

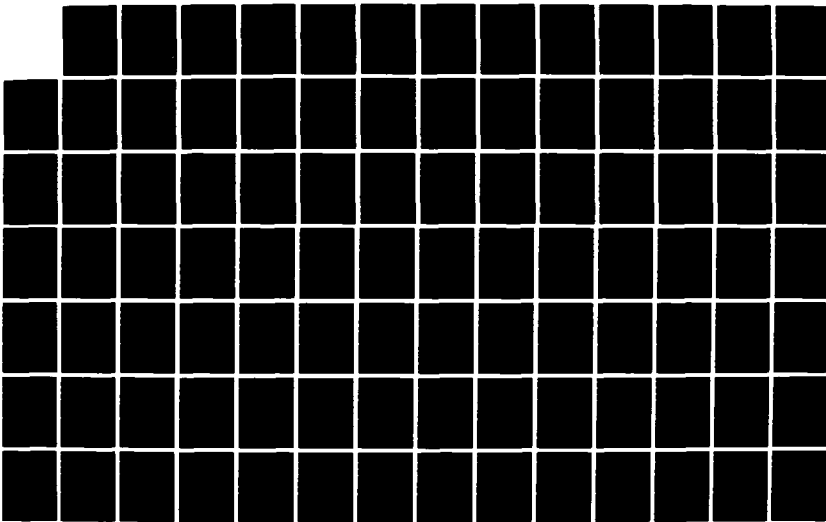
AD-A189 841

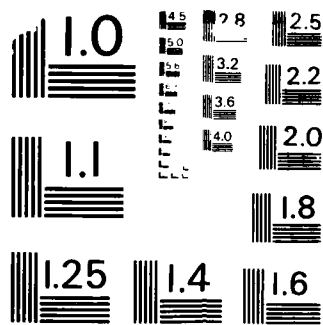
AN INVESTIGATION OF THE INTERFACING OF THE INTEL IAPX
432/670 COMPUTER SY.. (U) AIR FORCE INST OF TECH
WRIGHT-PATTERSON AFB OH SCHOOL OF ENGI.. L H BLACK
SEP 87 AFIT/GE/ENG/875-1 F/G 12/6

1/4

UNCLASSIFIED

NL

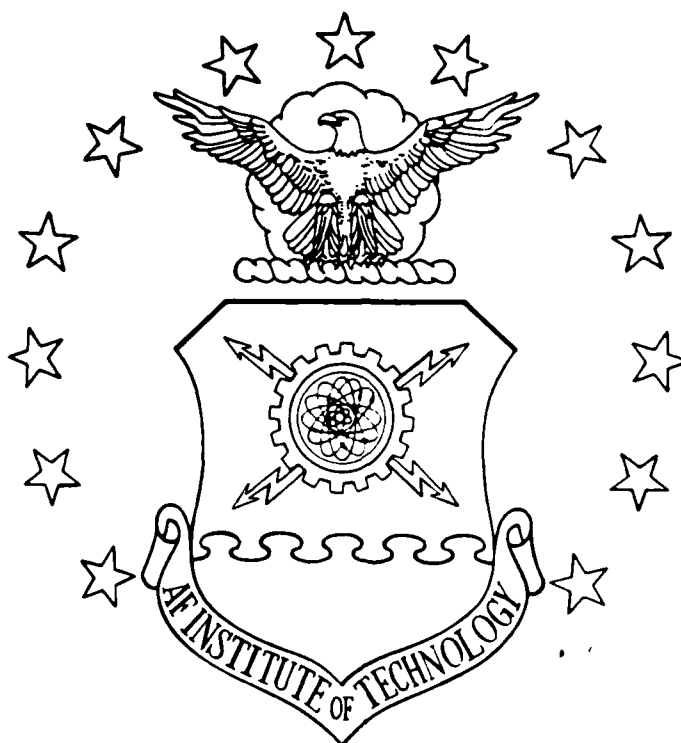




MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

DTIC FILE COPY

AD-A189 841



AN INVESTIGATION OF THE INTERFACING
OF THE
INTEL IAPX 432/670 COMPUTER SYSTEM
TO THE MIL-STD-1553B MULTIPLEX DATA BUS

THESIS

Lynn M. Black
Captain, USAF

AFIT/GE/ENG/87S-1

DTIC

ELECTE

MAR 07 1988

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

DISTRIBUTION STATEMENT A

Approved for public release;
Distribution Unlimited

88 3 01 042

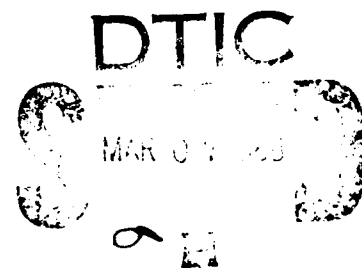
AFIT/GE/ENG/87S-1

AN INVESTIGATION OF THE INTERFACING
OF THE
INTEL iAPX 432/670 COMPUTER SYSTEM
TO THE MIL-STD-1553B MULTIPLEX DATA BUS

THESIS

Lynn M. Black
Captain, USAF

AFIT/GE/ENG/87S-1



Approved for public release; distribution unlimited.

AFIT/GE/ENG/87S-1

AN INVESTIGATION OF THE INTERFACING
OF THE
INTEL iAPX 432/670 COMPUTER SYSTEM
TO THE MIL-STD-1553B MULTIPLEX DATA BUS

THESIS

Presented to the Faculty of the School of Engineering
of the Air Force Institute of Technology
Air University
in Partial Fulfillment of the
Requirements for the Degree of
Master of Science in Electrical Engineering

Lynn M. Black, B.S., B.A., B.S.E.E.

Captain, USAF

September 1987

Approved for public release; distribution unlimited.

Preface

This project evolved following the introduction of the iAPX 432/600 line of computer systems by Intel Corporation. In an effort to promote this very innovative architecture, Intel donated numerous systems to universities interested in performing research in the new object-oriented programming languages. The Air Force was naturally interested in the 432/670, since its Ada compiler, although not validated, was one of the first released following approval of this new Department of Defense standard language for embedded computer systems. Maj Harold Carter, a faculty member in the AFIT electrical engineering department, was instrumental in obtaining a donated system for AFIT-sponsored research. While awaiting delivery of AFIT's system, several 432-related projects were proposed. These projects were all based on the premise that integration of the 432/670 into an Air Force avionic system would provide a much needed improvement in system processing capability. This project, involving interfacing of the 43/670 computer system to the relatively new MIL-STD-1553B multiplex data bus in order to provide centralized processing for the lower capability subsystems, was one of those proposed.

The original goal of this project was to implement and test the interface between the 432/670 and the 1553B data bus. Unfortunately, as often happens in the case of donations, particularly when the system involved is still in the development phase, the hardware did not arrive as scheduled. When delivered, three months prior to the completion of my AFIT assignment, the operating system for the Attached Processor (AP)

subsystem was in the form of a "user-configurable" operating system. Designing and testing this iRMX 88 operating system, which had been expected to be usable as delivered, greatly increased the scope of the project. In addition, since the hard disk for the Debugger Subsystem had not arrived at all, only trivial programs linked to the minimal iMAX operating system configuration could be downloaded to the 432/670. Since minimal iMAX does not support non-Debugger peripheral subsystems, no testing of the software would be possible. The late delivery, coupled with these two major deficiencies in the AFIT system, led to a last minute redirection of this project. Thus, this project evolved into an investigation of the interfacing of the 432/670 computer system and the 1553B bus, with the goal of demonstrating the feasibility of implementing this interface. The feasibility of the interface was not an accepted fact, since several AFIT faculty members doubted, with good reason, that the AP could meet the 1553B timing requirements.

I chose to approach this investigation by developing a conceptual design for a 432/670-based avionic subsystem, and then defining the requirements for this interface. I decided that the best way to prove the feasibility of the interface was to design both the hardware and software, including the iMAX and iRMX 88 operating systems, to a level at which I could show that all design requirements were met. During the course of the investigation, I found that, indeed, the AP was not fast enough to meet the timing requirements imposed by the chip set chosen to perform the remote terminal unit functions for the 1553B interface. Thus, these time-critical functions were moved into the hardware. Not only did this measure allow the last of the design requirements to be

met, proving the feasibility of the interface, but it also greatly enhanced the general applicability of the hardware interface design. Since the 432/670-specific portions of the software were also isolated to the degree possible, the resulting design can be easily modified for use with processors which may be more suitable for use in an avionic system.

The last minute redefinition of my project prevented me from finishing this project prior to my departure from AFIT. I am extremely grateful to Maj (now Dr) Harold Carter, my thesis advisor, for continuing to support my efforts, especially since his retirement from the Air Force. Likewise, I also owe thanks to my readers, Lt Col Walt Seward and Dr Tom Hartrum for their continued help. My greatest thanks go to my husband, Maj Roie Black for the unrelentless pushing which kept me motivated, despite my numerous other interests.

Table of Contents

	Page
Preface	ii
List of Figures	x
List of Tables	xii
Abstract	xiii
1. Introduction	1
1.1 Background	1
1.2 Problem	4
1.3 Scope	4
1.4 Approach	5
1.5 Thesis Development	6
2. The MIL-STD-1553B Bus	8
2.1 Introduction	8
2.2 Bus Operation	9
2.2.1 Transmission Method	9
2.2.2 Bus Protocol	10
2.3 Bus Hardware	17
2.3.1 Terminals	17
2.3.2 Bus Cable and Couplers	21
2.4 Bus Implementation Concerns	21
2.4.1 Bus Topology	22
2.4.2 Bus Control	24
2.4.3 Error Management	24

	Page
2.4.4 Redundancy and Functional Partitioning	25
2.4.5 Bus Loading	27
2.5 The DAIS Implementation	27
2.5.1 Bus Implementation	28
2.5.2 Operation Restrictions	29
3. The Intel iAPX 432/600 Computer Systems	32
3.1 Introduction	32
3.2 General Description	32
3.2.1 Processor Subsystem	34
3.2.2 Memory Subsystem	36
3.2.3 System Bus	37
3.2.4 Peripheral Subsystems	38
3.3 iAPX 432 Architecture	40
3.3.1 Object-Oriented Architecture	44
3.3.1.1 GDP Program Structure	45
3.3.1.2 IP Program Structure	48
3.3.1.3 Interprocessor Communication	50
3.3.2 General Data Processor Architecture	52
3.3.2.1 Instruction Coding	53
3.3.2.2 The GDP Pipeline	54
3.3.2.3 Addressing Mechanism	57
3.3.3 Interface Processor Architecture	59
3.4 iMAX Operating System	62
3.5 Ada Programming Environment	65
3.6 The AFIT Cross-Development System	69

	Page
4. Requirements	72
4.1 Introduction	72
4.2 Global Requirements	72
4.2.1 Flexibility	73
4.2.2 Reliability	73
4.2.3 Simplicity	73
4.2.4 Maintainability	73
4.2.5 Cost-Effectiveness	73
4.3 System Requirements	74
4.3.1 Computer System Configuration	74
4.3.1.1 432/670 Central System Software	74
4.3.1.2 Attached Processor Software	75
4.3.1.3 Attached Processor Hardware	75
4.3.2 MIL-STD-1553B Data Bus Configuration	76
4.3.2.1 Redundancy	76
4.3.2.2 Bus Coupling	76
4.3.2.3 Terminal Type	77
4.3.2.4 Transmission Method	77
4.3.2.5 Bus Protocol	77
4.3.2.6 Error Management	77
4.3.3 Operational Environment Constraints	77
5. Design Methodology	79
5.1 Introduction	79
5.2 Fundamentals of Structured Design	80

	Page
5.3 Design Tools	85
5.3.1 Data Flow Diagrams	85
5.3.2 Data Dictionary	87
5.3.3 Structured Design Diagrams	87
5.3.4 Analysis Methods	91
5.4 Hardware Design Methodology	93
5.5 Software Design Methodology	94
6. Hardware Design	96
6.1 System Concept	96
6.2 System Level Design	99
6.2.1 432/670 Central System	100
6.2.2 Debugger Subsystem	100
6.2.3 AP1 Subsystem	103
6.2.3.1 AP Module	104
6.2.3.2 1553B Bus Interface Module	105
6.2.3.2.1 RTU Module	106
6.2.3.2.2 RTU-1553B Interface Module	110
6.2.3.2.3 AP-RTU Module	110
6.3 432/670 Avionic Subsystem Concept of Operation	122
6.4 Hardware Design Analysis Summary	138
7. System Software Design	141
7.1 Introduction	141
7.2 Top Level Design	142

	Page
7.3 System Level Design	147
7.3.1 432/670 Central System Software	157
7.3.2 AP1 Subsystem Software	158
7.3.2.1 AP1 Executive	159
7.3.2.2 Input/Output Controller	161
7.4 Software Design Analysis Summary	162
8. Conclusions and Recommendations	165
8.1 Conclusions	165
8.2 Recommendations	168
Appendix A: The AFIT 432/670 CDS Configuration	170
Appendix B: Attached Processor Hardware Configuration Requirements	175
Appendix C: AP1 Hardware Design Block Diagrams	178
Appendix D: Hardware Signal Dictionary	219
Appendix E: Hardware Timing Diagrams	249
Appendix F: System Software Data Flow Diagrams	264
Appendix G: System Software Hierarchy Charts	278
Appendix H: Software Data Dictionary	285
Bibliography	312
Vita	315

List of Figures

Figure	Page
1. Manchester Data Encoding Scheme	10
2. MIL-STD-1553B Word Formats	12
3. MIL-STD-1553B Message Formats	16
4. Single Level Bus Topology	22
5. Multiple Level Bus Topology	23
6. Dual Redundant Bus/Terminal Configurations	26
7. Message Format for DAIS Mode Commands	31
8. Basic 432/670 System Organization	33
9. GDP Program Structure	46
10. IP Program Structure	49
11. The Three Stage GDP Pipeline	53
12. Logical to Physical Address Mapping Process	58
13. Program Development in the 432/670 Cross-Development Environment	68
14. AFIT's 432/670 Cross-Development System Configuration . .	70
15. Basic Data Flow Diagram	86
16. Data Flow Diagram Leveling	88
17. General Structure Chart Morphology	90
18. Symbol Conventions for Software Design Diagrams	95
19. 432/670 Avionic Subsystem Conceptual Design	97
20. 432/670 Avionic Subsystem Structure Chart	101
21. 1553BBI Module Block Diagram	107
22. General Transmit Command Timing Considerations	113

Figure	Page
23. General Receive Command Timing Considerations	114
24. Transmit Vector Word Timing	116
25. Essential Process Model	124
26. Software Context Diagram	143
27. Top Level Software Data Flow Diagram	144
28. Top Level Software Hierarchy Chart	146
29. System Level Software Data Flow Diagram	149

NOTE: Figures contained in Appendices are not shown in this table.

