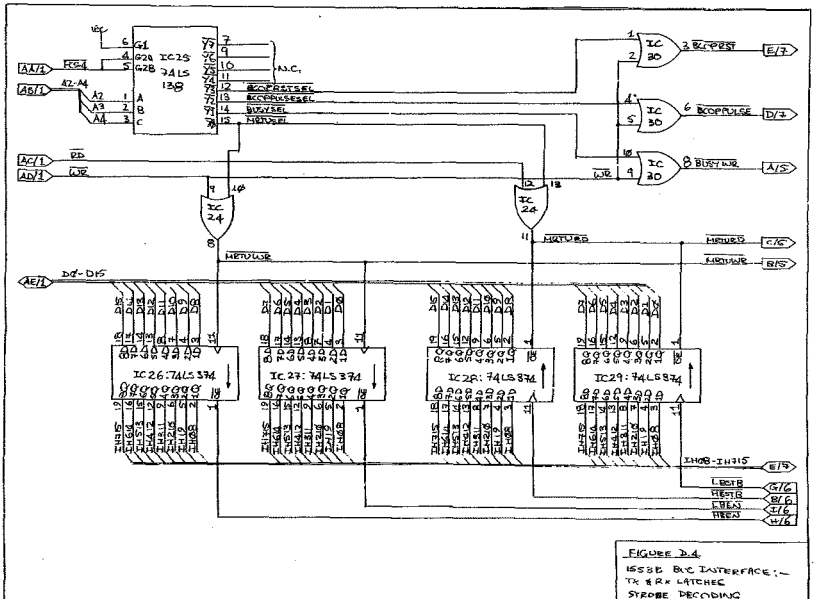


FIGURE D.4

ISSB BUS INTERFACE:-
TX & RX LATCHES
STROBE DECODING

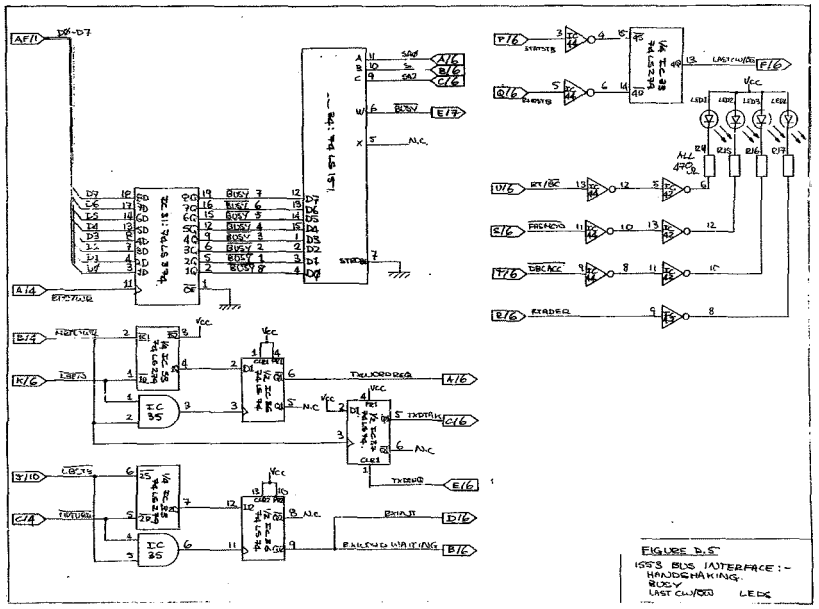


FIGURE 5.5
1553 BUS INTERFACE :- HANDSHAKING. BUSY LAST CHANNEL LEDS

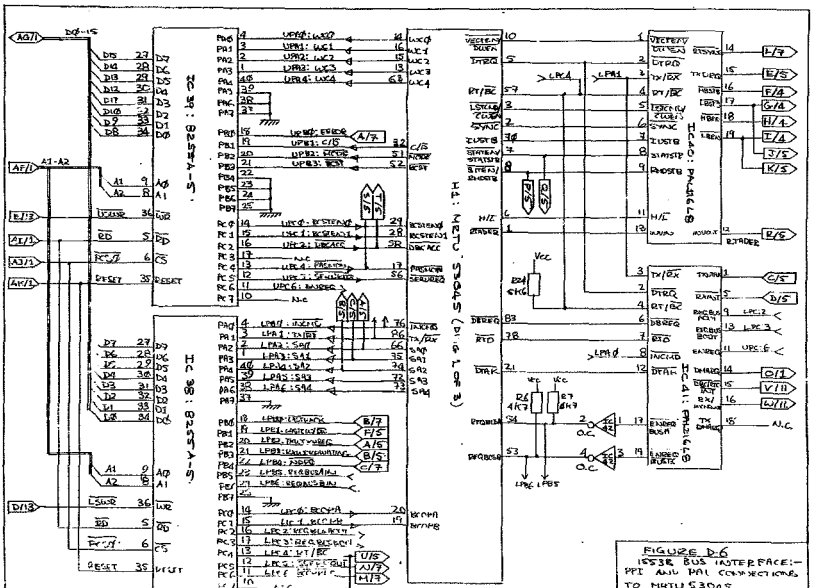


FIGURE D-6
1553B BUS INTERFACE:-
PPI AND PAL CONNECTIONS
TO MATU 5304S

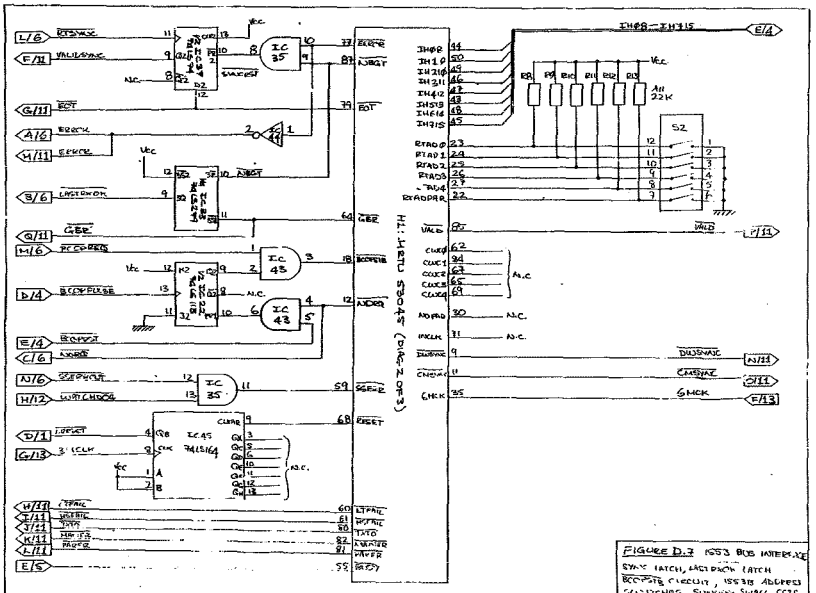


FIGURE D.7 1553 BUS INTERFACE
SYNC LATCH, LASTBOOK LATCH
BCOPSTB CIRCUIT, 1553B ADDRESS
SWITCHES, SUPPORT SPACE CCTS

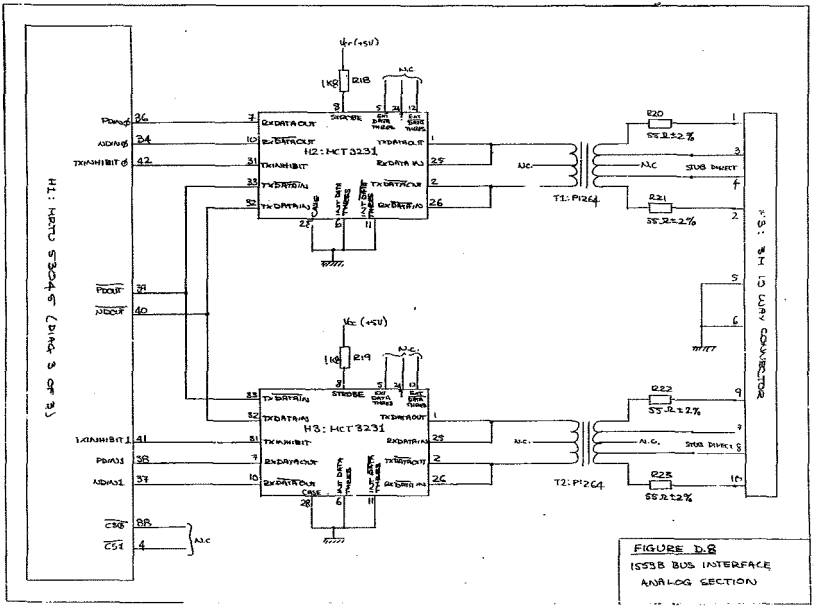


FIGURE D.8
1553B BUS INTERFACE
ANALOG SECTION

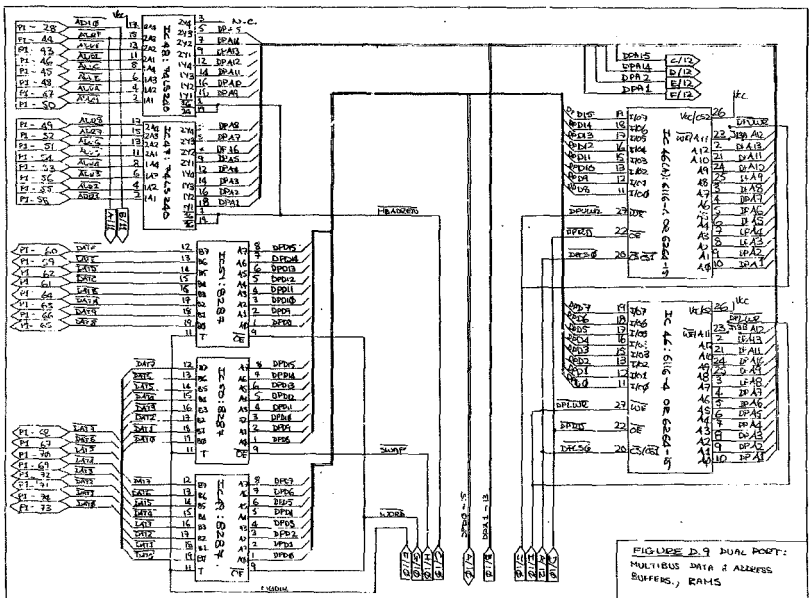


FIGURE D.9 DUAL PORT: MULTIBUS DATA & ADDRESS BUFFERS, RAMS

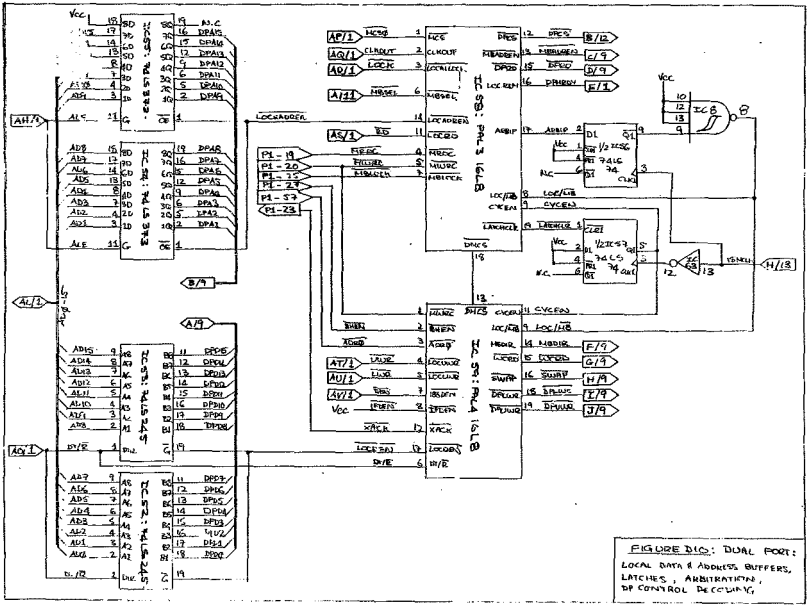


FIGURE D10: DUAL PORT:
LOCAL DATA & ADDRESS BUFFERS,
LATCHES, ARBITRATION,
& CONTROL DECODING

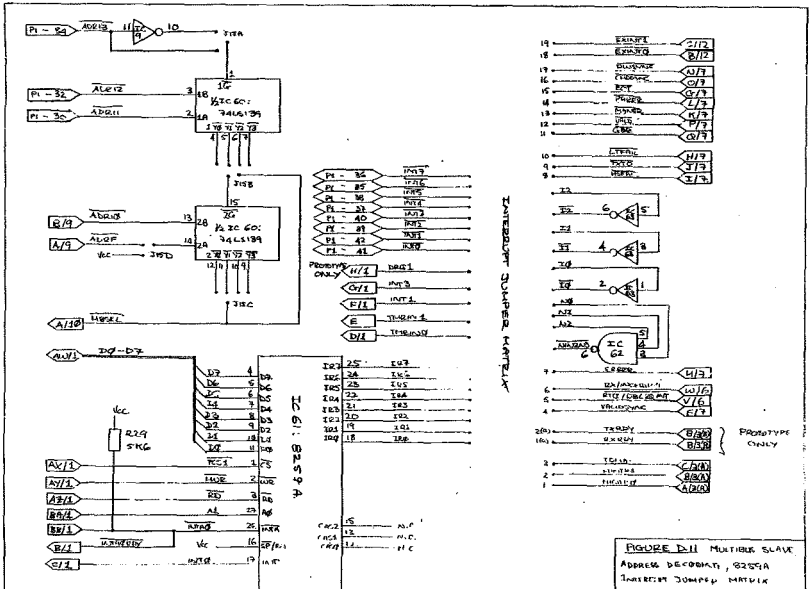


FIGURE D.11 MULTIBUS SLAVE ADDRESS DECODER, 8255A INTERRUPT JUMPER MATRIX

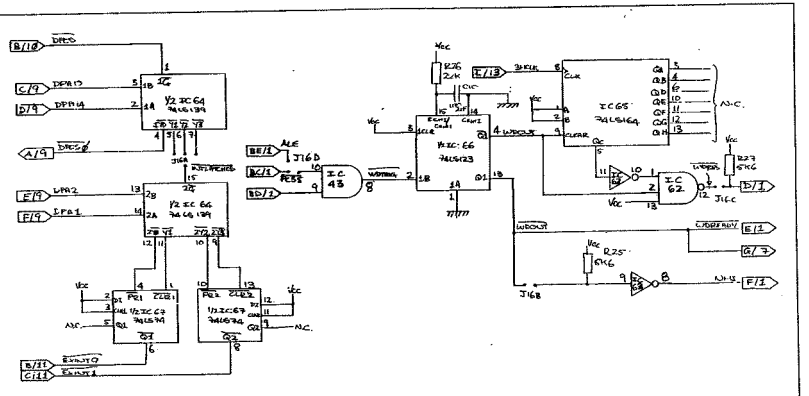


FIGURE D. 12 DUAL PORT:
INTERNAL DECODING
INTERRUPT LATCHES
WATCHDOG

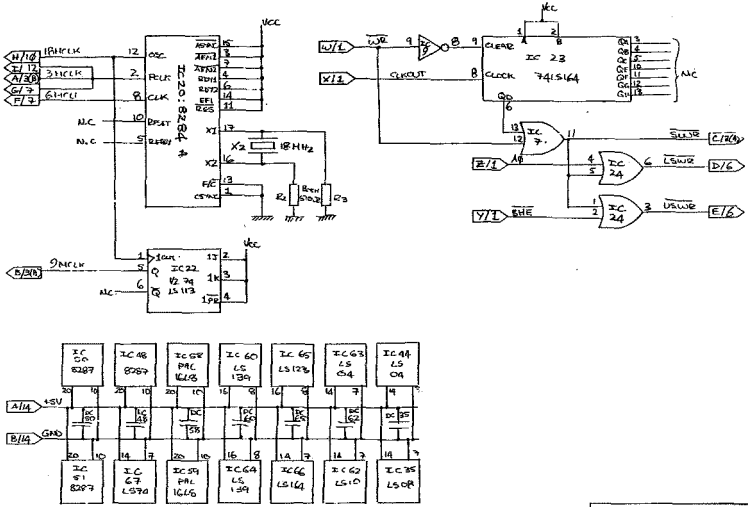


FIGURE D.13
CLOCK GENERATOR
SWR CIRCUIT
POWER SUPPLIES 1

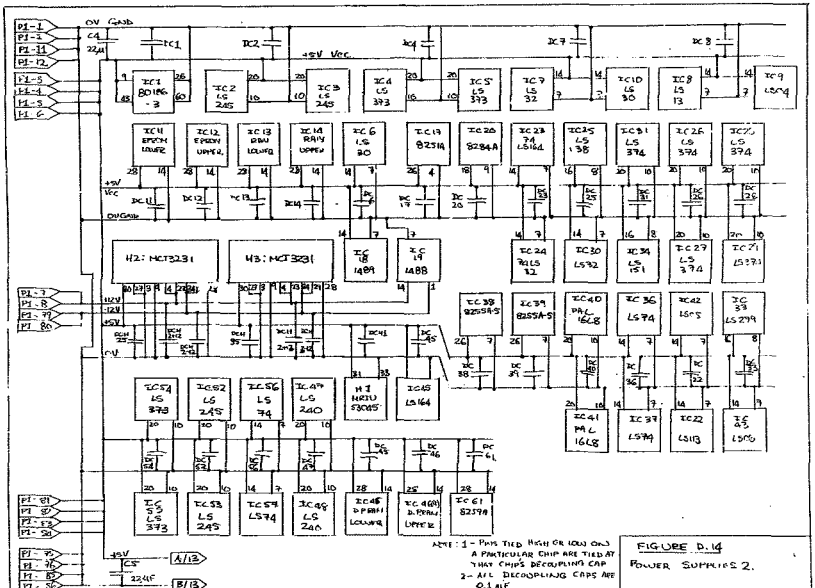


FIGURE D.14

POWER SUPPLIES 2.

INTEGRATED CIRCUITS

IC 1	80186-3	PROCESSOR 8MHz
IC 2	74LS 245	DATA BUS BUFFER UPPER
IC 3	74LS 245	DATA BUS BUFFER LOWER
IC 4	74LS 373	ADDRESS LATCH UPPER
IC 5	74LS 373	ADDRESS LATCH LOWER
IC 6	74LS 30	LOCAL SELECT DECODER
IC 7	74LS 32	
IC 8	74LS 13	RESET SCHMITT TRIGGER
IC 9	74LS 04	
IC 10	74LS 30	ARDY DRIVER
IC 11	2332 A 2764 27128 27256	LOWER EPROM ONE OF
IC 12	2332 A 2764 27128 27256	UPPER EPROM ONE OF
IC 13	6116-4 6264	LOWER LOCAL RAM ONE OF
IC 14	6116-4 6264	UPPER LOCAL RAM ONE OF
IC 15	* 74LS32	
IC 16	* 74LS32	
IC 17	Δ 8251A	USART
IC 18	Δ 1489	RS232C RX
IC 19	Δ 1488	RS232C TX
IC 20	8284A	CLOCK GENERATOR
IC 21	* 74LS123	SBX RESET
IC 22	74LS 113	32 FF & BCDP
IC 23	74LS 164	32X2 SHIFT REG
IC 24	74LS 32	
IC 25	74LS 138	HEXD LATCH/PULSE DECODER
IC 26	74LS 374	HEXD LATCHES
IC 27	74LS 374	
IC 28	74LS 374	
IC 29	74LS 374	
IC 30	74LS 32	
IC 31	74LS 374	BUSY LATCHES
IC 32	—	
IC 33	74LS 299	RS LATCHES
IC 34	74151	BUSY DECODER
IC 35	74LS 08	