List of Tables

Table	Page
1. MIL-STD-1553B Mode Codes	13
2. DAIS BIT Word Format	30
3. A Comparison of the iAPX 432 Architecture and Conventional Mainframe Architectures	43

Abstract

This project was an investigation into the interfacing of the Intel iAPX 432/670 computer system to the MIL-STD-1553B data bus. The project involved developing a conceptual design for a military avionic subsystem based on the 432/670 computer system, defining the specific requirements for the subsystem, and demonstrating the feasibility of implementing the interface. The conceptual design of the 432/670 Avionic Subsystem includes development, operational and maintenance configurations for the subsystem. The feasibility of interfacing this subsystem to the 1553B data bus was proven by designing the subsystem to a level at which the design could be evaluated against the specific requirements levied upon the interface. Strict adherence to structured design techniques, and implementation of time-critical message processing functions in the hardware resulted, in relative isolation of the 432/670-related characteristics. The design presented in this project is thus readily adaptable for use in subsystems based on processors with more standard architectures.

AN INVESTIGATION OF THE INTERFACING OF THE
INTEL iAPX 423/670 COMPUTER SYSTEM TO THE
MIL-STD-1553B MULTIPLEX DATA BUS

CHAPTER 1

INTRODUCTION

1.1 Background

Modern Air Force aircraft are equipped with sophisticated avionic systems which rely heavily upon real-time data processing for reliable operation. Since the development of integrated circuits, new methods of integrating the avionics using small computers and microprocessors have evolved. Since size, weight and power requirements are critical considerations, a number of the avionic systems have been configured with one central computer performing all control and processing functions. Recognizing that a failure of either the computer or its interface would cause a significant degradation in aircraft performance, the Air Force Avionics Laboratory (AFAL) sponsored a program to develop a decentralized system called the Distributed Processor/Memory (DP/M) System (Ref 1:1-2).

The original DP/M System concept used a number of programmable processor/memory elements (PEs). Processing would be performed by either single, independent PE's or by clusters of PE's. A cluster, which is a group of PE's interconnected by a local bus, would be required when a single PE was unable to handle all of the processing for

a given subsystem. A dual-redundant global data bus using time division multiplexing (TDM) would interconnect all of the single PE's and PE clusters. This global bus would be the only shared resource in the system and the dual-redundant communication lines would enhance its reliability. Since the loss of a single PE or one of the bus lines would degrade system performance but not result in total system failure, the DP/M System was considered to be a distinct improvement over centralized systems (Ref 1:2-3).

The system used in most current avionics evolved from the DP/M concept. A MIL-STD-1553B aircraft internal time/division command/response multiplex data bus functions as the global system bus. Terminal units interface the individual subsystems to the bus. A maximum of 31 terminals can be connected to a 1553B bus. Functionally, a terminal is classified as either a bus controller, a bus monitor or a remote terminal. Address and command data, as well as signal information, are transferred between any of the terminals under the control of the single master bus controller (Ref 2:1-6). A bus monitor detects all data being transferred on the bus and extracts selected information for later use. A remote terminal is any terminal which does not operate as a bus controller or bus monitor. Its primary function is to receive and transmit data to the other terminals. The simplest remote terminal configurations provide the signal conditioning required to convert between the subsystem signal type and the Manchester encoded serial multiplexed data format required by the bus. Smarter remote terminals can provide back-up control capability to the system if they possess enough processing power (Ref 3:15). Real-time data processing

power is often quite limited since it is performed by the avionics subsystem during acquisition or by the bus controller's processor, both of which are usually implemented as small microprocessor systems in order to comply with size and weight restrictions. The performance of avionic systems could be greatly enhanced by providing real-time access to the processing capability of a mainframe computer. The small, yet very powerful, Intel iAPX 432 Micromainframe family is an excellent candidate for such an application.

The Intel iAPX 432 was designed to function as an embedded computer for computationally complex applications in which small physical size, low power consumption and dependability are essential (Ref 4:1). By combining Very Large Scale Integration (VLSI) technology with an innovative object-based architecture, a 32-bit microcomputer which provides mainframe functionality, increased reliability and security was created (Ref 5:1). Intel used a new approach to computer technology when designing the 432 Micromainframe family. They integrated the hardware, software and methodology in their iAPX 432/600 line of computer systems, hoping that the result would be a significant reduction in the life-cycle costs of complex microcomputer applications (Ref 4:2). Each 432/600 system is composed of a processor subsystem, a memory subsystem and at least one peripheral subsystem. The peripheral subsystems handle all of the input/output operations, linking the processor and memory subsystems to the external devices. Each peripheral subsystem contains its own processor and memory and runs under its own real-time multitasking operating system. A 432 Interface Processor (IP) board provides the hardware interface to the main system.

The modular nature of the subsystems allows the 432/600 computers to be easily tailored for specific applications. A wide range of processing power, memory capacity and input/output capability is available (Ref 4:9). Also of interest is the fact that the programming language of the iAPX 432 processor is Ada, the high level language sponsored by the Department of Defense (Ref 4:iii).

1.2 Problem

This study is an investigation into the development of a general purpose interface between the Intel iAPX 432/670 computer system and the MIL-STD-1553B data bus as implemented in the Digital Avionics Information System (DAIS). The powerful 432/670 computer is an excellent candidate for use in the sophisticated avionic systems of military aircraft. When interfaced to the 1553B data bus, an embedded 432/670 system could perform all of the functions of a remote terminal while simultaneously making its powerful data processing capability available to other terminals on the bus. In order to exploit the full power of the 432/670, the remote terminal functions should be performed by a peripheral subsystem. The central system, which contains the general data processors, is then reserved for the more complex processing tasks. Since these tasks are as yet undefined, the primary focus of this investigation is on the configuration of the peripheral subsystem.

1.3 Scope

This study had three main objectives. The first objective was to develop a conceptual design for an embedded 432/670 computer system for military applications. Defining the requirements for an interface

between the 432/670 computer system and the MIL-STD-1553B data bus was the second objective. The third objective was to create system level hardware and software designs of the interface which included sufficient detail for subsequent feasibility analysis. Features included in the conceptual design which would be desirable but not required in an operational system were not included in the design; however, the design would not preclude their addition in the future. The last objective was then to analyze the interface design and determine the viability of a follow-on implementation of a 432/670 computer-based avionic subsystem.

1.4 Approach

The first task required for this study involved conducting an extensive literature search. Since the requirements for the 1553B data bus and the remote terminals are published Military Standards and Prime Item Specifications, locating information on interfacing applications was relatively easy. Finding current information concerning interfacing applications of the Intel iAPX 432/670 computer system was much more difficult. The available literature concentrated on the system architecture, rather than on applications. Since few users appear to have attempted to implement non-Intel supplied peripheral subsystems, the only material which could be found was that included in the Intel reference manuals. The next step was to acquire an in-depth knowledge of the architecture and the operation of AFIT's 432/670 and the Cross Development System which is used for the development of the 432's software. This included gaining familiarity with the Intel 8086, the processor in the peripheral subsystem and iRMX 88, the user configurable real-time multitasking operating system supplied for the 8086. A

conceptual design of an embedded system appropriate for an avionic system application was then created. The problem of interfacing this system to the 1553B data bus was then attacked. The hardware and software requirements for the actual interface were determined. The peripheral subsystem was then designed at the system level. The 8086 processor hardware, the iRMX 88 operating system, the 1553B interface device driver and the interface hardware were included in this design. The modifications to the 432's iMAX operating system which were needed to allow communication with a new peripheral subsystem were then considered. An integral part of the design phase was an analysis of each of the various parts of the design. The final task which was undertaken was to generate recommendations for the future work in developing this system.

1.5 Thesis Development

This thesis was developed in eight sections. Chapter 2 contains a brief description of the MIL-STD-1553B bus and its implementation in the DAIS. The 432/600 line of computer systems is discussed in Chapter 3. This chapter includes a general description of the 432/600 computer systems, the architecture of the iAPX 432 chip set and the configuration of AFIT's 432/670 Cross Development System. The system requirements are presented in Chapter 4. Chapter 5 describes the methodology chosen to ensure rigorous development of both hardware and software designs. Chapter 6 presents the 432/670 Avionic Subsystem system concept and the system level design of the bus interface hardware. Both the top and system level designs of the software are considered in Chapter 7. A brief analysis of the design and how it

meets the stated requirements is included with each design. Chapter 8 covers the conclusions and the recommendations for the follow-on efforts needed for the system implementation.

CHAPTER 2

THE MIL-STD-1553B BUS

2.1 Introduction

The MIL-STD-1553B Aircraft Internal Time Division Command/Response Multiplex Data Bus was developed when the complexity of the air vehicle avionics made the use of separate, independent subsystems impractical. Avionics integration, the cooperative use of shared information among avionic subsystems, eliminates unnecessary duplication of information detection and display, improves system reliability and performance and saves space. Multiplexing data over a serial bus in which the input/output circuitry is partitioned and distributed was determined to be the best method for achieving this integration since no complex, centralized controller is required. MIL-STD-1553B, the current revision of the standard for avionic subsystem integration through multiplexing, defines the information transfer protocol and the electrical characteristics of the bus and the bus interface (Ref 6:2-1 - 2-3). Air Force Notice 1 to MIL-STD-1553B, issued in February 1980, restricts the operation of Air Force bus implementations in order to provide a degree of standardization among the systems (Ref 7:I-14). This chapter presents an overview of the operation of the MIL-STD-1553B data bus and the Notice 1 restrictions, the bus hardware components and the major bus implementation considerations. Emphasis is placed upon bus protocol, the data transmission method and the remote terminal functions, since each of these has a direct affect upon the

design of a subsystem interface. Since the actual bus implementation, being application dependent, affects the interface design, the Digital Avionics Information System (DAIS) implementation is also described briefly.

2.2 Bus Operation

The title assigned to MIL-STD-1553B, "Aircraft Internal Time Division Command/Response Multiplex Data Bus" essentially describes the operation of the data bus. The bus is intended to operate internally within a single aircraft, linking various subsystems via a shielded twisted-wire pair. Information is transferred serially over the bus in a half-duplex manner at a rate of 1.0 Megabits per second. The information, in the form of messages, is multiplexed onto the bus using a time-division multiplexing scheme. Message integrity is maintained due to the asynchronous command/response mode of operation in which a single bus controller initiates all message transfer and retains sole control over data transmission (Ref 6:7-6,7-7). The details of the physical transmission of data over the bus and the bus protocol governing message formatting and message exchange are discussed in the next two sections.

2.2.1 Transmission Method. Serial digital pulse code modulation (PCM) is used to transmit data over the MIL-STD-1553B bus. Baseband modulation is used to minimize the transmission bandwidth and the hardware complexity (Ref 6:2-16). The transmission frequency over the bus is nominally 1.0 MHz, with short term variations of no more than ± 100 Hz and long term variations less than ± 1000 Hz allowed. Data are coded for transmission using Manchester II bi-phase level encoding. As

shown in Figure 1, each bit is represented by both a positive and negative pulse of equal duration. Message synchronization is obtained by incorporating invalid Manchester II waveforms (Ref 6:7-7 - 7-9). The synchronization waveforms are defined by the bus protocol and are described in greater detail in the next section.

2.2.2 Bus Protocol. The MIL-STD-1553B bus protocol is a comprehensive set of rules which defines message formats and governs the exchange of messages between all subsystems resident on the bus. The standard interface provided via the bus protocol allows a variety of dissimilar systems to communicate with each other. Difficulties are occasionally encountered due to the multitude of subsystems being designed for the 1553B. In most cases the incompatibility problem can be solved simply by routing all messages through the bus controller which can be programmed to perform format translation. Designers

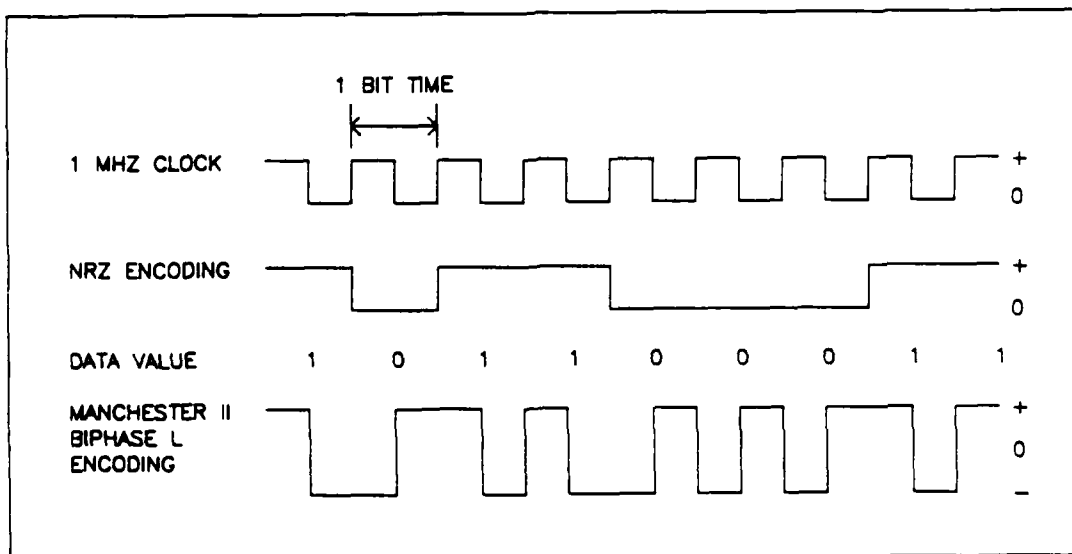


Figure 1. Manchester Data Encoding Scheme (Ref 6:7-8)

planning to add new subsystems to existing buses should take care to adhere strictly to the bus protocol and to use standard engineering units since modification of bus controller software may not be possible (Ref 6:11-1,11-2). Reference 6, Chapter 11 provides a detailed description of standard formats for commonly used data words and the Interface Control Document (ICD) format to be used for system specific word definitions.

The lowest level defined by the bus protocol is the word. Three types of words, command, status and data, are defined. Each word is 20 bits in length, with 3 bit times reserved for the unique synchronization waveform, 16 bits for information and 1 bit for parity (odd). The format of the each of the three word types is shown in Figure 2. The command word contains the address of the remote terminal to respond to the command (a unique address between 0 and 30 decimal), direction as to whether data is to be transmitted or received (T/R=0 for receive, T/R=1 for transmit), a subaddress/mode indication field and a data word count/mode code field. The subaddress/mode field contains a specific remote terminal subaddress if the value falls between 00001 and 11110 binary. A value of 00000 or 11111 binary indicates that the last field should be interpreted as a mode command instead of a word count for a message transmission. The mode commands provide the designer an efficient means for bus management; however, their use is optional. The standard mode code assignments, including those prohibited by Air Force Notice 1, are listed in Table 1. The data word contains contains 16 bits of data. A maximum of 32 words can be transmitted in a single block since the command word uses only 5 bits for word count. The status word

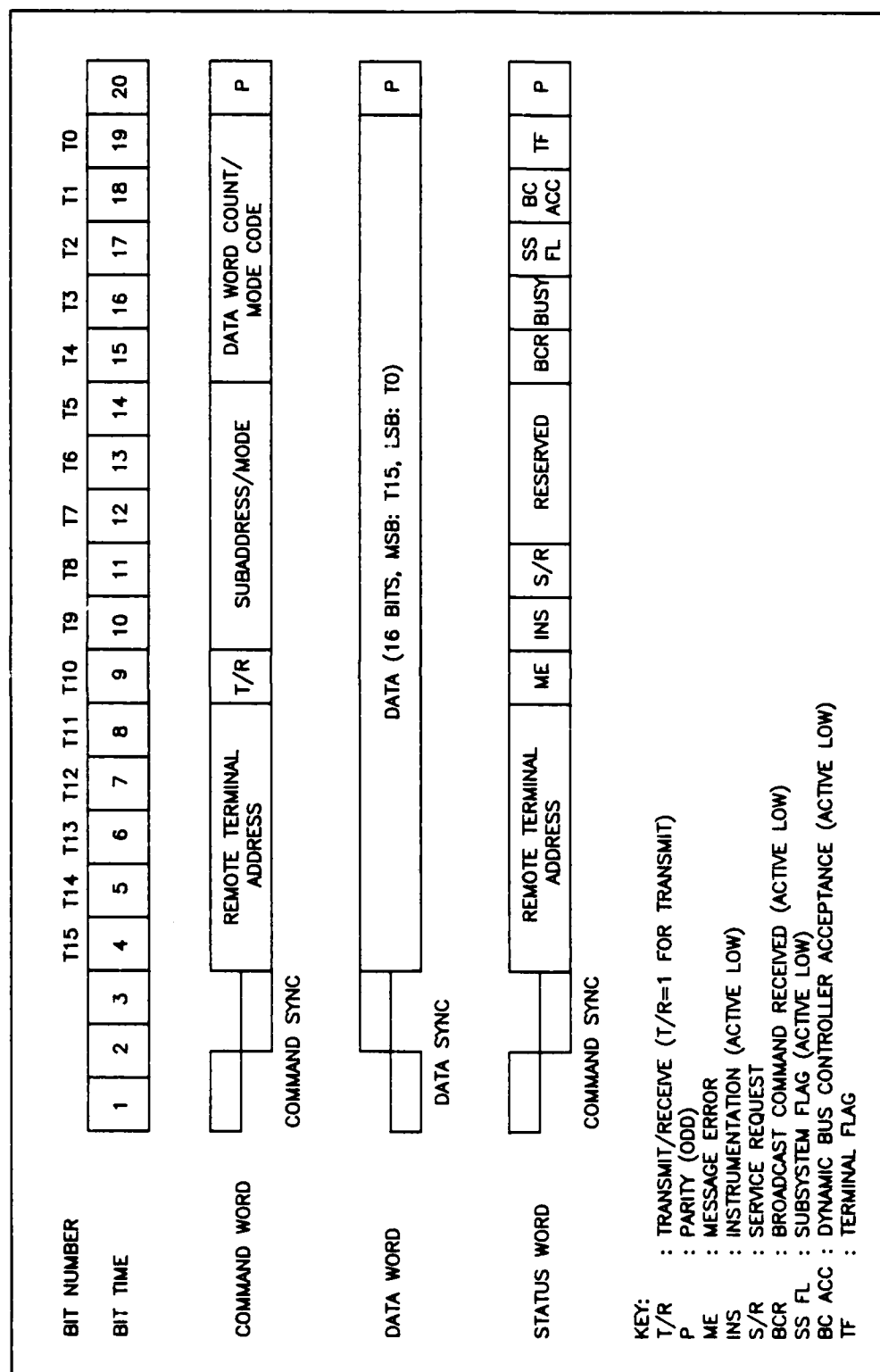


Figure 2. MIL-STD-1553B Word Formats (Ref 6:7-9)

Table 1. MIL-STD-1553B Mode Codes (Ref 7:I-6, I-15)

MODE CODE	FUNCTION	ASSOCIATED DATA WORD
* 00000	Dynamic Bus Control	No
00001	Synchronize	No
00010	Transmit Status Word	No
00011	Initiate Self-Test	No
00100	Transmitter Shutdown	No
00101	Override Transmitter Shutdown	No
* 00110	Inhibit Terminal Flag Bit	No
* 00111	Override Inhibit Terminal Flag Bit	No
01000	Reset Remote Terminal	No
01001	Reserved	No
↓	↓	↓
01111	Reserved	No
10000	Transmit Vector Word	Yes
10001	Synchronize	Yes
10010	Transmit Last Command	Yes
10011	Transmit BIT Word	Yes
* 10100	Selected Transmitter Shutdown	Yes
* 10101	Override Sel Transmitter Shutdown	Yes
10110	Reserved	Yes
↓	↓	↓
11111	Reserved	Yes

* Use restricted by Air Force Notice 1

provides the handshaking required by the command/response operation. Following receipt of a message from the bus controller, the addressed remote terminal must reply by transmitting the contents of its status word which contains its address, a message validity flag (error=1), an instrumentation bit (always logic zero), a service request bit (service request=1), a broadcast command received flag (broadcast command=1), a terminal busy flag (busy=1), an optional subsystem fault flag (fault=1), an optional dynamic bus control acceptance bit (backup bus controllers only, acceptance=1) and an optional terminal flag bit for indicating a remote terminal fault (fault=1). The use of the optional bits in the status word is implementation dependent, as is the use of the status word itself for some message types (Ref 6:7-8 - 7-28).

The highest level of the bus protocol defines message structure. Each message is composed of a series of words, response time gaps and intermessage time gaps. The message formats define the legal combinations and the ordering of word types and the placement of the appropriate time gaps. Ten "information transfer formats" are defined for the messages. Six of the formats support the command/response philosophy of the bus using status words for handshaking. The other four, designated "broadcast information transfer formats", are used for simultaneously transmitting messages to all remote terminals on the bus. The broadcast mode is specified by setting the remote terminal address of the command word to 11111 binary. While broadcasting may prove beneficial due to reduced overhead and ease of subsystem synchronization, deviation from the command/response mode, with the associated loss in fault monitoring capability and the possible increase

in system complexity due to use of subaddressing, is usually not recommended (Ref 6:7-10). In fact, Air Force Notice 1 forbids the use of the broadcast option (Ref 7:I-15).

The six standard message formats accepted by the Air Force are shown in Figure 3. The first three formats are used for data transfer. Each message consists of a command word containing the transfer control information and up to 32 data words. These formats allow for transmission of data between remote terminal units (RTUs) using the bus controller as an intermediary for routing and/or word translation and for direct transmission of data between remote terminals. The remaining three formats are used for issuing mode control commands to the remote terminals. One format is for a mode command with no additional data requirements, one is for a mode command with a single associated data (control information) word and one is for a mode command which requires that the remote terminal respond by transmitting a data word in addition to its status word. The intermessage gaps (12.0 microseconds minimum) and message response time (4.0 - 12.0 microseconds) shown in each message format are integral parts of the format and are essential for the maintenance of proper bus operation. The bus protocol also requires that a terminal fail-safe time-out (800 microseconds), which controls the maximum length of any single message, and a no-response time-out (14.0 microseconds), which sets a limit on how long the bus controller must wait for terminal to respond, be implemented (Ref 7:I-24). The interface between the data bus hardware and an individual subsystem must be designed with the message protocol in mind if timing problems due to hardware limitations are to be avoided.

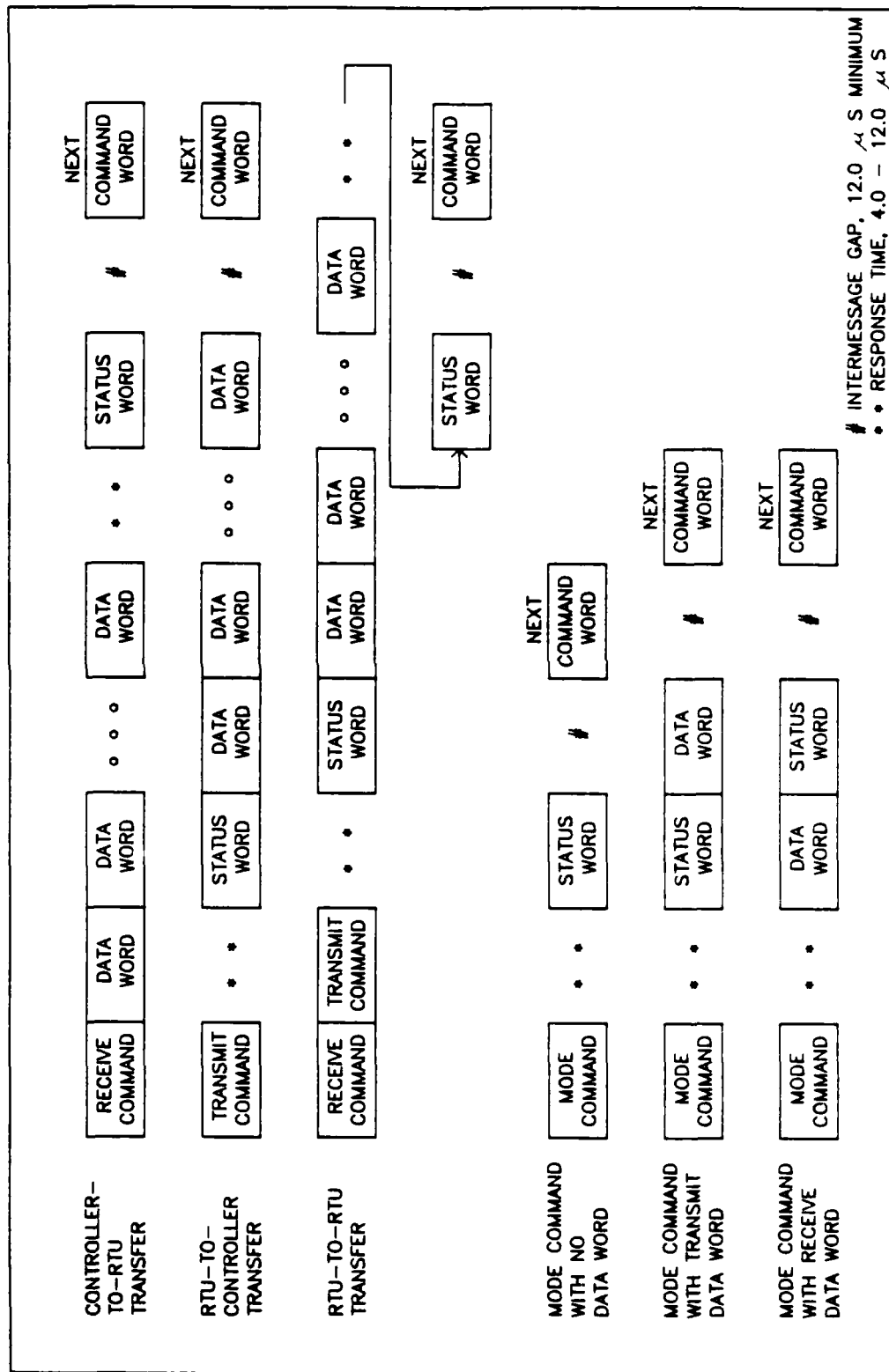


Figure 3. MIL-STD-1553B Message Formats (Ref 6:7-30)

2.3 Bus Hardware

MIL-STD-1553B defines the major hardware components which are used in 1553B data bus systems. Included in the description are the functions and the electrical characteristics of all of the terminal units which provide either data flow control, data storage or subsystem interfacing, as well as the electrical characteristics of the bus cable and the bus couplers which link the cable to the individual terminal units. This section provides a brief description of the function of all of these components since each component plays a significant role in the operation of the bus. The remote terminal unit and the bus couplers are discussed in the greatest detail since they actually provide the subsystem interface. The electrical characteristics of the hardware are much too detailed to be described here; however, those which have an impact upon the design of the subsystem interface are mentioned in either the interface requirements or the design description section.

2.3.1 Terminals. MIL-STD-1553B defines a terminal as an electronic module which is used to interface a subsystem to a data bus and a data bus to a subsystem (Ref 6:4-39). Thus, a terminal must be capable of both receiving and transmitting messages. This requirement is consistent with the requirement that the bus operate in a command/response mode. A great deal of flexibility in terminal design and capability is permitted. A terminal can be either an independent unit or an integral part of the subsystem. It may possess a great deal of processing capability or it may be able to perform only at the very basic receive/transmit level. Terminals are classified as either bus

controllers, bus monitors or remote terminals, depending upon their function on the data bus (Ref 6:2-13 - 2-14).