IC 36	74LS 74	HANDSHAKING LATCHES
IC 37	74LS 74	
IC 38	8255A-5	
IC 39	8255A-5	
IC 40	PAL 16L8	
IC 41	PAL 16L8	
IC 42	74LS 02	
IC 43	74LS 08	

DISCRETE COMPONENTS

D1	1N4148	RESET CIRCUIT
R1	2K2 ¼W (5%)	RESET CIRCUIT
C1	100 μF (10VP)	RESET CIRCUIT
S1	SPST	RESET BUTTON
X1	16 MHz	PROCESSOR CRYSTAL
C2	20pF	AT X1
C3	20pF	AT X1
C4	22 μF TANT	SUPPLY DECOUPLING
C5	22μF TANT.	SUPPLY DECOUPLING
M1	* FEMALE 16 BIT	CONNECTOR: (SBX)
M2	Δ 25 PWAY SUB-MIN	D-CONNECTOR
C6	Δ 330 pF	RS 232C TX
C7	Δ 330pF	
C8	Δ 330pF	
X2	18 MHz	8284A CRYSTAL
R2	510Ω	AT X2
R3	510Ω	AT X2
R4	*22K	SBX RESET
C9	*0.1μ	SBX RESET
C10	*0.1μ	SBX RESET
R5	*2K2	SBX RESET
M3	MRU5 30/5	
R6	4K7	
R7	4K7	O/C PULL-UPS
R8-R13	22K	DIP SWITCH PULL-UPS
S2	6 DIP SWITCHES	
M2	MCT 3231	
M3	MCT 3231	
T1	PC64	
T2	PC64	
R18-R9	1K8	
R20-R23	562Ω ± 2%	
R24-R29	470Ω	LED CIRCUITS
LED1-LED6	RED LED'S	
R24	5K6	RT/RC PULL-UP
M3	3M 10-PWAY	CONNECTOR
R25	5K6	XMIT PULL-UP
R26	22K	
C10	100μF	WATCHDOG RC
R23	5K6	CODES PULL-UP
R28	5K6	HSTINT PULL-UP
R29	5K6	INTERRUPT PULL-UP

* SBX INTERFACE

Δ STANDARD SERIAL INTERFACE

DECOUPLING CAPACITORS - ALL 0.1μF :-

DC1, DC2, DC4, DC6, DC7, DC8, DC11, DC12, DC13, DC14, DC15, DC20, DC23, DC25, DC31, DC26, DC28, DCH25, DCH2+12, DCH2-12, DCH35, DCH3-12, DCH3+12, DCH1, DC45, DC38, DC37, DC40, DC36, DC22, DC33, DC52, DC53, DC56, DC42, DC46, DC46, DC61, DC50, DC49, DC58, DC60, DC68, DC62.

INTEGRATED CIRCUITS (CONT)

IC46	6116-4 6264	DP RAM LOWER ONE OF
IC46(4)	6116-4 6264	DP RAM UPPER ONE OF
IC 47	74LS 240	MB ADR BUFFER
IC 48	74LS 240	
IC 49	8287	
IC 50	8287	MB DATA BUFFER
IC 51	8259	
IC 52	74LS 245	DP DATA BUFFERS
IC 53	74LS 245	
IC 54	74LS 373	DP ADD LATCH
IC 55	74LS 373	
IC 56	74LS 74	
IC 57	74LS 74	
IC 58	1618	
IC 59	1618	
IC 60	74LS 138	MB BASE ADR DECODE
IC 61	8259A	PIC
IC 62	74LS 10	
IC 63	74LS 04	
IC 64	74LS 138	DP DECODING
IC 65	74LS 164	WATCH DOG
IC 66	74LS 123	
IC 67	74LS 74	
IC 14	74LS 04	
IC 9A	74LS 04	
IC 45	74LS 164	LOBSET SHIFT REG

TABLE D.1

COMPONENT LIST

D.3 PROTOTYPE BOARD LAYOUT

The following two drawings are the layout of the prototype wire wrap board. They are to scale. The the right hand edge of the second figure joins onto the left hand edge of the first.