The bus controller is the terminal responsible for initiating all message transfers on the bus. Being interfaced to the host processor of the avionic system, the bus controller has access to all of the information needed to provide data flow control for the integrated system. In addition to generating all commands, the bus controller also receives and decodes the status words and identifies subsystem and bus malfunctions. The bus controller determines which channel to use in systems possessing redundant buses. The bus controller can pass the control responsibility to a backup controller if an error is detected by its self-test mechanism or if it is required to perform an alternate processing task (Ref 6:4-39).

The primary function of a bus monitor is to listen to the addresses of the messages being transferred on the bus and to extract and store certain information for later use. In this mode the bus monitor performs no bus transactions and transmits no status words to the bus controller. This is the only terminal type which need not have transmit capability; however, it is usually implemented with remote terminal capability in order to allow the bus controller to check for malfunctions (Ref 7:I-35). Bus monitors are permitted to function in an alternate mode in which they monitor all bus traffic and act upon any detected malfunctions. This mode is rarely implemented in avionic systems, however (Ref 7:I-1).

The remote terminal is defined as any active terminal which is not operating as either a bus controller or a bus monitor. A remote

terminal must transfer data between the bus and its subsystem when directed to do so by the bus controller. It is never permitted to initiate any action on the bus. Remote terminals can be designed to operate as either standalone or embedded units and they possess various degrees of intelligence. In order to enhance system flexibility, many remote terminals are designed as standalone units and are equipped with their own processors so that they are able to function as either remote terminals, bus monitors or bus controllers. These units can be used to provide backup support for the primary units in the event of system failure (Ref 7:I-35).

Remote terminals are composed of four functional elements: an analog receiver/transmitter, a bit/word processor (encoder/decoder), a message processor and subsystem interface circuitry. The minimum capabilities of each element are defined by MIL-STD-1553B. The analog transmitter/receiver and the encoder/decoder elements perform the word processing for the remote terminal. The analog transmitter/receiver is the unit which is tied directly to the 1553B bus cable. Its primary function is to interface the digital terminal logic to the bus. The unit contains the bus coupling hardware, an analog receiver and an analog transmitter. The receiver filters out low level noise and uses a combination of signal limitation and threshold detection circuits to convert the received signal to the bi-phase (TTL) form required by the subsequent units. The receiver contains a driver and controller which convert the TTL signals to the appropriate voltage levels for the bus, as well as a time-out circuit which ends transmission after a maximum of 800 microseconds. The bit/word processor unit performs all of the data

encoding and decoding functions for the remote terminal. The decoder identifies the type of word being received, performs a parity check and decodes the 16-bit word. Detection of encoding errors by the decoder results in the flagging of the appropriate error conditions. The encoder generates the synchronization pattern and the parity bit and encodes the bits of each 16-bit word it receives from the message processor (Ref 7:I-38).

The message processor and the subsystem interface circuitry are responsible for the actual processing of all of the messages being transferred by the remote terminal. A major function of the message processor unit is the performance of a detailed analysis of the 16-bit data words and the error messages received from the encoder/decoder. The analysis of the incoming messages includes command word decoding, address validation, message length validation and message error detection. The message processor is also responsible for generating the status words to be transmitted to the bus controller, controlling the direction of the data flow, buffering the incoming data and providing control signals to the subsystem interface circuitry. The subsystem interface circuitry is the only functional element which is application dependent. Since some applications require much more complex circuitry than do others, standardizing the functions and the design of the interface is not generally feasible. The greatest diversity in interface circuitry design is seen with the standalone remote terminals since each is interfaced to several different subsystems. The interface circuitry usually performs channel selection, signal conditioning, subsystem timing and subsystem calibration. The functions of the

embedded remote terminal are quite different than those of the standalone terminal. They include direct memory access (DMA) transfers to the subsystem memory, as well as interrupt and control signal generation (Ref 7:I-36 - I-39).

2.3.2 Bus Cable and Couplers. A MIL-STD-1553B data bus uses a twisted-pair shielded cable as its transmission medium. This cable permits high speed data transfer while still fulfilling the reliability, size and weight requirements levied by the aircraft operating environment. The cable/terminal interface is provided by bus couplers. Two types of coupling, direct coupling and transformer coupling, are described in MIL-STD-1553B. In direct coupling, the lines, or stubs, which run between the terminal and the cable are connected directly to the main bus cable. This method is acceptable if the impedance of the bus cable closely matches that of the stub, a situation which normally does not occur unless a stub length of less than one foot is used. Transformer coupling is used when longer stubs are required. With this type of coupling, the stub is connected to an isolation transformer and two isolation resistors before being interfaced to the main bus. Transformer coupling not only reduces distortion by appearing as a high impedance input, but also isolates and protects the main bus from any electrical malfunctions in the stub. Since isolation improves the reliability of the bus, the Air Force Notice 1 requires transformer coupling for all aircraft avionics systems (Ref 7:I-1, I-14).

2.4 Bus Implementation Concerns

The addition of a new subsystem to an existing bus system must be performed without interfering with the current data flow or degrading

the bus performance. Prior to the integration, the overall system design must be studied since each bus implementation levies a different set of system specific requirements upon a subsystem interface design. The interface design is dependent upon the bus topology, the control and error management schemes, the bus redundancy and the functional partitioning. In addition, to ensure that the addition of the new subsystem will not adversely affect the operation of the existing system, the subsystem's contribution to the bus loading must also be taken into consideration. Each of these factors is discussed below.

2.4.1 Bus Topology. The bus topology is the physical layout of the bus system. The bus system can be implemented using either a single or a multiple level topology. Figure 4 shows an example of the single level topology. This is the simplest scheme since all of the terminal units are connected to the same bus cable and a single bus controller is

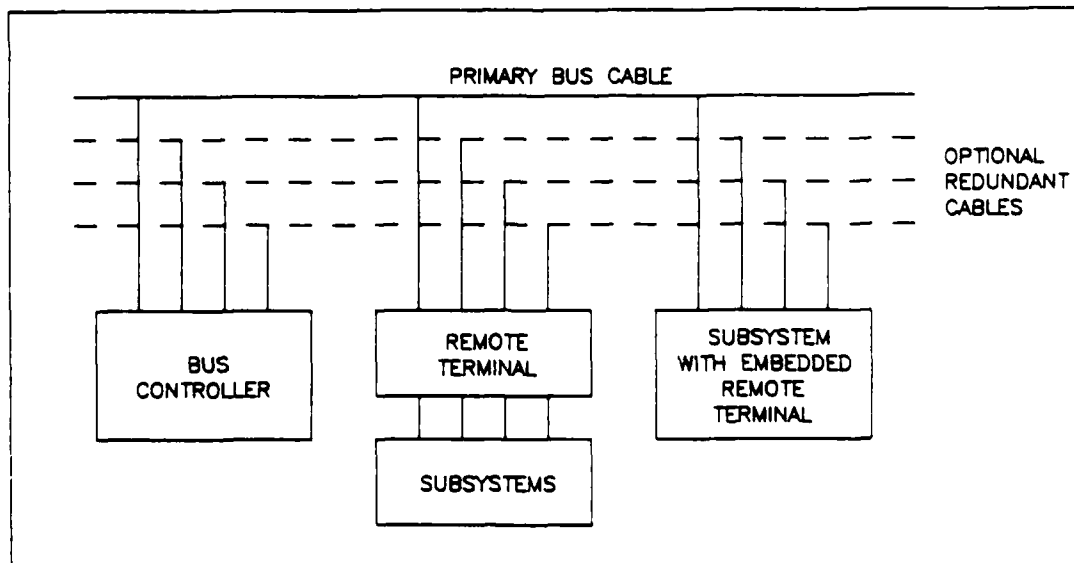


Figure 4. Single Level Bus Topology (Ref 6:7-2)

in charge of all message transfers. A maximum of 31 terminals may be interfaced to the bus. The optional redundant bus cables do not affect the level of the topology since all the terminals are connected to each cable and only one cable is in use at any given time. A multiple level topology is created by interconnecting single level bus systems in such a way that data can be passed from one bus subsystem to another. An example of this type of topology is shown in Figure 5. With a multiple level topology, control can take one of two forms: equivalent levels of control or hierarchical levels. The equivalent levels of control scheme is used most frequently since it is much more easily implemented. With this scheme, each subsystem bus operates autonomously unless data must be transferred between buses. The multiple levels of control scheme is complicated since the subordinate buses must maintain coordination with the global or superior buses at all times (Ref 7:I-25,I-26).

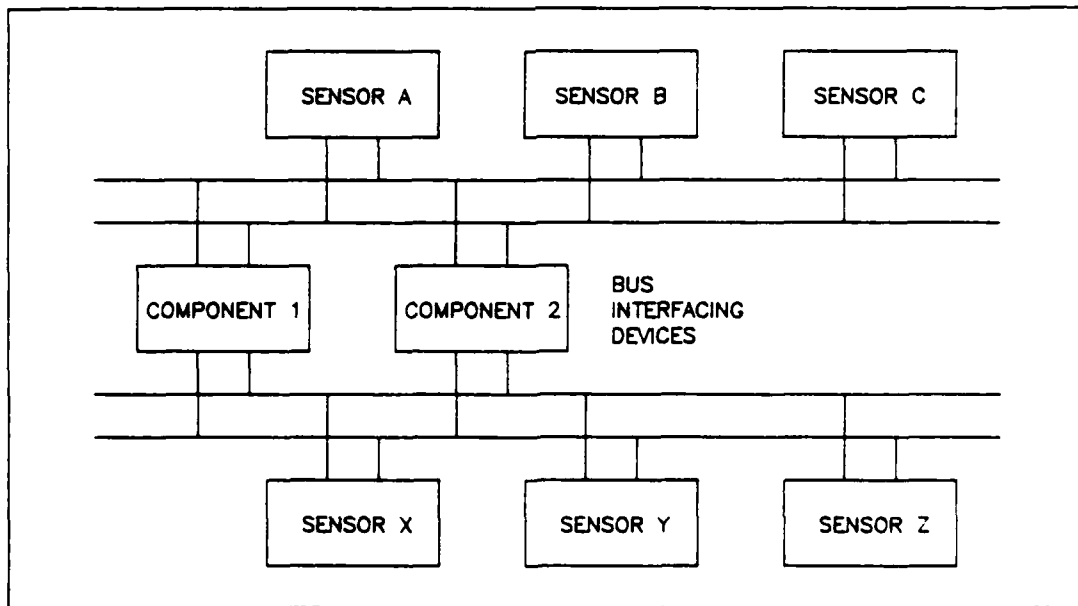


Figure 5. Multiple Level Bus Topology (Ref 7:I-26)

2.4.2 Bus Control. The bus control philosophy deals with a different aspect of control. Although MIL-STD-1553B allows only one controller to have control of the bus at any given time, it does not restrict the number of controllers which can time-share the control function. With the simplest control scheme, the stationary master approach, a single bus controller manages all of the communication on the bus. If a backup controller is to be used in the event of failure, the control is usually passed to the other through external circuitry. This scheme can be readily applied to the multiple level topology by placing both a primary and a backup controller on each bus subsystem. When the other control scheme, the non-stationary master approach, is implemented, the control function is shared by several bus controllers on a regular basis. Either a round robin or a time-based sharing mechanism is used for controller scheduling. Both of these approaches require the use of the dynamic bus control mode and are quite complicated. With a non-stationary master control scheme, the system must possess extensive error detection capabilities (Ref 7:I-26,I-27).

2.4.3 Error Management. Error management is a critical aspect of bus control. Errors fall into one of two categories, depending upon whether they are the result of a subsystem failure or a data bus/terminal failure. The primary responsibility for handling the first type of error lies within the subsystem itself; the data bus/remote terminal participates only to the extent of transmitting error notifications from the subsystem to the bus controller. Management of data bus/terminal errors is the sole responsibility of the bus system, however. MIL-STD-1553B requires that the bus hardware be able to

provide word and message validation through the identification of synchronization, Manchester encoding, word size, parity, message length, word discontinuity (within a given message) and no terminal response errors. Remote terminal status bits can optionally be set, as well. Although MIL-STD-1553B requires that the data bus system respond to all identified errors, specific techniques for error processing are determined by the bus system designer, and are, therefore, implementation dependent (Ref 7:I-27,I-28).

2.4.4 Redundancy and Functional Partitioning. Bus redundancy and functional partitioning are related aspects of the data bus implementation which must be selected in conjunction with the bus topology and the control scheme. The purpose of bus redundancy is to provide an alternate path for message transfer in the event of bus element failure. Redundant bus elements are normally the bus controller, the bus cabling and the remote terminal interface to the cable. Both an active and a backup bus controller can be used in a stationary master system. In this configuration, the master normally initiates transfer of control to the backup controller upon detection of self-test errors. Redundant cabling is used in most systems, with the bus controller selecting which path will be used for message traffic at any given time. As many redundant sets of bus cables as required to ensure acceptable operation can be used; however, dual redundancy is the norm and is required by Air Force Notice 1. Separate remote terminal units generally are used to tie each subsystem to the redundant cables so that an alternate message route to a subsystem is available if the remote terminal unit on the primary path fails. Figure 6 shows examples of several dual

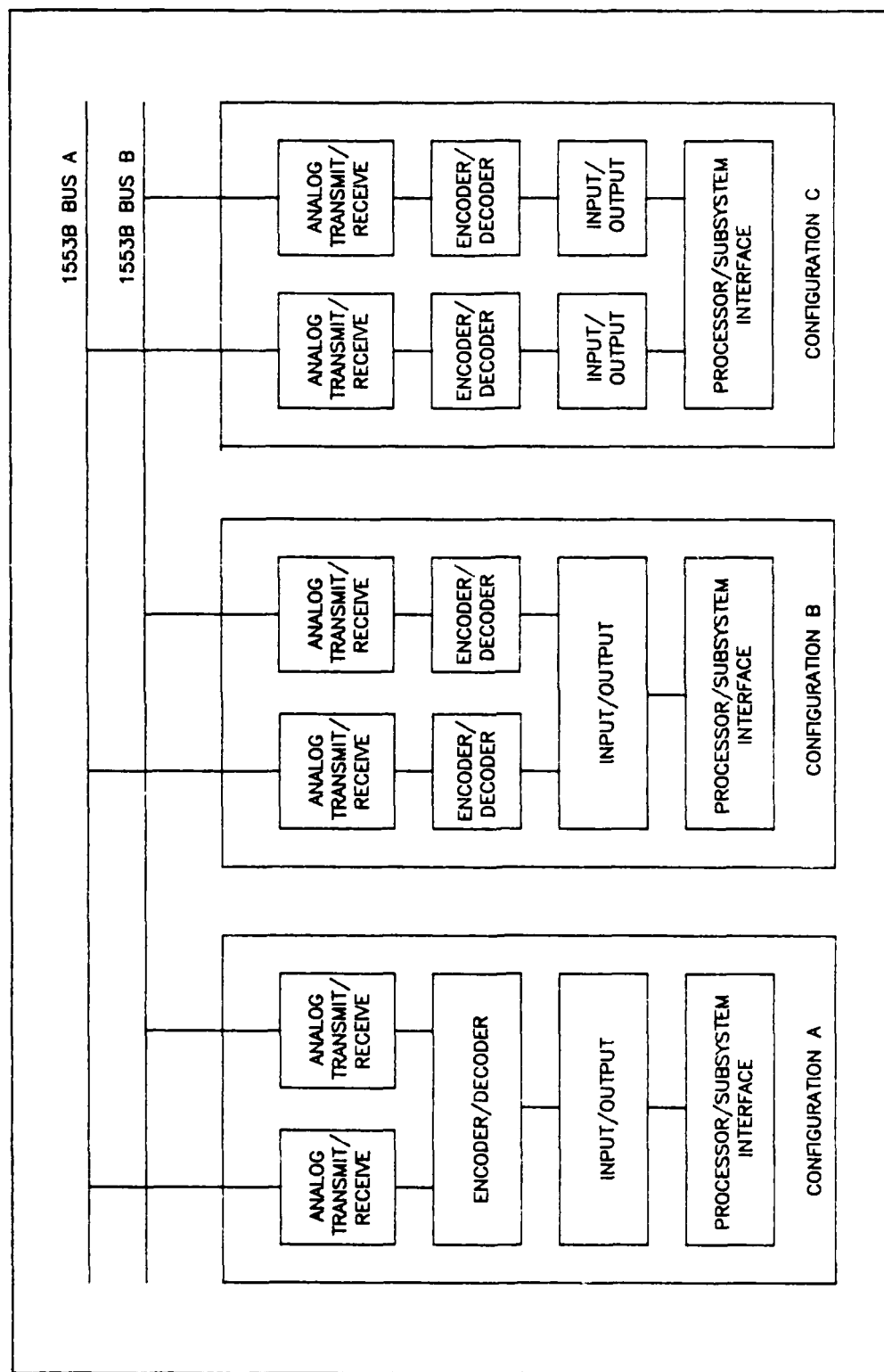


Figure 6. Dual Redundant Bus/Terminal Configurations (Ref 7:1-33)

redundant bus/remote terminal configurations. Configuration A is not acceptable for Air Force systems, since, in order for command word validation to be performed separately for each bus, redundancy must extend through the bit/word processor (encoder/decoder) portion of the terminal (Ref 7:I-32). Bus redundancy is used to ensure the availability of a data path, while functional partitioning is used as a means to optimize the data flow between subsystems. The designer uses the functional flows expected within the bus system to determine the partitioning scheme. Functional partitioning is a concern only with multiple level topologies and is used in addition to bus element redundancy (Ref 7:I-31 - I-33).

2.4.5 Bus Loading. Bus loading is an implementation dependent concern which is more a function of the amount of message traffic than the actual bus configuration aspects discussed above. During the initial design phase, and subsequently whenever a new subsystem is to be added, care must be taken to ensure that the sheer bulk of the traffic does not adversely affect the operation of the system. The simple mathematical analysis (Ref 7:I-33,I-34) requires a considerable amount of information concerning each subsystem which may not always be readily available. Luckily, since the bus protocol is quite efficient, problems usually arise only with large systems having a considerable amount of message traffic (Ref 7:I-33).

2.5 DAIS Implementation

The Digital Avionics Information System (DAIS) is a general purpose information transfer system based upon the MIL-STD-1553B data bus. The DAIS, which was developed by the Air Force Avionics Laboratory at

Wright-Patterson Air Force Base, Ohio, is used as a test bed for aircraft avionic systems. The major components of the DAIS are a set of Westinghouse AN/AYK-15 digital processors, a performance monitor interface unit, a user console and a MIL-STD-1553B data bus simulator. The AN/AYK-15 processors are general purpose processors designed for airborne embedded computer operation. The number of processors is dependent upon the requirements of the particular avionic system being tested; there is a one-to-one correspondence between the DAIS processor implementation and the actual avionic system. The processor software consists of the operational flight programs (OFPs) of the avionic system and operational test programs (OTPs). The performance monitor interface unit ties the AN/AYK-15 processors to a Digital Equipment Corporation (DEC) PDP-11 computer system which gathers system performance data. The user console is an intelligent terminal which allows flexible control of the test process. The MIL-STD-1553B bus simulator provides the means to simulate the internal command and data transfer between the AN/AYK-15 processors and any additional sensor or weapons subsystems which may be present in the avionic system. The AN/AYK processors are interfaced to the bus via special bus control interface units which meet the bus specification; the other subsystems and sensors use standard remote terminal units (Ref 6:6-61 - 6-63). The DAIS implementation of the MIL-STD-1553B bus is discussed in more detail below.

2.5.1 Bus Implementation. The DAIS implementation closely follows the MIL-STD-1553B specification. The simulator was designed to reflect the Air Force avionic system design standards and to provide support for the variety of system configurations falling within those standards.

The bus implementation possesses a single level bus topology with a stationary master control scheme. The system uses dual redundant bus cables with Channel A being a standard twisted-pair cable and Channel B being a fiber optic cable which is provided since the AN/AYK-15 processors possess a fiber optic interface. An error management scheme which uses built-in-test data is provided for the bus terminal hardware. The required format for the BIT word which must be transmitted upon receipt of the appropriate command is shown in Table 2. The BIT word contains two fields: the Terminal Failure Field (bits 0,1,4-9) which reflects the results of hardware self-test and the Message Error Field (bits 10-15). All Terminal Failure Field bits are reset by the Initiate Self-Test mode command and all but bit 0 are reset by a Reset Terminal mode command. The Message Error Field is reset with each transmit/receive command (Ref 4:25 - 26). Since the DAIS is a test bed, information concerning the particular avionic system under test is required to determine the subsystem error management scheme and the bus loading effects. Also, since use of the standard data word formats as defined in the ICD is not required, the data word formats are also dependent upon the particular system.

2.5.2 Operation Restrictions. The DAIS bus simulation complies with the Air Force Notice 1 to MIL-STD-1553B, February 1980) which limits several of the options available to MIL-STD-1553B designs. The directive prohibits the transmission of broadcast commands by the bus controllers and prohibits the use of several of the mode codes. Since the restriction is only on the use of these functions, hardware which has been designed to support them may still be incorporated into the bus

Table 2. DAIS BIT Word Format (Ref 4:25)

BIT	CONTENTS
0	Power on Reset (Set to Logic 1 on Power-On)
1	Interface Module Failure
2	Multiplex Terminal Unit 1 Out (Bus Channel A)
3	Multiplex Terminal Unit 2 Out (Bus Channel B)
4	Self-Test Failure (Terminal Hardware)
5	Self-Test Failure (Terminal Hardware)
6	Self-Test Failure (Terminal Hardware)
7	Self-Test Failure (Terminal Hardware)
8	Self-Test Failure (Terminal Hardware)
9	Self-Test Failure (Terminal Hardware)
10	No Data Received
11	Word Count High
12	Word Count Low
13	Data Parity Error
14	Invalid Data
15	Invalid Command

systems (Ref 7:I-14, I-15). The DAIS message transfer formats are the same as those shown in Figure 3. Figure 7 lists the DAIS mode codes and shows the corresponding message transfer formats.

MODE COMMAND	MODE CODE	MESSAGE FORMAT		SEE NOTES:
SYNCHRONIZE	00001	TRANSMIT COMMAND	• STATUS	1
TRANSMIT STATUS	00010	TRANSMIT COMMAND	• STATUS	2
INITIATE SELF-TEST	00011	TRANSMIT COMMAND	• STATUS	3
TRANSMITTER SHUTDOWN	00100	TRANSMIT COMMAND	• STATUS	
SHUTDOWN OVERRIDE	00101	TRANSMIT COMMAND	• STATUS	
RESET REMOTE TERMINAL	01000	TRANSMIT COMMAND	• STATUS	
TRANSMIT VECTOR WORD	10000	TRANSMIT COMMAND	• STATUS VECTOR WORD	4
SYNCHRONIZE WITH ONE DATA WORD	10001	RECEIVE COMMAND	• SYNC DATA WORD • STATUS	1
TRANSMIT LAST COMMAND	10010	TRANSMIT COMMAND	• STATUS LAST COMMAND	2,5
TRANSMIT BIT WORD	10011	TRANSMIT COMMAND	• STATUS BIT WORD	2
INVALID/RESERVED	ALL OTHER CODES	TRANSMIT OR RECEIVE COMMAND	• STATUS	6

* RESPONSE TIME, 4.0 - 12.0 MICROSECONDS

- NOTES: 1. S/R BIT SUPPRESSED, THEN RE-ENABLED FOR THIS STATUS.
2. NO UPDATING OR CLEARING OF BIT REGISTER ALLOWED.
3. TERMINAL FLAG (TF) BIT SUPPRESSED FOR THIS STATUS.
4. S/R BIT SUPPRESSED AND DISABLED FOR THIS STATUS.
5. CURRENT LAST COMMAND SHOULD NOT BE UPDATED.
6. MESSAGE ERROR (ME) BIT SET IN STATUS.

Figure 7. Message Formats for DAIS Mode Command (Ref 4:20,21)

CHAPTER 3

THE INTEL iAPX 432/600 COMPUTER SYSTEMS

3.1 Introduction

The Intel iAPX 432/600 computer systems were designed to support complex, software intensive applications which are normally handled by mainframe computer systems. Many architectural innovations were necessary in order to provide mainframe capability while still preserving the cost and size advantages of a microprocessor-based system. The 432/600 computers are based upon the iAPX 432 Micromainframe family of components. This advanced architecture gives the user 32-bit processing power, transparent multiprocessing, object-oriented addressing, a high level language instruction set and a Silicon Operating System which augments the Multifunction Applications Executive (iMAX) software operating system (Ref 8:1). In addition, the 432/600 computers systems exemplify a new approach to computer technology in which the hardware, software and methodology are integrated in the system design. Since the design of the 432/600 computer systems is so innovative, the systems are inadequately described by conventional concepts of computer architecture. Therefore, this chapter is devoted to introducing these new concepts.

3.2 General Description

The Intel iAPX 432/600 computers are modular, extensible systems. All 432/600 computer systems share the same basic system organization, shown in Figure 8. The Central System, which is composed of the

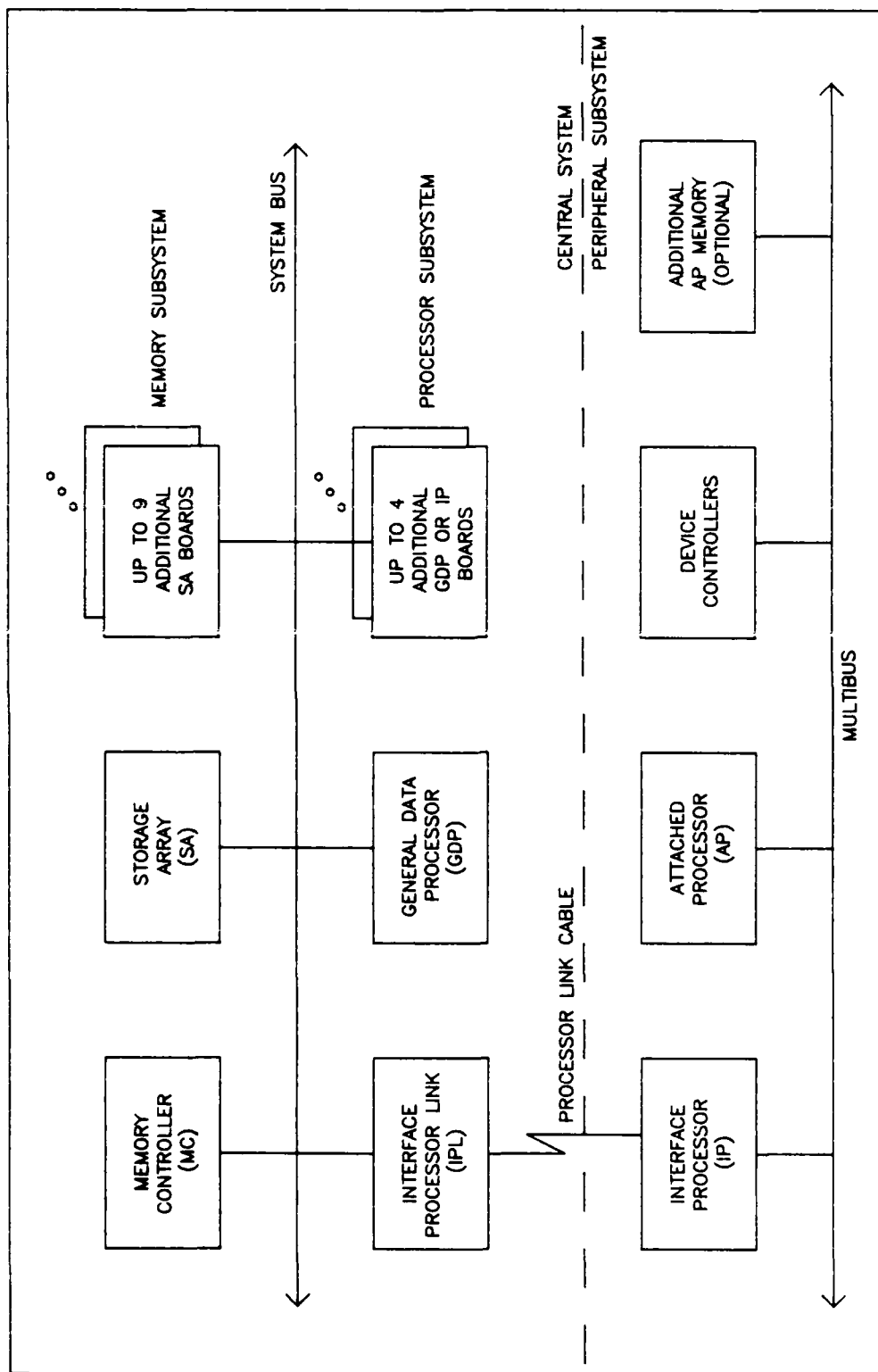


Figure 8. Basic 432/670 System Organization (Ref 8:3)

Processor Subsystem and the Memory Subsystem, handles all of the data processing tasks. The Peripheral Subsystems are responsible for performing all of the system input/output functions using the external devices. The Central System is linked to each Peripheral Subsystem through an Interface Processor Link (IPL) board located in the Central System and an Interface Processor (IP) board in the Peripheral Subsystem. This partition between the Central System and the Peripheral Subsystem serves as a protective barrier against unauthorized access by the external devices. The Peripheral Subsystems are capable of performing real-time processing for tasks in which transferring the data across this partition would introduce an unacceptable delay. A 432/600 system can support up to five independent Peripheral Subsystems. In addition, each of these subsystems, as well as the processor and the memory subsystems, is independently extensible. By varying the number of basic modules in each subsystem, varying degrees of processing performance and memory capacity can be obtained. Thus, each 432/600 system can be readily tailored to meet the requirements of a specific application. The systems can also be modified in the field as the requirements of the application change or grow (Ref 8:2).