FIGURE D.15(B)

D.4 JUMPER CONFIGURATION

The required connections of jumpers is obvious in many cases. Those that require further information are detailed below.

D.4.1 ROMs

The jumpers for configuring the pair of ROM sockets for various devices are on figure D.2.

TABLE D.2 ROM JUMPERS

DEVICES	J2A	J2B
2732A	N.C.	VCC/A13 - VCC
2764	PGM*/A14 - VCC	N.C.
27128	PGM*/A14 - VCC	VCC/A13 - A14
27156	PGM*/A14 - A15	VCC/A13 - A14

D.4.2 RAMs

Both the local RAM sockets in figure D.2 and the dual port RAM sockets in figure D.9 are configured in the same way. Note that both sockets in a pair must have the same configuration.

TABLE D.3 RAM JUMPERS

DEVICES	JUMPER
6116	WE*/A11 - WR* (UWR* or LWR*)
6264	WE*/A11 - A12

D.4.3 SBX Address

As mentioned in section C.4.2, there are two possible addressing schemes for the SBX interface. They are obtained as follows.

TABLE D.4 SBX JUMPERS

SCHEME	J3A	J3B
Byte Addressing	A4	A4*
Word Addressing	A0	BHE*

D.4.4 Multibus Slave Address Decoding

The following table shows how to connect the jumpers on figure D.11 to obtain a desired base address and size for the dual port memory as seen from Multibus.

TABLE D.5 MULTIBUS SLAVE ADDRESS DECODING JUMPERS

J15A:

Top 512k use ADR13*
Bottome 512k use ADR13*

J15B: (128k page within the 512)

0 - 128k	Y3*
128 - 256K	Y2*
512 - 384K	Y1*
384 - 512K	Y0*

J15J: (Select sub-page size)

64K size VCC

J15C: (64K sub-page within page)

0 - 64k	Y3*
64 - 128K	Y1*

32K size ADRF*

J15C: (32K sub-page within page)

0 - 32K	Y3*
32 - 64K	Y2*
64 - 96K	Y1*
96 - 128K	Y0*

APPENDIX E

PAL DESIGN AND PROGRAMMING

E.1 DESIGN PROCEDURE

No logic development aids were available to aid the implementation of circuits using PALs. The Stag programmer which was used, only accepted programming information in the form of the fuse map for the particular PAL. This meant that every logic function that was to be built into the PAL had to be translated from either circuit diagram, or truth table form, to a fuse map.

The procedure to put a circuit onto PAL is as follows:

- Each combinational logic function in the circuit is isolated and its inputs identified.

- The Boolean equation describing each function is derived. In the case of a circuit diagram this is done directly. In the case of a truth table, it may be done directly in the case of simple function, or a Karnaugh map may be used.

- The PAL or PALs are now chosen according to the number of inputs and outputs, and the required output function (ie. registered outputs or tri-statable I/O lines etc). This must be done taking into consideration the number of product terms required by the logic functions.

- If more than one PAL is required to provide sufficient input and output lines, a table of

each output and its required inputs is drawn up, and outputs having common inputs are grouped together. By this means the most compact grouping of functions is found. Pin assignments are then made, and the lines in the programmable matrix are labled.

- Very often it is convenient to choose a PAL which automatically inverts its outputs, if the functions to be implemented are active low. If this is the case, the Boolean equations of each function to be incorporated in the PAL, must be inverted.

- Boolean equations are now minimised and converted to sum of products form. This may be done with the aid of a Karnaugh map.

- If desired, outputs may be checked for possible glitching which may arise during transitions of the input lines. If necessary, extra redundant terms may be added to the Boolean equation to avoid these conditions.

- The fuse map of the PAL is then marked showing which fuses must be left intact to implement the the equation of each output function. All unused product terms must have all fuses intact to ensure that they are always low. Where it is required that a product term is always high (often to permanently enable a tri-state output), all fuses should be blown.

- Finally the fuse map is fed into the programmer and the PAL is programmed.

This procedure is both laborious and error prone. This is particularly true of the equation minimisation stage, the conversion to sum product form, and the plotting of fuse maps. Any serious use of PALs definitly requires a design aid.

The following sections contain all the essential information about the PALs used in the two separate areas of this design. The intermediate design steps are not shown in most cases. Where more than one version of the PAL was made, due to design or programming errors, only the final version is shown.

The following conventions are used:

All signal names are in capital letters. Those that are active low have an asterisk (*) after the name. For the sake of clarity, signal names such as RT/BC* are not used in truth tables, Boolean equations or fuse maps. Instead, RT represents this signal "high" and RT* represents it "low". The same is applied to TX/RX* (use TX or TX*), LOC/MB* (use LOC and LOC*), DT/R* (use DT and DT*), and H/L* (use H or H*), etc.

In truth tables the entry (x) indicates a "dont care" condition. Usually output are given "dont care" entries for combinations of inputs which should never occur.

E.2 BUS INTERFACE PALS

Two AMD 16L8 PALs were used. These have inverting outputs, thus all equations must be inverted. All I/O lines are either permanently input or permanently output. The product terms controlling the output tri-state buffers must thus be programmed accordingly, to be either

permanently high or permanently low.

The PALs are PAL 1 and PAL 2 in the board layout. Three versions of PAL 2 were made after design errors, and bugs in minimisation and transfering to the fuse maps were found. The most recent fuse maps are shown at the end of section E.2

The functions that have been put into these PALs are:

E.2.1 TX Latch Enable Lines

Outputs: LBEN* and HBEN*
Inputs: RT/BC*, TX/RX*, H/L, DTRQ*, DWEN*/VECTEN*, CWEN*

LBEN* and HBEN* are asserted under identical combinations of the inputs, except that LBEN* requires that H/L* is low, while HBEN* requires that it is high. The following truth table describing the lines thus omits H/L*.

The RX latches must be enabled in RT mode when:

- fetching a data word to be transmitted, or
- reading the vector word,

and in BC mode when:

- fetching a data word to be transmitted,
- loading a command word, or
- loading a single data word.

TABLE E.1 LBEN* AND HBEN

In RT mode (ie. RT = 1):

TX	DTRQ*	DWEN*/VECTEN*	LBEN* & HBEN*
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	x
1	0	1	0
1	1	0	0
1	1	1	1

In BC mode (ie. RT = 0):

TX	DTRQ*	DWEN*/VECTEN*	CWEN*	LBEN* & HBEN*
0	0	0	0	x
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	x
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	x
1	0	0	1	x
1	0	1	0	x
1	0	1	1	0
1	1	0	0	x
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1

Using a Karnaugh map, the minimal inverted Boolean expressions with no output glitches are:

```
LBEN = H*.VECTEN/DWEN +
       H*.DTRQ.TX +
       H*.RT*.CWEN

HBEN = H.VECTEN/DWEN +
       H.DTRQ.TX +
       H.RT*.CWEN
```

E.2.2 F Strobes

Outputs: L nd HBSTB*
Inputs: RT/BC*, H/L*, TX/RX*, DTRQ*, STATSTB, RMDSTB*, IUSTB, SYNC

Again, LBSTB* and HBSTB* are asserted under identical combinations of the inputs, except that LBSTB* requires that H/L* is low, while HBSTB* requires that it is high. The following truth table describing the lines thus omits H/L*.

The RX latches must be strobed in RT mode when:

- a data word is passed to subsystem, or
- a sync data word is received,

and in BC mode when:

- a data word is passed to subsystem,
- a status word is received, or
- a single data word is received.

TABLE E.2 LBSTB* AND HBSTB*

For RT mode (ie. RT = 1):

TX	DTRQ*	IUSTB	SYNC*	LBSTB* & HBSTB*
0	0	0	0	x
0	0	0	1	1
0	0	1	0	x
0	0	1	1	0
0	1	0	0	1
0	1	0	1	1
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	0
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	0
1	1	1	1	1

For BC mode (ie. RT = 0):

TX	DTRQ*	STATSTB	RMDSTB	IUSTB	LBSTB* & HBSTB*
0	0	0	0	0	1
0	0	0	0	1	0
0	0	0	1	0	x
0	0	0	1	1	x
0	0	1	0	0	x
0	0	1	0	1	x
0	0	1	1	0	x
0	0	1	1	1	x
0	1	0	0	0	1

TABLE E.2 (Continued)

TX	DTRQ*	STATSTB	RMDSTB	IUSTB	LBSTB* & HBSTB*
0	1	0	0	1	1
0	1	0	1	0	0
0	1	0	1	1	x
0	1	1	0	0	0
0	1	1	0	1	x
0	1	1	1	0	x
0	1	1	1	1	x
1	0	0	0	0	1
1	0	0	0	1	1
1	0	0	1	0	0
1	0	0	1	1	x
1	0	1	0	0	0
1	0	1	0	1	x
1	0	1	1	0	x
1	0	1	1	1	x
1	1	0	0	0	1
1	1	0	0	1	1
1	1	0	1	0	0
1	1	0	1	1	x
1	1	1	0	0	0
1	1	1	0	1	x
1	1	1	1	0	x
1	1	1	1	1	x

Using Karnaugh Maps, the minimised inverted glitch free Boolean expressions are:

```
LBSTB = H*.RT.SYNC.IUSTB +
        H*.TX*.IUSTB.DTRQ +
        H*.RT*.STATSTB +
        H*.RT*.RMDSTB

HBSTB = H.RT.SYNC.IUSTB +
        H.TX*.IUSTB.DTRQ +
        H.RT*.STATSTB +
        H.RT*.RMDSTB
```

E.2.3 Data Transfer Handshaking and DMA Requests

Outputs: DMARQ, TXDMAR[illegible], TXDTRQ, DTAK*
Inputs: RXINT, DTRQ*, TX/RX*, TXDTAK

The following circuit was designed and is now used to obtain Boolean expressions.