3.2.1 Processor Subsystem. The Processor Subsystem contains a combination of General Data Processor (GDP) boards and IPL boards. The computational and input/output capability requirements of the application determine the actual subsystem configuration. At least one board of each type is needed for system operation. The remainder of the boards can be any combination of GDPs and IPLs. The total number of boards which can be supported is determined by the system hardware and

by the iMAX operating system. iMAX requires that the slots in the System Bus backplane be partitioned by subsystem type, hence each subsystem is assigned specific slots for its use. The larger backplanes have more slots dedicated to the processor subsystem; for example, the 18 slot backplane, which is the largest currently offered for the 432/600 system, will support a combination of six GDP and IPL boards (Ref 8:4).

The GDP boards handle the processing for all of the 432/600 system's computational tasks. Each GDP board contains a 43201/43202 chip-pair and its own clock generator. The separate clock generators provide independent internal sequencing for each GDP. Thus, each processor in the subsystem runs at its own clock rate and is not affected by the rates of any other GDP or the rate of the Memory Subsystem. Intel's Silicon Operating System permits each GDP to perform its own process scheduling. When a GDP becomes idle, it will start executing the next process waiting in the system's ready queue. When that process is completed, enters an input/output wait state or times out, the GDP automatically reschedules the process and begins executing the next process. These two hardware features allow the system to provide software-transparent multiprocessing capability. If iMAX is initially configured to handle the maximum number of GDPs which might reside in a particular system at any given time, GDP boards can then be added or removed from the system without affecting the system software, application software or the memory access arbitration logic. The only affect that changing the number of GDP boards has upon the system is the change in processing performance. The GDPs communicate with each

other, with the 43203 Interface Processors (IPs) and with the system memory. A round-robin arbitration scheme is used to prevent conflicts and to give all of the GDPs and IPs an equal opportunity to access memory. A processor becomes Bus Master when its turn comes up, then, when its memory access is complete, it is assigned the lowest priority. The logic for the arbitration scheme, which is stored in Programmable Read-Only Memory (PROM) on each processor board, provides software-transparent arbitration for up to six processors without the aid of special hardware. The IPLs serve as the representatives of the IPs on the System Bus. The IPLs link the Peripheral Subsystems to the Central System and handle the bus arbitration for their companion IPs. The IPs, which reside on separate busses, contain their own clock generators as do the GDPs. Each IPL operates at the clock rate of its companion IP unless it is communicating with the GDPs or with the Memory Subsystem, in which case it uses the System Bus clock rate (Ref 9:1-2 - 1-6).

3.2.2 Memory Subsystem. The Memory Subsystem, which resides on the System Bus with the Processor Subsystem, is composed of a Memory Controller (MC) board and between one and ten Storage Array (SA) boards. The MC board, which has a dedicated slot in the System Bus, has two major functions. Using its 8 MHz clock and two sets of control lines, it controls and synchronizes all of the communication of the System Bus. The MC also exercises control over all of the memory accesses. It services all GDP and IP requests for memory, translates the physical addresses to determine which SA board to access, supports interprocessor communication and local register accesses and maintains a centralized hardware error log. The controller logic supports byte addressability,

providing one, two, four, six, eight or ten byte increments in response to the processor read requests. The MC can control up to 16 SA boards; however, the largest backplane currently available for the 432/600 actually allows for only 10 boards. The MC can also support memory interleaving, if the system contains an even number of SA boards. The SA boards each hold 256K bytes of memory with Error Correction Code (ECC) protection. The individual SA boards contain the hardware required to generate, check and store a 7-bit ECC for each 32-bit word stored. For each data word read from memory, single bit errors can be detected and corrected and double bit errors can be detected. All uncorrected errors are logged in a system error register. As with the Processor Subsystem, the capacity of the existing system can be modified by merely adding or removing SA boards from the System Bus (Ref 9:1-6, 1-7).

3.2.3 System Bus. Instead of defining a standard bus structure, the iAPX 432 defines a communications protocol for all 432-based systems. This approach allows system designers to select the type of interconnect structure best suited to their application. The 432 communications protocol was designed to minimize bus traffic and to exploit the bus width regardless of the actual bus implementation. The protocol defines a variable length packet communication system for both processor to memory and interprocessor communications. The processors communicate with memory by transmitting request packets to the MC. Since reply packets are generated only for memory read requests and since the variable packet length decreases the number of memory accesses required, bus usage is minimized. The interprocessor communications

protocol allows any processor to direct a message to any another processor or to broadcast a message to a number of processors at once. Actually, the messages are deposited in a section of memory reserved for interprocessor messages, then the receiving processors are signaled so that they know they have messages waiting. Interprocessor messages usually contain start, stop or redispach commands (Ref 10:120). The System Bus in which the 432/600 Central System resides is an implementation of this communications protocol. The System Bus contains a 36-bit Specification/Address/Data (SAD) bus, an Arbitration Bus and various control and error lines. The SAD bus has 32 lines reserved for data and addresses and 4 lines for parity bits, one for each byte. The Arbitration Bus consists of a group of lines which are dedicated to memory access arbitration. There are two sets of control lines, one for the GDP and IPL boards and another for the SA boards. All communications on the bus are synchronous and are completely controlled by the MC (Ref 9:3-3).

3.2.4 Peripheral Subsystems. The Peripheral Subsystems function as the input/output processors of the 432/600 system, relieving the Central System of the responsibility for low level device control and data transfers. A Peripheral Subsystem contains an IP board, an Attached Processor (AP) board, peripheral device controllers and any additional AP memory. Each subsystem is mounted in a separate Multibus backplane and is interfaced to the Central System through a Processor Link cable which connects the companion IP and IPL boards (Ref 9:1-2). Any Multibus compatible components can be used as APs, device controllers, direct memory access controllers and communication

controllers. The Central System treats each AP/IP/IPL combination as a logical input/output processor.

An AP is an autonomous single board computer, usually based on a 16-bit processor, which is responsible for managing the Peripheral Subsystem. The AP coordinates the input/output processes and supervises all data transfers. It polls devices, handles interrupts generated by the device controllers and sets up and monitors DMA transfers. The AP also controls the IP, directing it to set up the communications link between the subsystem and the 432 Central System. All data transfers across this link are under the direct control of the AP. The IP merely executes the instructions as they are received, sending an interrupt to the AP at the completion of each command. The AP can also function as an independent processor, handling real-time data processing tasks (Ref 8:5). In order to adequately perform all of these required tasks, the AP runs under its own real-time multitasking operating system. The AP software can be written in any language or combination of languages which permits it to perform its functions within the subsystem's time constraints. Since each Peripheral Subsystem is independent of the Central System and the other Peripheral Subsystems, its software and operating system can be readily configured to give optimal performance.

The Peripheral Subsystems are linked to the Central System via their IP/IPL board pairs. The IP/IPL pair supplies both a physical and logical interconnection between the System Bus and the AP in the subsystem. The IP acts as a slave to the AP, sequentially executing the instructions it receives from the AP. The IP expands the AP's instruction set, supplying the iAPX 432 object-oriented instructions

which the AP needs in order to manipulate data within the Central System's address space. The IP/IPL pair gives the AP access to this address space through a software controlled group of programmable associative memories called window registers. Through these windows, the IP maps a portion of the AP's address space into the 432's. The mapping process is totally transparent, so the GDPs and the AP both handle normal intersystem communications as though they were ordinary memory reads and writes (Ref 10:120). During normal operation, information is actually transferred through the IP window in the form of dynamic data structures called objects. Object-oriented addressing and transfer occurs whenever the IP is in its logical reference mode. The IP can also operate in a physical reference mode, in which the 432's protection mechanisms can be by-passed. In this mode, segments of memory can be mapped directly from AP memory to 432 memory. Using this mechanism, the AP can load the images of the required system objects into the 432's memory to initialize the Central System. Thus, the 432/600 computer system could not operate without at least one Peripheral Subsystem (Ref 11:1-22).

3.3 iAPX 432 Architecture

The major components of the iAPX 432 Micromainframe family are the 43201/43202 GDP chip pair and the 43203 IP chip. All three are Very Large Scale Integration (VLSI) devices which are fabricated using a +5 volt, depletion load, N-channel HMOS technology (Ref 7:1). Each chip is housed in a 64-pin Quad In-Line Package and dissipates less than 2.5 watts of power. The iAPX 432 chips are very high density; the GDP pair contains 160,000 transistors and the IP has 65,000. The complex designs

were created using advanced design techniques such as regular logic structure and wiring topology. The 432 chips are microprogrammed. The GDP's microprogram resides in a 4K by 16-bit ROM; the IP uses a 2K by 16-bit ROM. The size of the IP microprogram ROM is minimized by storing two bits in each ROM cell and by using high-performance microarchitecture. The microarchitecture is so powerful that the GDP instruction set occupies only six percent of the total microprogram. The Silicon Operating System, which handles a number of high level, time critical, complex and security sensitive operating system functions, is implemented in a combination of microcode and hardware on each chip. Dual mode operation is another important feature of all the 432 components. In the master mode, the chip functions normally. In the checker mode, however, instead of placing data on the output pins, the chip samples these pins and compares the states of these lines with the signals which would have been output. Any mismatch generates an error signal. By wiring together two identical components which are operating in different modes, a fault-sensitive device can be built. The operation of the two chips is perfectly synchronized, so that an error by either chip is detected during the checking process and a signal which halts both chips is generated. Highly fault sensitive systems can be built by replacing the individual components by fault sensitive units (Ref 10:121,122). In addition to the features mentioned above which allow microprocessor packaging of a powerful and reliable 32-bit processor, the 432 Micromainframe family possesses a number of specialized architectural features which permit a high degree of system performance to be attained at a relatively low cost.

The primary goal of the innovative 432 architecture is to increase system performance while decreasing cost. The major differences between the innovative architecture of the 432 Micromainframe family and conventional computer architectures are listed in Table 3. The implementation of these features results in a much improved performance to cost ratio. Since software costs far exceed hardware costs for conventional computers, many standard system support functions were moved into hardware and a language-directed architecture was implemented. The Silicon Operating System provides hardware support for many of the operating system functions including process scheduling, interprocess communication and dynamic storage allocation. Powerful fault-handling and multiprocessing mechanisms are also incorporated in the hardware. The language-directed 432 architecture, unlike that of most conventional systems, supports today's high level languages and programming environments. The 432 is designed to be programmed exclusively in a high level language like Ada. The completely symmetric instruction set contains a full set of operators for all data types ranging from bits to long (80-bit) floating point numbers and an identical set of addressing modes for all operands. Many high level instructions are directly implemented in the instruction set, as well. With such features, compiler writing and data manipulation is simplified and a more compact code can be generated. The 432's architecture implements an object-oriented methodology which is based upon abstract data types and protection domains. Physically, the memory is divided into over 16 million individual hardware protected segments. These segments are logically organized into complex structures called objects,

Table 3. A Comparison of the iAPX 432 Architecture and Conventional Mainframe Architectures (Ref 12:1-7)

ARCHITECTURE FEATURE	CONVENTIONAL MAINFRAME ARCHITECTURE	iAPX 432 ARCHITECTURE
MEMORY ORGANIZATION		
Organization, Size	Linear 2**24 - 2**32 Bytes	Structured Segmented: 2**24 Segments, 2**16 Byte Displacement 2**40 Byte Virtual Address Space
Logical to Physical Address Translation	Single-Level Map Page-Based Relocation and Virtual Memory	Two-Level Map Segment-Based Relocation and Virtual Memory
Protected Memory Unit	Fixed-Size Page	Individual Program Module or Data Structure
DATA MANIPULATION		
Expression Evaluation	General Register	Stack or Memory-to-Stack
Primitive Data Types	Characters, Unsigned Integers, Integers, Reals	Characters, Unsigned Integers, Integers, Reals, Temporary Reals
Floating Point Hardware	Yes	Yes
Addressing Modes	Some Modes Not Available for All Operands	Symmetrical: All Modes Available for Every Operand
PROGRAMMING ENVIRONMENT SUPPORT		
Operating System	No Multi-Purpose Support No Support for Dynamic Storage Allocation Very Limited or No Multi-Processor Operation	Multi-Process Mechanisms in Hardware Dynamic Storage Allocation Mechanisms in Hardware Software-Transparent, Multi-Processor Operation
High Level Language	Assembly Language-Oriented Instruction Set	Oriented Toward High Level Languages
Programming Methodology	No Support	Object-Based Architecture

which need not be stored in physically contiguous segments. This logical organization provides better support for the program structures than do the typical linear memory organizations. In addition, the manipulation of data structures is simplified to such an extent that moving the operating system functions into hardware, where they can be handled quickly and securely, becomes practical (Ref 11:1-5 - 1-7).