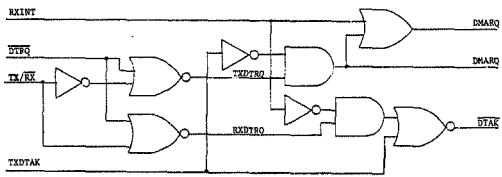


FIGURE E.1 HANDSHAKING AND DMA REQUEST LINES

The minimised glitch free inverted expressions are:

TXDTRQ* = TX* + DTRQ*

TXDMARQ* = DTRQ* + TX* +TXDTAK*

DMARQ* = RXINT*.DTRQ* +
RXINT*.TX* +
RXINT*.TXDTAK

DTAK = TXDTAK + RXINT.DTRQ.TX*

E.2.4 Sync Interrupt Strobe

Output: RTSYNC*
Inputs: SYNC*, RT/BC*

The following circuit is implemented:

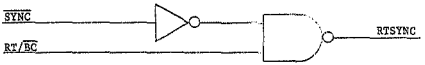


FIGURE E.2 RTSYNC LINE*

Inverted Boolean expression is:

RTSYNC = SYNC.RT

E.2.5 Bus Request Logic

Outputs: ENREQBUSA*, ENREQBUSB*
Inputs: REQBUSAOUT, REQBUSBOUT, ENREQ, RT/BC*

The following circuit drives the open collector gates on the hybrid:

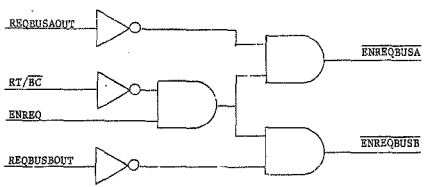


FIGURE E.3 ENREQBUS LINES

The inverted Boolean expressions are:

ENREQBUSA = REQBUSAOUT + RT + ENREQ*

ENREQBUSB = REQBUSBOUT + RT + ENREQ*

E.2.6 RXINT and INCMDINT Combination

Output: RX/INCMDINT
Inputs: RXINT, INCMD*, RT/BC*

The two interrupt lines are *combined as shown:*

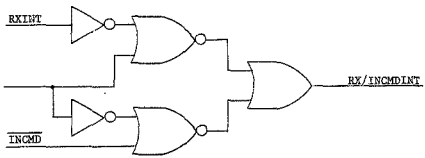


FIGURE E.4 RX/INCMDINT LINE

Inverted glitch free Boolean equation is:

```
RX/INCMDINT* = RXINT*.INCMD* +
               RXINT*.RT* +
               INCMD*.RT
```

E.2.7 DBCINT and RTOINT Combination

Output: DBC/RTOINT
Inputs: DBREQ*, RTO*, RT/BC*

These two interrupts are combined by the following circuit:

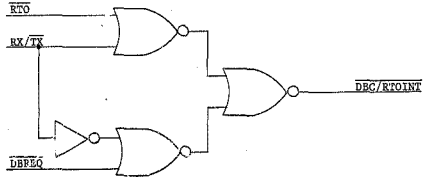


FIGURE E.5 DBC/RTOINT LINE

The inverted glitch free Boolean expression is:

```
DBC/RTOINT = RTO.RT* +
             DBREQ.RT +
             DBREQ.RTO
```

E.2.8 Extra PAL Lines

Output: INVOUT
Input: INVIN

After assignment of functions to the PALs, two lines were left over on one of the PALs. These are simply made into an inverter for general use.

After inversion :

INVOUT = INVIN

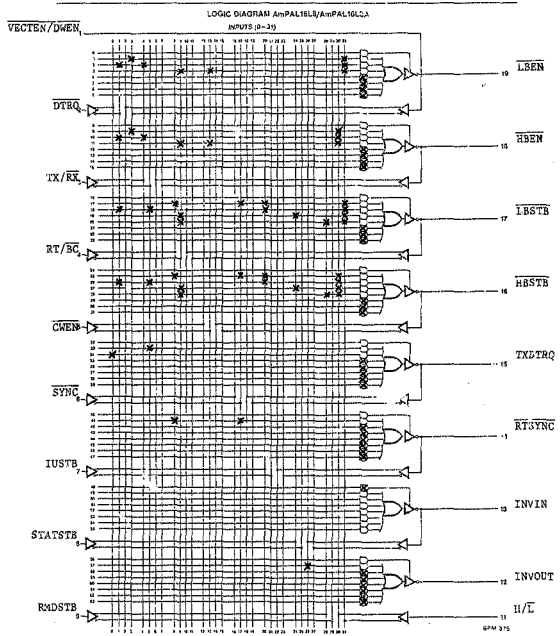


FIGURE E.6 PAL 1 VERSION 0

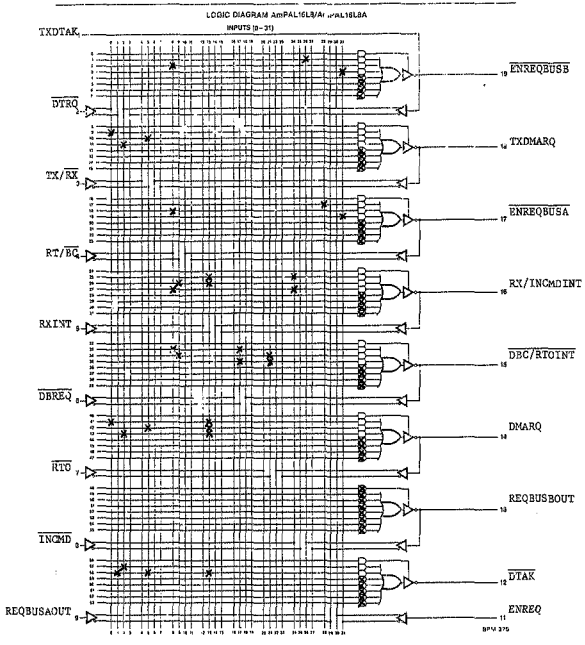


FIGURE E.7 PAL 2 VERSION 2

E.3 DUAL PORT ARBITRATION AND CONTROL PALS

Two AMD 16L8 PALs were also used for this circuit. Although the arbitration logic uses two latches, a PAL with registered output could not be used as the latches have to be independantly clockable. Once again each Boolean expression has to be inverted because of the PAL inverting outputs.

The PALs are PAL 3, which is mainly the arbitration logic, and PAL 4, which contains most of the control logic. Two versions of each were made. The fuse maps of the latest versions are shown at the end of this section E.3.

The arbitration and control logic is illustrated in the hardware design. The individual combinational functions are implemented as follows:

E.3.1 Delayed Memory Chip Select

Output: DMCS*
Inputs: MCS0*, CLKOUT

The circuit is:

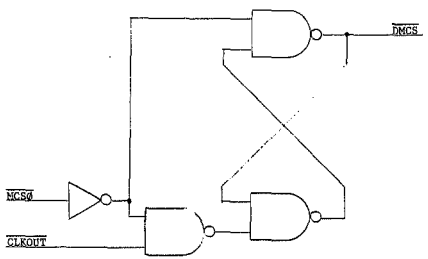


FIGURE E.8 DMCS* LINE

This circuit is different from those encountered previously in that it is not truly combinational. It implements an RS flip-flop (with some gating in front of it). To do this, the output has to be fed back into the circuit. It is also necessary to feed the input back as it is an input to the next function (ARBIP). This is acheived easily in the PAL using an I/O line. The inverted Boolean expression is:

```
DMCS = MCS0.DMCS +
       MCS0.CLKOUT
```

E.3.2 Arbitration Latch Input

Output: ARBIP
Inputs: DMCS*, LOCALLOCK, MBSEL*, MWRC*, MRDC*, MBLOCK*, LOC/MB*

The logic diagram was derived from the following truth table which describes the arbitration scheme outlined in C.6.6.1.

TABLE E.3 ARBITRATION TRUTH TABLE

LOC	LOCREQ	MBREQ	ARBIP
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

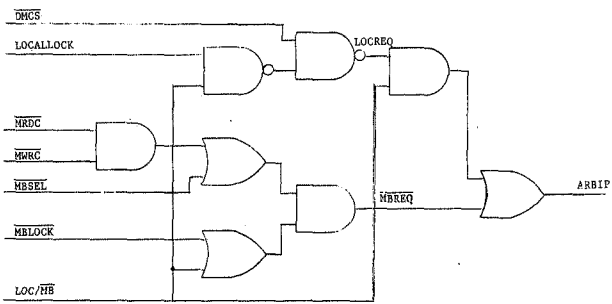


FIGURE E.9 ARBIP LINE

This line is also required by another function (LATCHCLR see next section). It is therefore also fed back into the PAL (but not back to itself). This circuit was minimised and checked for [illegible]t glitches by means of a Karnaugh map to give:

```
ARBIP* = DMCS*.LOCAL..CK*.MRDC.MBSEL +
         DMCS*.LOCALLOCK*.MWRC.MBSEL +
         LOC*.MRDC.MBSEL +
         LOC*.MRWC.MBSEL +
         MBLOCK.LOC*
```

E.3.3 Cycle Enable Latch Clear Line

Output: LATCHCLR*
Inputs: ARBIP, LOC/MB*

The output is simply the logical XNOR of the two inputs. Inverting, this becomes a logical XOR (:+:) as follows:

LATCHCLR = ARBIP :+: LOC

Expanding the XOR function, the Boolean expression to be programmed into the PAL is:

LATCHCLR = ARBIP.LOC* +
ARP .LOC

E.3.4 Control logic

Outputs: DPCS*, DPRD*, *DPUWR*, DPLWR*, LOCKREADY**, XACK*, LOCADREN*, MBADREN*, LOCDEN*, WORD*, SWAP*, MBDIR*
Inputs: LOC/MB*, CYCEN*, DMCS*, LOCRD*, LOCUWR*, LOCLWR*, DT/R*, 186DEN*, MRDC*, MWRC*, BHEN*, ADR0*, I/FDEN*

The control logic is shown in the figure E.10. It is simply one large combinational circuit.

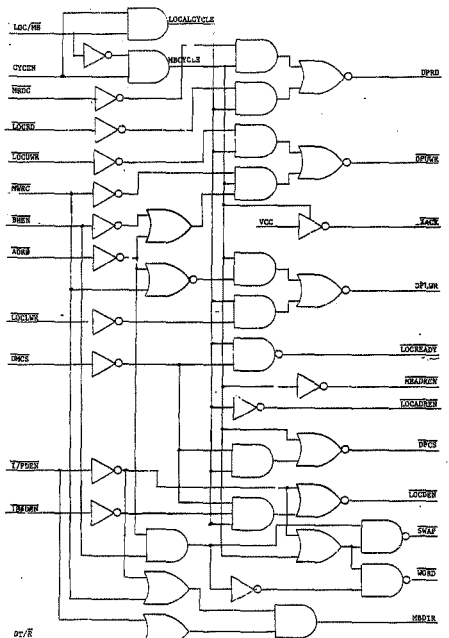


FIGURE E.10 CONTROL LOGIC

There is only one line requiring special explanation; XACK*. This output directly drives the Multibus XACK* line, so it must be tri-stated. It is implemented simply by setting the output always active low (by blowing all the fuses in one of the product terms). The Boolean expression, which consists of one product term is then programmed onto the tri-state enable line. The Boolean expression is thus:

```
XACK* tri-state enable = CYCEN.LOC*
```

The inverted glitch free minimised Boolean expressions for all the other lines are:

```
DPRD = MRDC.LOC*.CYCEN +
       LOCRD.LOC.CYCEN +
       MRDC.LOCRD.CYCEN

DPUWR = CYCEN.LOC.LOCUWR +
        CYCEN.LOC*.BHEN.MWRC +
        CYCEN.LOC*.ADR0.MWRC
        CYCEN.MWRC.LOCUWR.ADR0
        CYCEN.MWRC.LOCUWR.BHEN

DPLWR = LOC*.CYCEN.MWRC.ADR0* +
        LOC.CYCEN.LOCLWR +
        ADR0*.CYCEN.LOCLWR.MWRC

LOCDEN = I/FDEN +
         LOC*.CYCEN.186DEN.DMCS

MBDIR = MWRC.I/FDEN* +
        DT*.IFDEN
```

```
WORD = BHEN.CYCEN.LOC* +
       BHEN.I/FDEN +
       ADR0*.I/FDEN +
       ADR0*.CYCEN.LOC*

SWAP = BHEN*.ADR0.CYCEN.LOC* +
       BHEN*.ADR0.I/FDEN

MBADREN = LOC*.CYCEN

LOCADREN = LOC.CYCEN

LOCREADY = DMCS.LOC.CYCEN

DPCS = DMCS.LOC.CYCEN +
       LOC*.CYCEN
```

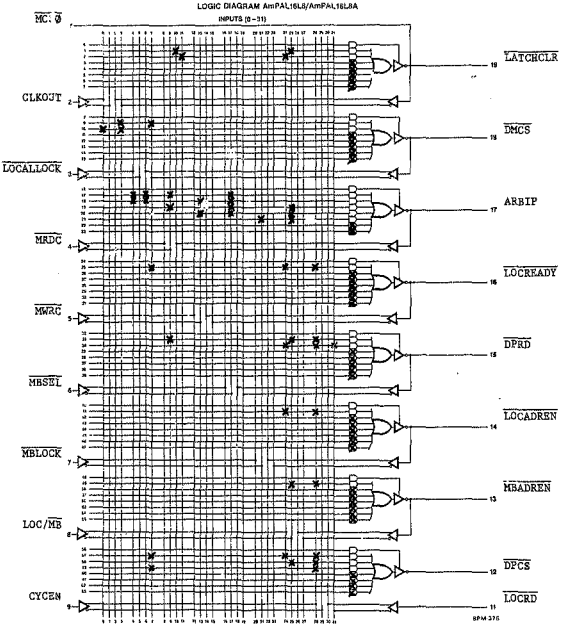


FIGURE E.11 FAL 3 VERSION 1

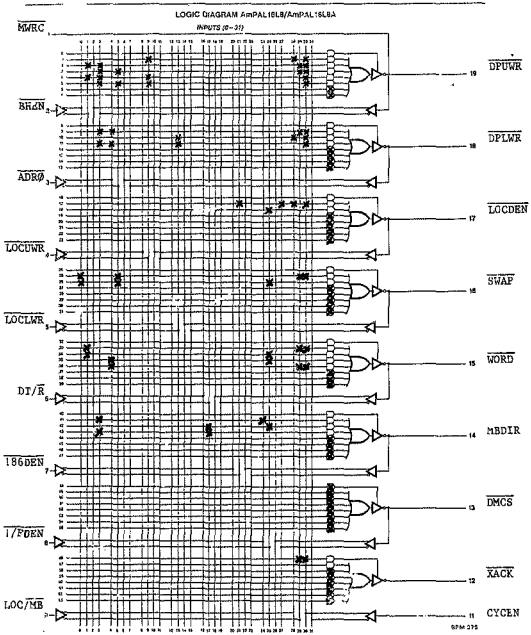


FIGURE E.12 PAL 4 VERSION 1

APPENDIX F

TEST SOFTWARE LISTINGS

The following software was written during the testing of the dual port memory and 1553B bus interface. It is representative of that which was written, both in assembler and Forth, for testing and debugging other parts of the design. No strict software design techniques were used when writing these routines, rather they are the result of adding necessary features as testing progressed. They should, however, illustrate how the 1553B bus interface hardware is driven and give some indication of the power of the Forth language.

F.1 ASSEMBLER HYB ,T

This is a self contained assembler program, which initialises the system and allows a choice of operations on the bus interface to be performed.