3.3.1 Object-Oriented Architecture. Object-oriented architecture is a distinct style of architecture in which the concepts of data abstraction, domain-based protection and high level system functionality are integrated. Data abstraction is a successor to the top-down, structured programming methodology. In data abstraction, a program is designed around an appropriate set of encapsulated data structures called objects. The procedure/message interface to each of these objects is the same so that it is the characteristics of the objects, rather than their implementation that are of primary concern to the programmer. Software designed using data abstraction methodology is inherently more reliable and possesses a reduced life cycle cost. The concepts of domain-based protection include the creation of independent address spaces, or domains, for subsystems, the decomposition of the operating system into a set of abstractions and the use of capability-based addressing and access control in which the program structures themselves contain information specifying legal operations and access types. Integrating these concepts with those of data abstraction and implementing a number of high level system functions directly resulted in the creation of an object-based architecture which possesses the following characteristics:

1. Objects for system management and control, program environments, flow of control and intercommunication are provided,
2. Facilities for the support of type management for system objects exist,
3. User object extension, in which the user can define and create his own objects, is supported,
4. Object addressing and protection mechanisms, as well as an instruction interface, are provided.

These characteristics provide the framework for the architecture of the iAPX 432 (Ref 15:4-6).

The majority of the functions of the 432 are implemented using objects. These objects, whose structures range from relatively simple to quite complex depending upon their type, are used in resource management, interprocess communication, basic computation, multitasking and multiprocessing. They determine the language run-time environment and allow for capability, or protected, addressing. All programs which run on 432 processors consist of a series of objects and are, in fact, objects themselves.

3.3.1.1 GDP Program Structure. A program which runs on a 432 GDP consists of five types of objects: processor objects, process objects, context objects, instruction objects and data objects. The basic GDP program structure is shown in Figure 9. Each physical processor possesses its own processor object which is stored in system memory. The processor object contains processor state information (running, halted, etc.), diagnostics and machine check information, references (pointers) to other objects and a reference to the process it is

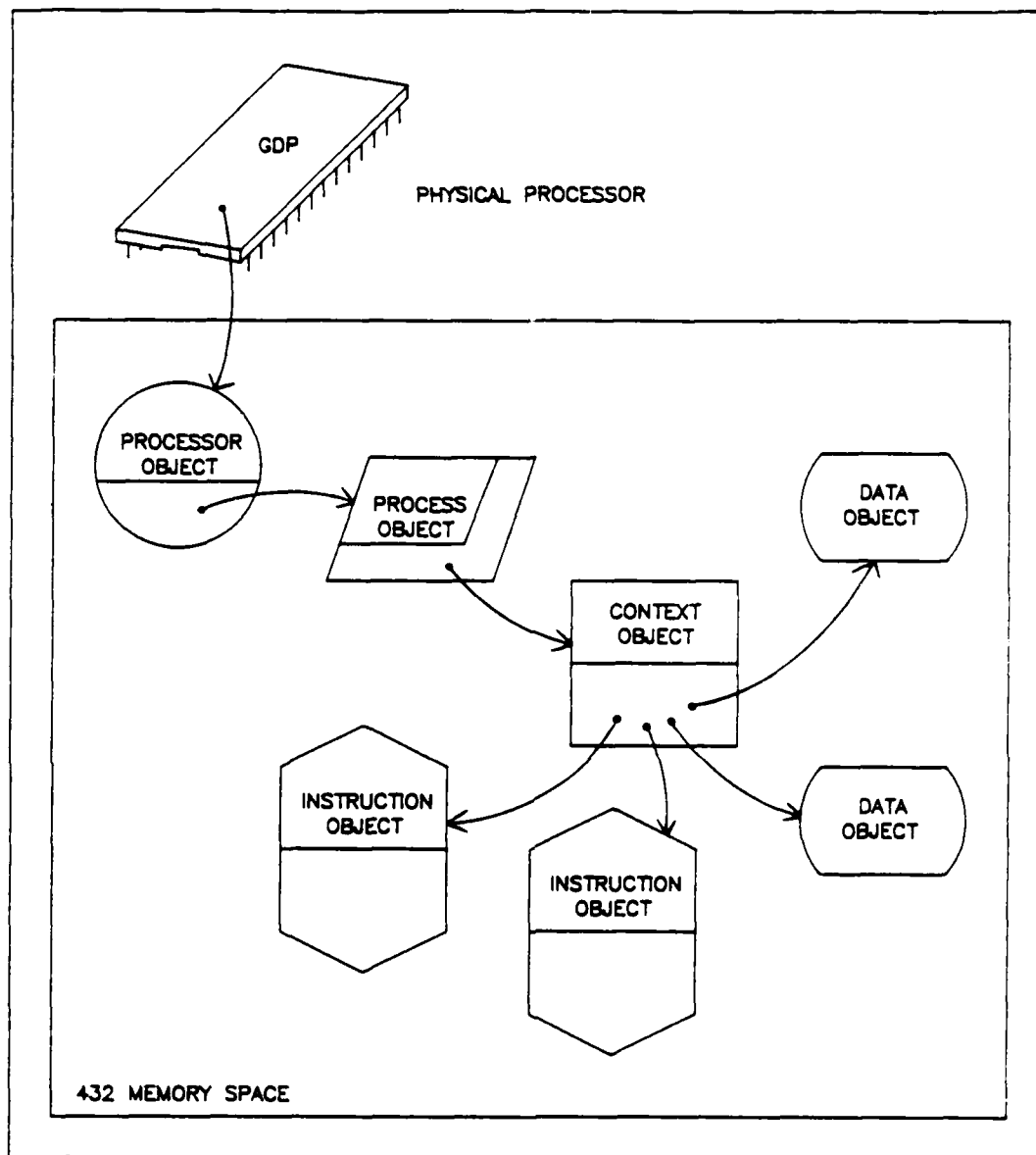


Figure 9. GDP Program Structure (Ref 16:3-19)

currently executing. The references are crucial since the object-oriented addressing and protection mechanism only allows a processor to access an object for which it has an object reference. The references

can change dynamically during program execution; for example, the process reference changes when the processor switches between different processes. The physical processor addresses its processor object using an on-chip reference (Ref 16:3-1 - 3-4).

The second type of object contained in a GDP program is a process object. Every program contains at least one process or activity which must be scheduled to run on a processor and there is one process object for each process in the system. The process object contains information on the status of the process, any requirements for concurrent scheduling or scheduling after the completion of another process and references to the context which is currently being executed. Each process consists of one or more procedures. A procedure is a set of instructions logically grouped together in order to perform a specific operation. As in conventional computer systems, the 432's procedures are called, passed input parameters and may return results when they return to the calling routine. Each procedure is represented by the third type of object, the context object. Since the same procedure may be called by several different processes, a copy or instance of the variable part of the procedure is created for each. An part of the procedure which is not modified during its execution can be shared by other processes. Each instance is assigned its own context object which contains its instruction pointer, the stack pointer for its stack, the return link to the calling context and the references to the objects which it is allowed to access. The list of accessible objects for a context is called its access environment. Any instance of a procedure may be called with different input parameters, so each context must have a

separate access environment. The protection mechanism does not allow the access environment of a procedure to be passed on when it calls another procedure. This feature provides a greater degree of security than is available in most conventional systems. A context object is able to access other context objects, instruction objects and data objects (Ref 16:3-5 - 3-17).

All of the required instructions and data for a program are stored in the instruction and data objects. The instruction objects are the only objects which contain instructions and they contain only instructions. This isolation has two major advantages. The system reliability is improved since, due to the type checking which is enforced by the protection mechanism, only instruction objects and hence, instructions, can be executed. In addition, all 432 programs are re-entrant since the instructions are not modified during execution. The data objects contain only data. The type of the data object is software defined. The data object is the only type of GDP program object which cannot be recognized and directly manipulated by the hardware (Ref 16:3-18,3-19).

3.3.1.2 IP Program Structure. A 432 Interface Processor (IP) program possesses a structure quite similar to that of the GDP, as shown in Figure 10. Structural compatibility is desirable since a standard processor interface and communications protocol can then be used for both IPs and GDPs. Most of the differences can be attributed to the fact that the IP functions as a slave to the Attached Processor (AP), receiving and executing instructions from the AP one at a time. An IP does not fetch its own instructions, so no instruction object is

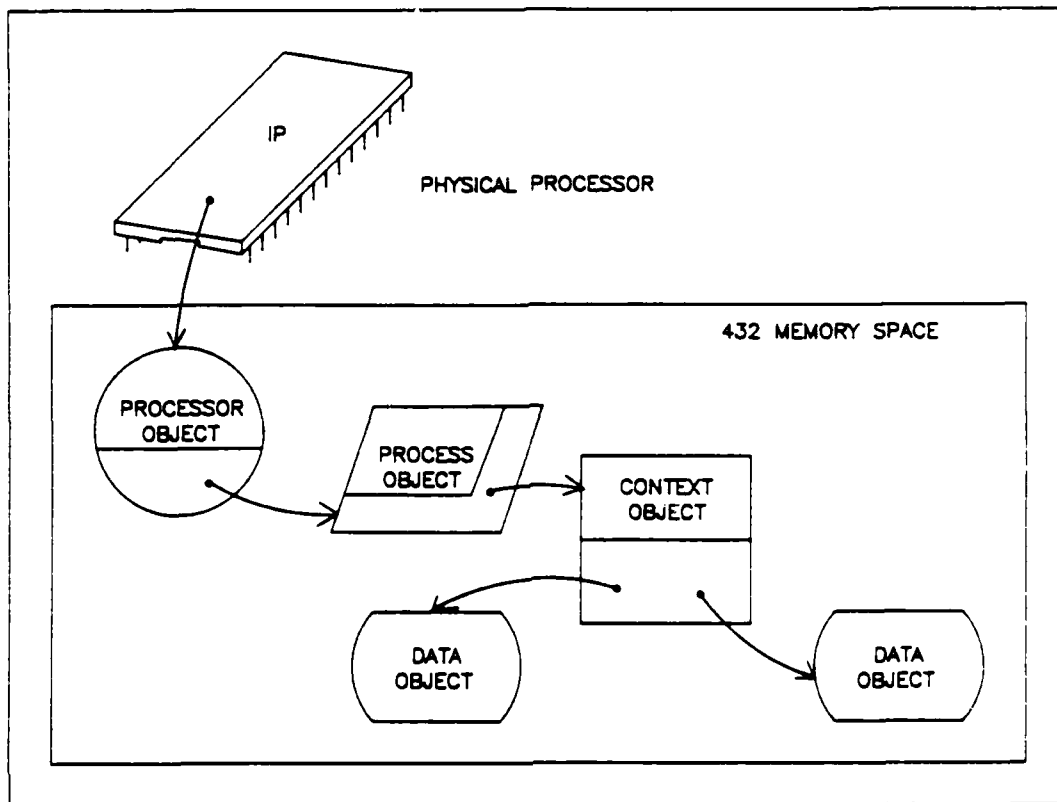


Figure 10. IP Program Structure (Ref 16:7-7)

required. An IP program structure is much more static than that of a GDP because an IP is not autonomous and possesses no flow of control. Since processes are switched only under AP control, process switching occurs much less frequently. In addition, there is no context switching within an IP process since context switching arises only when the flow of control within a process is passed from one procedure to another. Contexts are implicitly switched when the processes are switched. Program compatibility constraints require the process and procedure to be maintained within separate objects despite the fact that there is only one context object per process (Ref 16:7-7 - 7-9).

The IP's processor, process and context objects contain much of the same type of information as do their respective GDP counterparts. Except for some processor specific information, the processor and process objects are nearly identical. The context objects are the least alike. Instead of a reference to an instruction object, the IP possesses a function request area where the AP can write an instruction for the IP to execute and a status information area which the AP can read to determine if the execution was successful. IP objects are protected in the same manner as the GDP objects, with the IP being permitted access only to those objects for which it holds an object reference. The IP's access environment is extended to the AP whenever the AP uses the logical reference mode to access the 432 memory through the IP windows. The AP can disable the protection mechanisms of the 432/600 by putting the IP in the physical reference mode. Entry into this privileged mode is tightly controlled; an AP can only place its IP in the physical reference mode when the IP context object has an object reference for the IP's processor object. The physical reference mode is used only for 432/600 system initialization and diagnostic support. The logical reference mode is used for normal interprocessor communication (Ref 16:7-7 - 7-19).

3.3.1.3 Interprocessor Communication. A standard processor interface and a hardware supported interprocessor communication system are the keys to the 432/600's software-transparent multiprocessing capability. Three types of hardware-defined objects support multiprocessing. The processor objects, which were discussed previously, contain the processor specific information and can be

located anywhere in memory so that interprocessor memory contention does not occur. The second type of object is the communication port. These objects provide support for communication and synchronization between processes, allowing multiple asynchronous processes belonging to the same program to time-share a single processor or to execute in parallel on separate processors. The communication ports serve as first-in-first-out message buffers for the concurrent processes. The messages, which can be of any object type, are manipulated by the SEND and RECEIVE operators. Of course, it is only the references to the objects and not the actual objects that are moved. The third type of object, the dispatching port, provides the processor management facilities which allow any 432 software to run on a system with either a single or multiple processors. There are three elements in the 432 processor management scheme. The scheduling policy is implemented in software. Policy decisions are made by the operating system software which then sets the scheduling parameters of the process object. The software then hands the process to hardware for scheduling and dispatching. The process object is sent as a message to the appropriate dispatching port using the SEND operator. The dispatching port reference in the process object determines which port is to be used. The dispatching ports serve as the process queues for the processors. A process's priority in the queue is determined by its scheduling parameters and the parameters of other waiting processes. When a processor needs a process to execute it goes to the dispatching port referenced in its processor object. Generally, all processors of the same type share a port, so the highest priority process needing that type of processor is always executed by

the first available processor. If the dispatching port queue is empty when a processor arrives, the processor waits at the port for a process to become available. A process will execute on the processor until it times out, is forced to wait for a message from a communication port or is completed. The processor returns unfinished processes to its dispatching port. Since no instructions such as schedule or dispatch are ever required, the entire processor management scheme is software-transparent (Ref 16:4-1 - 5-25).