The initialisation code used in this program is an example of how the full prototype terminal system could be initialised. It is almost identical to that which precedes the jump into the Forth system. Note in particular that the MRTU 53045 hybrid is initialised to a known state as soon as possible after powering up.

```
"80186"
;
; TESTMNTU - INITIAL TESTS OF MRTU53045 HYBRID CHIP SET
;
CR              EQU             0DH
LF              EQU             0AH
CNTLZ           EQU             01AH

ICB_ADD         EQU             0FF00H          ;INTERNAL CONTROL BLOCK

UMCS_ADD        EQU             ICB_ADD+0A0H
LMCS_ADD        EQU             ICB_ADD+0A2H
PACS_ADD        EQU             ICB_ADD+0A4H
MMCS_ADD        EQU             ICB_ADD+0A6H
MPCS_ADD        EQU             ICB_ADD+0A8H

;CHIP SELECT REGISTER EQUATES
UMCS_VAL        EQU             0FE3F           ;8K ROM SIZE
                                                ;1 WAITS, ARDY IGNORED
LMCS_VAL        EQU             0BFCH           ;4K UPPER LIMIT
                                                ;0 WAITS, ARDY IGNORED
PACS_VAL        EQU             003DH           ;PBA=0
                                                ;1 WAIT, ARDY IGNORED
MPCS_VAL        EQU             013DH           ;MCS* LINES 2K EACH
                                                ;A1,A2 PROVIDED
                                                ;PCS* LINES IN I/O SPACE
                                                ;1 WAIT, ARDY IGNORED

PBA             EQU             0

                ORG             0FFFF0000H

;INITIALISE LMCS REGISTER FOR 8K ROM
                MOV             AX,LMCS_VAL
                MOV             DX,LMCS_ADD
                OUT             DX,AX

;JMP TO BOTTOM OF ROM
                JMP             FAR PTR INIT

;INITIALIZE CHIP SELECT REGISTERS
                ORG             0FE000000H
INIT            MOV             AX,LMCS_VAL
                MOV             DX,LMCS_ADD
                OUT             DX,AX
                MOV             AX,PACS_VAL
                MOV             DX,PACS_ADD
                OUT             DX,AX
                MOV             AX,MPCS_VAL
                MOV             DX,MPCS_ADD
                OUT             DX,AX

;PPI EQUATES
PCS0_BASE       EQU             PBA+8
PPI_PORTA_EVEN  EQU             PCS0_BASE+0
PPI_PORTA_ODD   EQU             PCS0_BASE+1
PPI_PORTB_EVEN  EQU             PCS0_BASE+2
PPI_PORTB_ODD   EQU             PCS0_BASE+3
PPI_PORTC_EVEN  EQU             PCS0_BASE+4
PPI_PORTC_ODD   EQU             PCS0_BA E+5
PPI_CONT_EVEN   EQU             PCS0_BASE+6
PPI_CONT_ODD    EQU             PCS0_BASE+7
```

```
PPI_MODE_WORD   EQU             9292H           ;WORD TO INITIALIZE BOTH PPIS AT ONCE:
                                                ;PORTS A - MODE 0 I/P
                                                ;PORTS B - MODE 0 I/P
                                                ;PORTS C - MODE 0 O/P

;INITIALIZE PPIS
                MOV             DX,PPI_CONT_EVEN
                MOV             AX,PPI_MODE_WORD
                OUT             DX,AX

;SET CHIP SET TO MINIMAL STATE
MRTU_NULL_WORD  EQU             0011010001110000B
                                                ;MSB:NC.
                                                ;ENREQ FALSE
                                                ;SERVEREQ# FALSE
                                                ;PASMON# FALSE
                                                ;NC.
                                                ;DBCACC# FALSE
                                                ;BCSTEN1 FALSE
                                                ;BCSTEN0 FALSE
                                                ;NC.
                                                ;BCOPREQ# FALSE
                                                ;SSERROUT# FALSE
                                                ;RT/BC# SET TO RT
                                                ;REQBUSBOUT FALSE
                                                ;REQBUSAOUT FALSE
                                                ;BCOPB FALSE
                                                ;BCOPA FALSE

                MOV             AX,MRTU_NULL_WORD
                MOV             DX,PPI_PORTC_EVEN
                OUT             DX,AX

;INITIALIZE STACK POINTER
                MOV             AX,0
                MOV             SS,AX
                MOV             SP,0FFFH

;TIMER EQUATES
MAXCOUNTA0_ADD  EQU             ICB_ADD+052H
MAXCOUNTB0_ADD  EQU             ICB_ADD+054H
CONT0_ADD       EQU             ICB_ADD+056H

MAXCOUNTA0_VAL  EQU             6               ;GIVES 9600 BAUD AT
MAXCOUNTB0_VAL  EQU             7               ;16X FACTOR
CONT0_VAL       EQU             0C093H          ;NO HALT ON MAX COUNT
                                                ;ALTERNATE MAX COUNT
                                                ;INTERNAL CLOCK
                                                ;NO PRESCALER
                                                ;NO RETRIGGER
                                                ;INTERRUPT DISABLED
                                                ;ENABLE COUNTER

;START BAUD RATE TIMER
                MOV             AX,MAXCOUNTA0_VAL
                MOV             DX,MAXCOUNTA0_ADD
                OUT             DX,AX
                MOV             AX,MAXCOUNTB0_VAL
                MOV             DX,MAXCOUNTB0_ADD
                OUT             DX,AX
                MOV             AX,CONT0_VAL
                MOV             DX,CONT0_ADD
```

```
                OUT             DX,AX

;USART EQUATES
PCS2_BASE       EQU             PBA+0100H
USART_DATA      EQU             PCS2_BASE
USART_STATUS    EQU             PCS2_BASE+2
USART_CONTROL   EQU             USART_STATUS

MODE_VAL        EQU             11001010B       ;2 STOP BITS
                                                ;PARITY DISABLE
                                                ;CHAR LENGTH 7 BITS
                                                ;BAUD RATE FACTOR 16X
COMMAND_VAL     EQU             00000101B       ;ENABLE RX AND TX
TX_RDY_MASK     EQU             01H
RX_RDY_MASK     EQU             02H

;INITIALIZE USART
                MOV             AL,0
                MOV             DX,USART_CONTROL
                OUT             DX,AL           ;ENSURE COMMAND REG
                CALL            DELAY           ;IS ADDRESSED
                OUT             DX,AL
                CALL            DELAY
                OUT             DX,AL
                CALL            DELAY
                MOV             AL,04H
                OUT             DX,AL           ;RESET USART
                CALL            DELAY
                MOV             AL,MODE_VAL
                OUT             DX,AL
                CALL            DELAY
                MOV             AL,COMMAND_VAL
                OUT             DX,AL
                CALL            DELAY

;LATCH EQUATES
PCS4_BASE       EQU             PBA+0200H
LATCH_ADD       EQU             PCS4_BASE+0
BUSY_ADD        EQU             PCS4_BASE+4
BCDPULSE_ADD    EQU             PCS4_BASE+8
BCDRST_ADD      EQU             PCS4_BASE+12

;DISPLAY MENU
INIT_MESS_OP    MOV             SI,OFFSET INIT_MESSAGE
                CALL            MESS_OP

MENU_LOOP       CALL            RX_CHAR
                CMP             AL,'B'          ;POLL FOR OPTION AND JUMP  TO
                JZ              BUS_CONT        ;SELECTED ROUTINE
                CMP             AL,'T'
                JZ              TRANS_COM
                CMP             AL,'M'
                JNZ             COMP_C
                JMP             MODE_DATA_COM
COMP_C          CMP             AL,'C'
                JNZ             MENU_LOOP
                JMP             CONT_TRANS_COM

SET_BC_MODE     EQU             00001000B       ;RESET PORT C BIT 4
SET_ENREQ_TRUE  EQU             00001101B       ;SET PORT C BIT 6

;CONFIGURE AS BUS CONTROLLER
BUS_CONT        CALL            TX_CHAR
```

```
                MOV             DX,BCOPRST_ADD
                OUT             DX,AX           ;ENSURE THAT THERE IS NO BCOPSTB*
                MOV             DX,PPI_CONT_EVEN
                MOV             AL,SET_BC_MODE
                OUT             DX,AL
                MOV             DX,PPI_CONT_ODD
                MOV             AL,SET_ENREQ_TRUE
                OUT             DX,AL           ;ENABLE BUSREQ LINES
                MOV             SI,OFFSET BUS_CONT_MESS
                CALL            MESS_OP
                JMP             INIT_MESS_OP

;TRANSMIT A COMMAND WORD
TRANS_COM       CALL            CHECK_BC
                JNZ             INIT_MESS_OP
OPT_MESS_OP     MOV             SI,OFFSET OPT_MESS
                CALL            MESS_OP
GET_OPT         CALL            RX_CHAR
                CMP             AL,'0'          ;INPUT COMMAND TYPE TO BE SENT
                JZ              OPT_ZERO
                CMP             AL,'1'
                JZ              OPT_ONE
                CMP             AL,'2'
                JZ              OPT_TWO
                CMP             AL,'3'
                JZ              OPT_THREE
                CMP             AL,'G'
                JZ              GO
                JMP             GET_OPT
OPT_ZERO        MOV             AH,00000000B
                JMP             OPT_SET
OPT_ONE         MOV             AH,00000001B
                JMP             OPT_SET
OPT_TWO         MOV             AH,00000010B
                JMP             OPT_SET
OPT_THREE       MOV             AH,00000011B
OPT_SET         CALL            TX_CHAR
                MOV             DX,PPI_PORTC_EVEN
                IN              AL,DX
                AND             AL,11110000B
                OR              AL,AH
                OUT             DX,AL           ;SET BCOP AND BUSREQ LINES AS REQUIRED
                JMP             OPT_MESS_OP
COMMAND_WORD    EQU             0000100000100011B ;AT ADDRESS=1
                                                  ;RECEIVE
                                                  ;SUBADDRESS=1
                                                  ;WORD COUNT=3
GO              MOV             AX,COMMAND_WORD
                MOV             DX,LATCH_ADD
                OUT             DX,AX           ;WRITE COMMAND WORD TO BE TRANSMITTED TO TX LATCHES
                MOV             DX,BCOPPULSE_ADD
                OUT             DX,AX           ;SET BCOPSTB
                MOV             CX,3            ;INITIALIZE COUNTER FOR 3 WORDS
WORD_LOOP       CALL            WAIT_TXWORDREQ
                MOV             AX,CX           ;SEND WORD COUNTER AS DATA
                MOV             DX,LATCH_ADD
                OUT             DX,AX
                LOOP            WORD_LOOP       ;UNTIL REQUIRED NO. OF WORDS HAVE BEEN SENT
                MOV             SI,OFFSET TRANS_COM_MESS
                CALL            MESS_OP
                JMP             INIT_MESS_OP

;TRANSMIT A COMMAND WORD FOLLOWED BY A DATA WORD
```

```
MODE_DATA_COM   CALL    CHECK_BC
                JZ      MODE_WORD_OP
                JMP     INIT_MESS_OP
MODE_WORD       EQU     0000000000001111B
MODE_W  _OP     MOV     AX,MODE_WORD
                MOV     DX,LATCH_ADD
                OUT     DX,AX           ;WRITE COMMAND WORD TO BE TRANSMITTED TO TX LATCHES
                MOV     DX,PPI_PORTC_EVEN
                IN      AL,DX
                AND     AL,11110000B
                OR      AL,00000001B    ;BUS NO. 0 OPTION 1
                OUT     DX,AL           ;SET BCOP AND BUSREQ LINES AS REQUIRED
                MOV     DX,BCOPPULSE_ADD
                OUT     DX,AX           ;SET BCOPSTB
DATA_WORD       EQU     1010101010101010B
                MOV     AX,DATA_WORD
                MOV     DX,LATCH_ADD
                CALL    WAIT_TXWORDREQ
                OUT     DX,AX           ;WRITE COMMAND WORD TO BE TRANSMITTED TO TX LATCHES
                MOV     DX,PPI_PORTC_EVEN
                IN      AL,DX
                AND     AL,11110000B
                OR      AL,00000010B    ;BUS NO. 0 OPTION 2
                CALL    WAIT_MDRQ
                OUT     DX,AL           ;SET BCOP AND BUSREQ LINES AS REQUIRED
                MOV     DX,BCOPPULSE_ADD
                OUT     DX,AX           ;SET BCOPSTB
                MOV     SI,OFFSET MODE_DATA_MESS
                CALL    MESS_OP
                JMP     INIT_MESS_OP

;TRANSMIT A COMMAND WORDS CONTINUOUSLY          ;THIS ROUTINE ALLOWS THE WAVEFORM
CONT_TRANS_COM  CALL    CHECK_BC                ;TO BE DISPLAYED ON AN OSCILLISCOPE
                JZ      CTC_MESS1_OP
                JMP     INIT_MESS_OP
CTC_MESS1_OP    MOV     SI,OFFSET CTC_MESS_1
                CALL    MESS_OP
COMMAND_WORD_1  EQU     0000010010000000B       ;RT ADDRESS=0
                                                ;RECEIVE
                                                ;SUBADDRESS=0
                                                ;WORD COUNT=32
CTC_LOOP        MOV     AX,COMMAND_WORD_1
                MOV     DX,LATCH_ADD
                OUT     DX,AX           ;WRITE COMMAND WORD TO BE TRANSMITTED TO TX LATCHES
                MOV     DX,PPI_PORTC_EVEN
                IN      AL,DX
                AND     AL,11110000B
                OR      AL,00000100B    ;BUS NO. 1 OPTION 0
                OUT     DX,AL           ;SET BCOP AND BUSREQ LINES AS REQUIRED
                MOV     DX,BCOPPULSE_ADD
                OUT     DX,AX           ;SET BCOPSTB
                CALL    RX_CHAR_TEST    ;SEE IF A CHARACTER HAS BEEN RECEIVED
                JNZ     CTC_END         ;IF SO END TRANSMISSION
                MOV     CX,000FH
WAIT            LOOP    WAIT            ;DELAY BEFORE TESTING MDRQ#
                CALL    WAIT_MDRQ
                JMP     CTC_LOOP        ;THEN BEGIN TRANSMISSION AGAIN
CTC_END         MOV     SI,OFFSET CTC_MESS_2
                CALL    MESS_OP
                JMP     INIT_MESS_OP

;ROUTINE TO CHECK THAT CHIP SET IS IN BC MODE
CHECK_BC        PUSH    DX
```

```
                PUSH    AX
                MOV     DX,PPI_PORTC_EVEN
                IN      AX,DX
                AND     AL,00010000B    ;CHECK MRTU IS IN BUS CONT MODE
                JZ      BC_OK
                MOV     SI,OFFSET BC_ERROR_MESS
                CALL    MESS_OP
                OR      AL,00100001B
                JMP     CHECK_BC_END
BC_OK           AND     AH,01000000B    ;CHECK ENREQ LINES ENABLED
                JNZ     ENREQ_OK
                MOV     SI,OFFSET ENREQ_ERR_MESS
                CALL    MESS_OP
                OR      AL,00000001B
                JMP     CHECK_BC_END
ENREQ_OK        AND     AL,00000000B
CHECK_BC_END    POP     AX
                POP     DX
                RET

;ROUTINE TO WAIT FOR TXWORDREQ LINE TO BECOME TRUE
WAIT_TXWORDREQ  PUSH    AX
                PUSH    DX
                MOV     DX,PPI_PORTB_EVEN
WAITTXREQ_LOOP  IN      AL,DX
                AND     AL,00001000B
                JZ      WAITTXREQ_LOOP
                POP     DX
                POP     AX
                RET

; ROUTINE TO WAIT FOR THE NDRQ LINE TO BECOME FALSE
WAIT_NDRQ       PUSH    AX
                PUSH    DX
                MOV     DX,PPI_PORTB_EVEN
WAITNDRQ_LOOP   IN      AL,DX
                AND     AL,00010000B
                JZ      WAITNDRQ_LOOP
                POP     DX
                POP     AX
                RET

; MESSAGE OUTPUT ROUTINE
MESS_OP         PUSH    AX
MESS_OP_LOOP    LODS    CS:BYTE PTR MESSAGES
                CMP     AL,CNTLZ        ;D/P CHARS TO USART
                JZ      RETURN          ;UNTIL CONTROL Z IS
                CALL    TX_CHAR         ;ENCOUNTERED
                JMP     MESS_OP_LOOP
RETURN          POP     AX
                RET

; TRANSMIT CHARACTER ROUTINE
TX_CHAR         PUSH    DX              ;SAVE REGS
                PUSH    AX
                MOV     DX,USART_STATUS
TX_LOOP         IN      AL,DX           ;WAIT FOR TX TO BE READY
                AND     AL,TX_RDY_MASK
                JZ      TX_LOOP
                MOV     DX,USART_DATA
                POP     AX              ;RESTORE CHAR
                OUT     DX,AL           ;TRANSMIT
                POP     DX
```

```
                RET

; RECEIVE CHARACTER ROUTINE
RX_CHAR         PUSH    DX
RX_LOOP         CALL    RX_CHAR_TEST    ;WAIT FOR CHAR TO ARRIVE
                JZ      RX_LOOP
                MOV     DX,USART_DATA
                IN      AL,DX           ;RECEIVE CHAR
                POP     DX
                RET

;TEST USART RECEIVER PROCEDURE
RX_CHAR_TEST    PUSH    DX
                MOV     DX,USART_STATUS
                IN      AL,DX
                AND     AL,RX_RDY_MASK
                POP     DX
                RET

;DELAY PROCEEDURE
DELAY           PROC    NEAR
                RET

MESSAGES        EQU     $
INIT_MESSAGE    DB      CR,LF,LF,'TESTMRTU - TEST MRTU53445 HYBRID CHIP SET',CR,LF,LF,
                DB      'CHOOSE FROM :',CR,LF
                DB      'CONFIGURE AS BUS CONTROLLER - B',CR,LF
                DB      'TRANSMIT A COMMAND WORD - T',CR,LF
                DB      'TRANSMIT A MODE COMMAND AND A DATA WORD - M',CR,LF
                DB      'CONTINUOUS COMMAND WORD TRANSMISSION - C',CR,LF,CNTLZ
BUS_CONT_MESS   DB      CR,LF,'BUS CONTROL MODE - BUSREQ LINES ENABLED',CR,LF,CNTLZ
BC_ERROR_MESS   DB      CR,LF,'ERROR - MTRU NOT IN BUS CONTROL MODE',CR,LF,CNTLZ
ENREQ_ERR_MESS  DB      CR,LF,'ERROR - BUSREQ LINES NOT ENABLED',CR,LF,CNTLZ
OPT_MESS        DB      CR,LF,'REQUIRED BUS CONTROL OPTION (0,1,2 OR 3) OR G TO EXECUTE : ',CNTLZ
TRANS_COM_MESS  DB      CR,LF,'SINGLE COMMAND WORD TRANSMITTED',CR,LF,CNTLZ
MODE_DATA_MESS  DB      CR,LF,'MODE COMMAND AND DATA WORD TRANSMITTED',CR,LF,CNTLZ
CTC_MESS_1      DB      CR,LF,'TRANSMITTING - HIT ANY KEY TO STOP',CR,LF,CNTLZ
CTC_MESS_2      DB      CR,LF,'TRANSMISSION TERMINATED',CR,LF,CNTLZ
```

F.2 FORTH PRIMITIVES

The following non-standard primitive assembler words had been added to the version of Forth used, specifically to test 8086 systems:

O! (word offset --) Stores a byte in the *location* calculated by using the contents of the variable SEGMENT as a *segment* base and adding the offset to it.

OC! (byte offset --) As above except a byte is stored.

O@ (offset -- word) Fetches a word from the location calculated by the method above.

OC@ (offset -- byte) As above except a byte is fetched.

EXCHANGE (word offset -- word) A locked exchange is performed with the word on the *stack being swapped for* the one in the location calculated by the same method as used above.

P! (word portadr --) The word is stored at the I/O port address.

PC! (byte portadr --) The byte is stored at the port address.

P@ (portadr -- word) The word at the I/O port address is fetched.

PC@ (portadr -- byte) The byte at the *port address* is

fetched.

The following primitives were added specifically to drive the bus interface. They effect the transfer of single data words in either direction between a bus controller and remote terminal. Polling of lines rather than interrupts are used. These operations cannot be done directly in Forth because a fast response is required.

BCRX (adr --) The hybrid (assumed in BC mode) is strobed to initiate command word transmission (the command word is assumed to be a receive command already in the TX latch). The word in the location specified by the address is supplied as a data word when the command word has been taken.

BCTX (adr --) Hybrid BC transmission of a transmit command is initiated as above. When a data word is received in the RX latch, it is stored in the location specified by the address.

RTSERVE (adr --) The hybrid is assumed to be in RT mode. It is polled until a command is received. If it is a transmit command, the word in the address is supplied to the TX latches. If it is a receive command, when a word arrives in the RX latch, it is written to the location given by the

address + 2.

The assembler code for these Forth primitives is appended at the end of the usual Forth dictionary. The code is given below:

```
;*************************************************************************
;
; THE FOLLOWING PRIMATIVES DRIVE THE MRTU 53045 CHIP SET
;
;*************************************************************************
;

;LATCH EQUATES
PCS4_BASE       EQU             PBA+020H
LATCH_ADD       EQU             PCS4_BASE+0
BUSY_ADD        EQU             PCS4_BASE+4
BCOPPULSE_ADD   EQU             PCS4_BASE+8
BCOPRST_ADD     EQU             PCS4_BASE+12

;-------------------------------------------------------------------------
;               BCRX            MRTU            RECEIVE COMAND IN BC MODE
;-------------------------------------------------------------------------
;
                HEADER          084H,LAST_STD_LABEL,M1,BCR,X
BCRX            CFACODE
                POP             AX
                PUSHA
                MOV             BX,AX
                MOV             DX,BCOPPULSE_ADD
                OUT             DX,AX
                MOV             DX,PPI_PORTB_EVEN
BCRX1           IN              AL,DX
                AND             AL,00001000B
                JZ              BCRX1
                MOV             AX,[BX]
                MOV             DX,LATCH_ADD
                OUT             DX,AX
                POPA
                NEXT


;
;-------------------------------------------------------------------------
;               BCTX            MRTU            TRANSMIT COMAND IN BC MODE
;-------------------------------------------------------------------------
;
                HEADER          084H,M1,M2,BCT,X
BCTX            CFACODE
                POP             AX
                PUSHA
                MOV             [X],AX
                MOV             [X],BCOPPULSE_ADD
                OUT             D,,AX
                MOV             DX,PPI_PORTB_EVEN
BCTX1           IN              AL,DX
                AND             AL,00001000B
                JZ              BCTX1
                MOV             DX,LATCH_ADD
                IN              AX,DX
                MOV             DX,PPI_PORTB_EVEN
BCTX2           IN              AL,DX
                AND             AL,00001000B
                JZ              BCTX2
                MOV             DX,LATCH_ADD
                IN              AX,DX
                MOV             [BX],AX
```

```
                POPA
                NEXT
;
;-----------------------------------------------------------------------
;               RTSERVE         MRTU            Rx/Tx SERVICE IN RT MODE
;-----------------------------------------------------------------------
;
                HEADER          087H,M2,LAST_LABEL,RTSERV,E
RTSERVE         CFACODE
                POP             AX
                PUSHA
                MOV             BX,AX
                MOV             DX,PPI_PORTA_EVEN
RTSERVE1        IN              AL,DX
                AND             AL,00000001B
                JNZ             RTSERVE1
                IN              AL,DX
                AND             AL,00000010B
                JZ              RTSERVE2
                MOV             AX,[BX]
                MOV             DX,LATCH_ADD
                NOP
                NOP
                OUT             DX,AX
                JMP             RTSERVE3
RTSERVE2        MOV             DX,PPI_PORTB_EVEN
RTSERVE4        IN              AL,DX
                AND             AL,00001000B
                JZ              RTSERVE4
                MOV             DX,LATCH_ADD
                IN              AX,DX
                INC             BX
                INC             BX
                MOV             [BX],AX
RTSERVE3        POPA
                NEXT




                NOP
XENDDICT        NOP
LENGTH_FORTH    EQU             $-XFORTH
```

F.3 DUAL PORT MEMORY LOCK TEST

The semaphore scheme implemented in Forth to test the lock feature of the dual port memory is given:

```
HEX

: DISP C EMIT 1 WAIT BEGIN CR 0 EMIT 1 WAIT     (DISPLAY FIST 100H MEMORY LOCATIONS)
100 0 DO I 0CR 4 .R LOOP ?TERMINAL UNTIL ;      (ABOVE SEGMENT BASE)


: SET-SEM       ( OFFSET --- )
  1 SWAP 0!
  ;

: GET-SEM       ( OFFSET --- )
  BEGIN
    DUP @ SWAP EXCHANGE
  UNTIL
  DROP
  ;

HEX


: COMPETE
  BEGIN
    0 GET-SEM
    '.' .'
    0 SET-SEM
    ?TERMINAL
  UNTIL
;
```

F.4 FORTH BUS INTERFACE DRIVERS

The following words were used to drive the 1553B bus interface:

```
HEX

( VARIOUS CHIP SET INTERFACE ADDRESSES )

: PPI-PORT-A 0 ;
: PPI-PORT-B 2 ;
: PPI-PORT-C-EVEN 4 ;
: PPI-PORT-C-ODD 5 ;
: PPI-PORT-C 4 ;
: PPI-CONT-EVEN 6 ;
: PPI-CONT-ODD 7 ;
: PPI-CONT 6 ;

: LATCH-ADD 280 ;
: BUSY-ADD 284 ;
: BCOPPULSE-ADD 288 ;
: BCOPRST-ADD 28C ;

( DMA CHANNEL ADDRESSES )

: ICR-ADD FF08 ;

: DMA-CONT ICR-ADD CA + ;
: DMA-TC ICR-ADD C8 + ;
: DMA-UF ICR-ADD C4 + ;
: DMA-SW ICR-ADD C8 + ;

( USEFUL DEFINITIONS )

: BINARY 2 BASE ! ;
: TRUE 1 ;
: FALSE 0 ;
: NOT 0 = ;

( WORDS TO TEST CHIP SET )

BINARY

: BCOPRST                       ( RESET BCOPSTB* LINE )
0 BCOPRST-ADD P! ;

: BCMODE                        ( CONFIGURE AS BUS CONTROLLER )
BCOPRST
00001000 PPI-CONT-EVEN PC!                      ( RESET PORT C BIT 4 )
                                ( THIS SETS BC MODE )
CR ." BUS CONTROL MODE" ;

: RTMODE                        ( CONFIGURE AS REMOTE TERMINAL )
00001001 PPI-CONT-EVEN PC!                      ( SET PORT C BIT 4 )
                                ( THIS SET RT MODE )
CR ." REMOTE TERMINAL MODE" ;

: ENREQ                         ( ENABLE BUS REQUEST LINES )
00011101 PPI-CONT-ODD PC!                       ( SET PORT C BIT 14 )
                                ( THIS ENABLES BUSREQ LINES )
CR ." BUSREQ LINES ENABLED" ;

: DISREQ                        ( DISABLE BUS REQUEST LINES )
00001100 PPI-CONT-ODD PC!                       ( RESET PORT C BIT 14)
                                ( THIS DISABLES BUSREQ LINES )
CR ." BUSREQ LINES DISABLED" ;
```

```
: ENRESET                       ( ENABLE MRTU RESET )
00006111 PPI-CONT-ODD PC!                       ( SET PORT C BIT 11)

CR ." MRTU RESET ENABLED" ;

: DISRESET                      ( DISABLE MRTU RESET )
00000113 PPI-CONT-ODD PC!                       ( RESET PORT C BIT 11)

CR ." MRTU RESET DISABLED" ;


: CHECK-BC                      ( CHECK THAT CHIP SET IS IN BC MODE )
PPI-PORT-C-EVEN PC@ 00010000 AND NOT            ( TEST IF "/BC# LINE )
IF
  PPI-PORT-C-ODD  PC@ 01000000 AND              ( IF BC THEN TEST ENREQ LINE )
  IF
    TRUE
  ELSE
    FALSE CR ." BUS REQUEST LINES NOT ENABLED"
  THEN
ELSE
  FALSE CR ." NOT IN BC MODE"
THEN ;

: BCOPSET                       ( SET BCOPSTB# LINE )
CHECK-BC
IF
  0 BCOPPULSE-ADD P!
THEN ;

: COM-SETUP                   ( DATA BUSNUMBER OPTION -- )
                              ( SET UP COMMAND WORD FOR TRANSMISSION )
00000011 AND                          ( MASK OPTION )
SWAP 00000011 AND [ HEX ] 4 [ BINARY ] *      ( MASK BUSNUMBER AND SHIFT 2 BITS LEFT )
OR                                    ( COMBINE THEM )
PPI-PORT-C-EVEN PC@ 11110000 AND              ( READ IN PORT VAL AND CLEAR LOWER 4 BITS )
OR PPI-PORT-C-EVEN PC!                        ( COMBINE WITH NEW LOWER 4 BITS AND WRITE OUT )

LATCH-ADD P! ;                                ( WRITE DATA TO TX LATCH )

: COM-OP
COM-SETUP BCOPSET ;

: RXCOM                       ( LOAD_ADR NETW_ADDR -- )
                              ( ISSUE SINGLE RX COMMAND )
[ DECIMAL ] 2048 [ BINARY ] *                 ( SHIFT RT ADDRESS TO BIT 11D )
0000000000100001                              ( RX , SUB_ADR 1 , WORD_COUNT 1 )
OR 0 0 COM-SETUP                              ( ALWAYS BUS 0 OPTION 0 )
CHECK-BC
IF
  BCRX
THEN
;

: TXCOM                       ( LOAD_ADR NETW_ADDR -- )
                              ( ISSUE SINGLE TX COMMAND )
[ DECIMAL ] 2048 [ BINARY ] *                 ( SHIFT RT ADDRESS TO BIT 11D )
0000010000100001                              ( TX , SUB_ADR 1 , WORD_COUNT 1 )
OR 0 0 COM-SETUP                              ( ALWAYS BUS 0 OPTION 0 )
CHECK-BC
IF
  BCTX
THEN
```

F.5 COMMAND INTERRUPT SERVICE ROUTINE

The following assembler routine is invoked when an interrupt is caused by the INCMD* line going active. The code is the result of much experimentation to find the quickest way to program the DMA channel, particularly in the case of a transmit command. The method ultimately used by this routine, is to leave the DMA channel permanently programmed for a transmit command. If a receive command occurs, the channel is reprogrammed, and then at the end of the receive command returned to original state.

```
"80186"
;
; INCMDINT - REMOTE TERMINAL INCMD INTERRUPT SERVICE ROUTINE
;

CR              EQU             0DH
LF              EQU             0AH
CNTLZ           EQU             01AH


PBA             EQU             0               ;PROGRAMMABLE BASE ADDRESS

;PPI EQUATES
PCS0_BASE       EQU             PBA+0
PPI_PORTA_EVEN  EQU             PCS0_BASE+0
PPI_PORTA_ODD   EQU             PCS0_BASE+1
PPI_PORTB_EVEN  EQU             PCS0_BASE+2
PPI_PORTB_ODD   EQU             PCS0_BASE+3
PPI_PORTC_EVEN  EQU             PCS0_BASE+4
PPI_PORTC_ODD   EQU             PCS0_BASE+5
PPI_CONT_EVEN   EQU             PCS0_BASE+6
PPI_CONT_ODD    EQU             PCS0_BASE+7

;LATCH EQUATES
PCS4_BASE       EQU             PBA+0C00H
LATCH_ADD       EQU             PCS4_BASE+0
BUSY_ADD        EQU             PCS4_BASE+4
BCOMPULSE_ADD   EQU             PCS4_BASE+8
BCOMRST_ADD     EQU             PCS4_BASE+12


ICB_ADD         EQU             0FF00H          ;INTERNAL CONTROL BLOCK

DMA_CONT        EQU             ICB_ADD+0CAH
DMA_TC          EQU             ICB_ADD+0CBH
DMA_DP          EQU             ICB_ADD+0C4H
DMA_SP          EQU             ICB_ADD+0C6H

DPM_BASE        EQU             1

EOI_REG         EQU             ICB_ADD+22H


;USART EQUATES
PCS2_BASE       EQU             PBA+0100H
USART_DATA      EQU             PCS2_BASE
USART_STATUS    EQU             PCS2_BASE+2
USART_CONTROL   EQU             USART_STATUS
TX_RDY_MASK     EQU             01H
RX_RDY_MASK     EQU             02H

                ORG             00000000H       ;SET INTERRUPT VECTORS TO POINT
INT_VECTORS     DDG             13              ;TO THIS ROUTINE
                DD              INCMDINT

                ORG             00402800H
INCMDINT        PUSH            AX
                PUSH            DX

                IN              AL,[PPI_PORTA_ODD]      ;READ IN WORD COUNT

                XOR             AH,AH
```

```
                MOV     DX,DMA_TC       ;WRITE TO DMA TERMINAL COUNT
                OUT     DX,AX
                IN      AL,IPP1_PORTA_EVEN  ;READ SUBSADDRESS AND TX/RX* LINE
                MOV     AH,AL
                XOR     AL,AL
                SHR     AX,1            ;SHIFT TO USE AS BLOCK ADDRESS
                SHR     AX,1
                TEST    AL,80H          ;TEST TX/RX* LINE
                JZ      RECEIVE

TRANSMIT        MOV     DX,DMA_SP       ;PROGRAM DMA SOURCE POINTER
                OUT     DX,AX
                MOV     DX,DMA_CONT
                MOV     AX,0001011010011113
                OUT     DX,AX
                                        ;ENABLE DMA CHANNEL
                MOV     DX,DMA_SP
                IN      AX,DX           ;FETCH SUBADDRESS AGAIN
                AND     AH,00011111B
                JZ      MODE_CMD_T
                XOR     AH,00011111B    ;SEE IF COMMAND WAS A MODE COMMAND
                JZ      MODE_CMD_T

                PUSH    SI
                MOV     SI,OFFSET TRANSMIT_MESS
                CALL    MESS_OP

                JMP     ENDROUTINE

RECEIVE         MOV     DX,DMA_DP
                OUT     DX,AX           ;PROGRAM DMA DESTINATION POINTER

                AND     AH,00011111B
                JZ      MODE_CMD_R      ;SEE IF COMMAND WAS A MODE COMMAND
                XOR     AH,00011111B
                JZ      MODE_CMD_R

                MOV     DX,DMA_DP+2     ;INITIALISE OTHER DMA REGISTERS
                MOV     AX,DPM_BASE
                OUT     DX,AX
                MOV     DX,DMA_SP
                MOV     AX,LATCH_ADD
                OUT     DX,AX
                MOV     DX,DMA_SP+2
                XOR     AX,AX
                OUT     DX,AX
                MOV     DX,DMA_CONT
                MOV     AX,1010001010011113
                OUT     DX,AX           ;ENABLE DMA CHANNEL

WAIT_DMA        MOV     DX,DMA_CONT
                IN      AX,DX
                AND     AX,02H
                JNZ     WAIT_DMA        ;WAIT UNTIL DMA IS COMPLETE

                MOV     DX,DMA_SP+2     ;REPROGRAM OTHER DMA REGISTERS FOR
                MOV     AX,DPM_BASE     ;TRANSMIT COMMAND
                OUT     DX,AX
                MOV     DX,DMA_DP
                MOV     AX,LATCH_ADD
                OUT     DX,AX
                MOV     DX,DMA_DP+2
                XOR     AX,AX
```

```
                OUT             DX,AX

                PUSH            SI
                MOV             SI,OFFSET RECEIVE_MESS
                CALL            MESS_OP

ENDROUTINE      MOV             DX,EOI_REG
                MOV             AX,8000H        ;SEND NON-SPECIFIC EOI COMMAND TO PIC
                OUT             DX,AX

                POP             SI
                POP             DX
                POP             AX
                IRET

MODE_CMD_T      MOV             DX,DMA_CONT
                XOR             AX,AX
                OUT             DX,AX           ;DISABLE DMA CHANNEL
MODE_CMD_R      PUSH            SI
                MOV             SI,OFFSET MODE_CMD_MESS
                CALL            MESS_OP
                JMP             ENDROUTINE


; MESSAGE OUTPUT ROUTINE
MESS_OP         PUSH            AX
MESS_OP_LOOP    LODS            CS:BYTE PTR MESSAGES
                CMP             AL,CNTLZ        ;D/P CHARS TO USART
                JZ              RETURN          ;UNTIL CONTROL Z IS
                CALL            TX_CHAR         ;ENCOUNTERED
                JMP             MESS_OP_LOOP
RETURN          POP             AX
                RET

; TRANSMIT CHARACTER ROUTINE
TX_CHAR         PUSH            DX              ;SAVE REGS
                PUSH            AX
                MOV             DX,USART_STATUS
TX_LOOP         IN              AL,DX           ;WAIT FOR TX TO BE READY
                AND             AL,TX_RDY_MASK
                JZ              TX_LOOP
                MOV             DX,USART_DATA
                POP             AX              ;RESTORE CHAR
                OUT             DX,AL           ;TRANSMIT
                POP             DX
                RET

MESSAGES
TRANSMIT_MESS   DB              CR,LF,'++++ TRANSMIT COMMAND ++++',CR,LF,CNTLZ
RECEIVE_MESS    DB              CR,LF,'++++ RECIVE COMMAND ++++',CR,LF,CNTLZ
MODE_CMD_MESS   DB              CR,LF,'++++ MODE COMMAND ++++',CR,LF,CNTLZ
```

REFERENCES

The citations are listed alphabetically under the sections to which they relate.

MIL-STD-1553B

1 HALEY A.L. "MIL-STD-1553 Validation test results"

2 JOHNSON B.W. and JULICH P.M. "Fault tolerant computer system for the A129 helicopter", IEEE Transaction on Aerospace and Electronic Systems, Vol. AES-21, No.2, March 1985, pp. 220-229.

3 Military Standard Aircraft Internal Time Division Command/Response Multiplex Data Bus, Washington D.C.: U.S. Department of defense MIL-STD-1553B, 21 September 1978.

4 MIL-STD-1553B Data Bus Summary, Issue 1, Marconi Electronic Devices Limited, Lincoln, England, March 1984.

5 MIL-STD-1553 Multiplex Applications Handbook, SCI Systems, Inc., Huntsville, Alabama.

6 SEABRIDGE A.G. and LANCASTER P.A. "The 'dual redundant' remote terminal in high integrity 1553B based systems", Electronic Engineering, January 1982, pp. 29-34.

7 SHAHSAVARI M.M., CALHOUN M.D., INGELS F.M., BENEDICK F., CUMMINGHAM P. and CONNELL C. "Error protection for hierarchical MIL-STD-1553B data bus structures", IEE Southeastcon, 1982, pp 37-40.

8 SUNDSTROM D.E. and EDWARDS J.A. "Inside MIL-STD-1553: Efficient embedded protocols", IEEE National Aerospace and Electronics Conference (NAECON), 1981, pp. 318-329.

NETWORKING

9 DESJARDINS R. and WHITE G. "Ansi Reference Model For Distributed Syste...

1553B BUS INTERFACE COMPONENTS

10 CT 1555 Data Terminal Bit Processor, Circuit Technology Incorporated, Farmingdale, N.Y.

11 DANCE M. "Mil 1553B data comms - a British niche", Electronics Industry, December 1982, pp. 11-15.

12 FRIEDMAN S.N. "MIL-STD-1553 dynamic bus controller/remote terminal hybrid set", IEEE National Aerospace and Electronics Conference (NAECON), 1983, pp. 639-644.

13 LEDAMUN D. and GOODWIN M. "Exploring the possibilities of the 1553B data bus", Electronic Engineering, March 1983, pp. 147-152.

14 MEYER D.H. and PARR D.R. "A MIL-STD-1553 flexible interface device and applications", IEEE National Aerospace and Electronics Conference (NAECON), 1983, pp. 618-624.

15 SCHAIRE S. and CAVIN J. "Single chip bus interface unit eases MIL-STD-1553B remote terminal/bus controller designs", IEEE National Aerospace and Electronics Conference (NAECON), 1982, pp. 864-871.

16 STC 1553 Hybrid Range Data Sheets, STC Components, Great Yarmouth, Norfolk.

17 WILLIAMS D.G. "Local Networks: Industrial controller joins the MIL-STD-1553 bus", Electronic Design, October 14, 1982 pp. 205-211.

MARCONI CHIP SET

18 MEDL 1553B LSI Chip Set - Remote Terminal Specification, Marconi Electronic Devices Limited, Lincoln, England.

19 MIL-STD-1553B Chipset for Bus Control Interface, Marconi Electronic Devices Limited, Lincoln, England.

20 Subsystem Interface for MEDL 1553B LSI Remote Terminals, Marconi Electronic Devices Limited, Lincoln, England.

MULTIBUS

21 GARROW R., JOHNSON J., and SOLTESZ L. "16-bit single-board computer maintains 8-bit family ties", Electronics, October 12, 1978, pp. 105-110.

22 iSBC 86/12 Single Board Computer Hardware Reference Manual, Intel Corporation, Santa Clara.

23 iSBC Applications Manual, Intel Corporation, Santa Clara, 1980.

24 Multibus Handbook, Intel Corporation, Santa Clara, 1983.

25 NADIR J. and McCORMICK B. "Bus arbiter streamlines multiprocessor design", Computer Design, June 1980, pp. 103-109.

26 WILSON D. "Multibus: Evolving to meet new system demands", Digital Design, February 1983, pp. 76-104.

80186/8086

27 "AP-186 application note", Microprocessor and Peripheral Handbook, Volume 1, Intel Corporation, Santa Clara, 1984.

28 HEMENWAY J. and TEJA E. "Increase 8086 throughput by using interrupts", EDN, May 20, 1979, pp. 179-183.

29 iAPX 86/88, 186/188 User's Manual: Programmer's Reference, Intel Corporation, Santa Clara, 1983.

FORTH

30 BRODIE L. Starting Forth, 1st ed. California: Prentice-Hall, 1981.

COMPONENT DATA

31 CTI MIL-STD-1553B Data Sheets, Circuit Technology Incorporated, Farmingdale, N.Y.

32 HM 6264 CMOS 64k Static Ram Data Sheet, Hitachi.

33 I.C. Memories Hitachi.

34 Logic Data Book (TTL), National Semiconductor Corporation, 1981.

35 MEDL MCT3231 Low Power Driver/Receiver Data Sheet, Issue 2, Marconi Electronic Devices Limited, Lincoln, England, March 1984.

36 MEDL MIL-STD-1553B Chip Set Data Sheet, Marconi Electronic Devices Limited, Lincoln, England.

37 MEDL MRTU 53045 Remote Terminal Bus Control Data Sheet, Issue 2, Marconi Electronic Devices Limited, Lincoln, England, March 1984.

38 Memory Components Handbook, Intel Corporation, Santa Clara 1983.

39 Microprocessor and Peripheral Handbook, Intel Corporation, Santa Clara, 1983.

40 PAL handbook, 3rd ed. Monolithic Memories Inc. Santa Clara, 1983.

41 Programmable Array Logic Handbook, Advanced Micro Devices, Inc. Sunnyvale, 1983.

42 Textool 68 Lead Chip Carrier Socket data sheet, Textool/3M, Irving.

Author Holt Geoffrey Anthony
Name of thesis Development Of A Mil-std-1553b Time Division Data Bus Terminal. 1985

PUBLISHER:
University of the Witwatersrand, Johannesburg
©2013

LEGAL NOTICES:

Copyright Notice: All materials on the University of the Witwatersrand, Johannesburg Library website are protected by South African copyright law and may not be distributed, transmitted, displayed, or otherwise published in any format, without the prior written permission of the copyright owner.

Disclaimer and Terms of Use: Provided that you maintain all copyright and other notices contained therein, you may download material (one machine readable copy and one print copy per page) for your personal and/or educational non-commercial use only.

The University of the Witwatersrand, Johannesburg, is not responsible for any errors or omissions and excludes any and all liability for any errors in or omissions from the information on the Library website.