The 432's object-oriented architecture is much too complex to be discussed in depth in this study. The highlights of the architecture have been presented in order to introduce the reader to these innovative concepts. The 432's object-oriented architecture is discussed in much greater detail in References 16 and 17.

3.3.2 General Data Processor Architecture. The iAPX 432 derives much of its power from its innovative object-oriented architecture. In addition, the individual components of the family have all been designed with unique architectural features which allow the 432 to independently maximize the performance of the computational and input/output functions. The GDP, which is responsible for the system's computational functions, does not support interrupts or service requests, nor does it possess user-visible registers as do conventional microprocessors. Instead, the GDP is logically organized as a three-stage microprogram-controlled pipeline, as shown in Figure 11. The first two stages of the pipeline, the Instruction Decoder and the Microinstruction Sequencer, are located on the 43201 chip which is designated as the Instruction Decoding Unit (IDU). The third stage, the Execution Unit, is on the

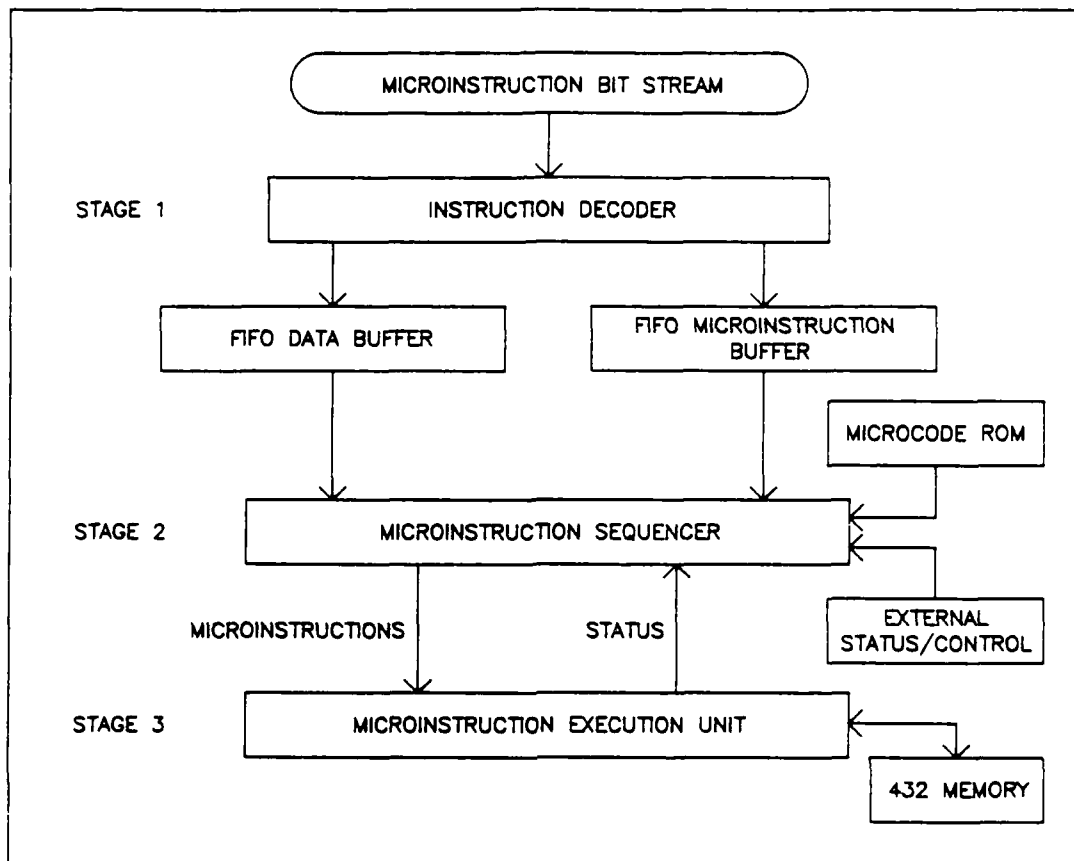


Figure 11. The Three-Stage GDP Pipeline (Ref 18:532)

43202 Microinstruction Execution Unit (MEU). Each stage, which operates independently, was designed to help simplify the decoding the rich, complex iAPX 432 instruction set (Ref 18:530,531).

3.3.2.1 Instruction Coding. The instruction coding scheme for the 432's instruction set was designed to minimize the amount of memory occupied by the instructions and still allow efficient encoding/decoding operations. The first requirement was met by using variable length, bit-aligned encoding. Filler bits, which decrease the storage efficiency of schemes which use byte or word alignment, are not necessary. The

instruction format was selected to simplify decoding. Each instruction is organized into an ordered set of variable length fields: the class field, the format field, the reference field and the opcode field. The class field specifies the number of required operands and the data type of each. Since up to three operands may be referenced by a single instruction and since some of these may be implicitly referenced by the instruction, a means of associating the explicit references with their operands is needed. Thus, the format field, which specifies this mapping, is required whenever the class field contents indicate that the instruction requires operands. The reference field is necessary when the format field mapping indicates that explicit references are expected. This field is partitioned into subfields, with one being reserved for each explicitly referenced operand. The last field of the instruction is the opcode field which specifies what operation is to be performed. Dependencies between the fields of an instruction are quite often encountered, in particular, the interpretation of the opcode field often depends upon the contents of the class field. For this reason, the fields of a particular instruction must be arranged in an ordered fashion to allow efficient one-pass decoding of the instruction (Ref 18:531, 532).

3.3.2.2 The GDP Pipeline. The first stage of the GDP pipeline, the Instruction Decoder (ID), is responsible for the actual decoding of the instruction bit stream. Two major tasks are involved: interpreting the macro-instruction stream to determine which sequence of micro-instructions to initiate and extracting the logical addresses of the operands. Since the 432 instructions and their internal fields are

variable in length, the ID must interpret the first few bits of an incoming instruction to determine how many additional words to read from memory, which fields the instruction contains and the length of each field. Since there may be dependencies among some of the fields, enough information to enable proper interpretation of the dependent fields must be stored. The ID also stores all of the information necessary for recovery from instruction faults until the execution of the instruction is completed. After all of the fields of an instruction have been read and processed, the ID issues the microinstructions for executing the instruction or, if the instruction is complex, transfers control to the Microinstruction Sequencer (MS), then proceeds to decode the next instruction (Ref 7:2,3).

The MS is the second stage in the pipeline. The primary function of the MS is to decide which microinstruction is to be sent to the Microinstruction Execution Unit (MEU) during a given machine cycle. Although the majority of the microinstructions are generated by the ID, instructions can also be obtained from the 4K by 16 microcode ROM or a small constant ROM. A priority scheme based upon the status of the GDP determines the source of the next microinstruction to be transferred to the MEU for execution. The GDP's external status is always checked at the beginning of a machine cycle. If a GDP initialization is required or if a fault from the MEU is detected, the MS immediately jumps to the proper microcode sequence for those states. If none of the external status lines have been asserted, the internal status is checked to see if the ID has requested a macroinstruction fetch. If a fetch is required, the MS stores any microinstructions which are awaiting

execution, then steals a cycle from the currently executing microcode sequence in order to issue the fetch microinstruction. In the absence of both internal and external control signals, the next instruction in the current microcode sequence is selected for execution. When the execution of that sequence is complete, the next microinstructions from the ID are transferred to the MEU and the next microcode sequence is initiated. If the pipeline is empty, the MS enters an idle state, issuing no operation microinstructions to the MEU (Ref 18:533).

The MEU, the final stage in the GDP pipeline, transfers the microinstructions received from the MS to one of its two subprocessors for execution. The two subprocessors, called the Data Manipulation Unit (DMU) and the Reference Generation Unit (RGU), operate independently in order to enhance overall system efficiency. The subprocessors are generally able to execute a microinstruction in one machine cycle. Each subprocessor has its own internal sequencer which is used to control the execution of the more complex microinstructions requiring multiple cycles. The DMU contains a set of operand registers and arithmetic functional units which enable the hardware to recognize nine data types, to perform 16- and 32-bit multiplication, division and modulus calculation and to control the 32-, 64- and 80-bit floating point arithmetic functions. The RGU translates the 32-bit logical addresses into 24-bit physical addresses, provides domain-based protection for all objects referenced, performs the sequencing for the variable length memory accesses and controls the internal top-of-stack pointer register (Ref 7:3,4). Since all of the objects used by the 432 are stored in main memory, and all information is stored in the form of objects, the

object-based addressing mechanism must be both efficient and universally applicable.

3.3.2.3 Addressing Mechanism. The 432 possesses a structured segmented memory with a virtual address space of 2^{40} bytes. The system can support up to 2^{24} (16 million) segments each of which has a maximum length of 2^{16} (64K) bytes. The virtual address space is shared by the different programs with each being assigned an access environment of 2^{32} bytes. By dynamically altering its access environment, a program can potentially address any location in the virtual address space. A two-step mapping process is used to generate the physical address of a reference from the logical address, as shown in Figure 12. Each logical address consists of two 16-bit components, the segment selector and the displacement. The segment selector is used in identifying the base address of the object containing the referenced operand. First, the access descriptor for the object is selected from one of the four access objects, each of which holds 16K access descriptors. This access descriptor contains the access rights information which is checked by the domain-protection mechanism during each memory access, as well as two indices which designate the object descriptor. One of these indices references the object table which contains the object descriptor. The other indicates which of the 4K entries is the desired object descriptor. The object descriptor contains the 24-bit base address and the length of the object. This base address is added to the displacement, giving the physical address of the operand within the object. The length data is simultaneously added to the base address and the result is compared to the physical address

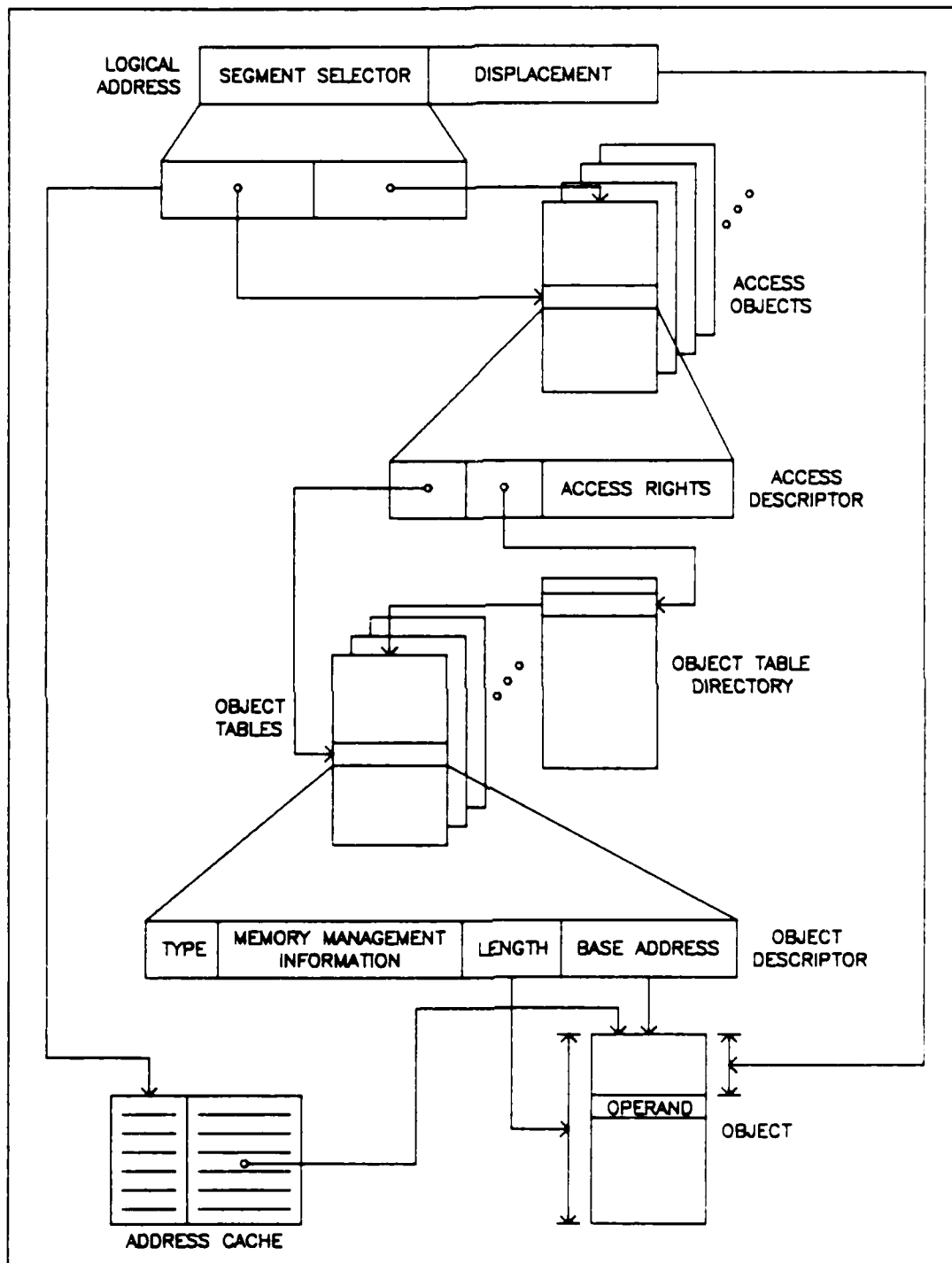


Figure 12. Logical to Physical Address Mapping Process (Ref 17:KEY-4)

to ensure that the memory reference is within the correct segment. If the reference is outside of the segment, an address exception is issued to the Instruction Decoding Unit (IDU). Together, the base address and the displacement create the virtual address space of 2^{40} bytes (Ref 17:KEY-1 - KEY-4).

The efficiency of the Reference Generation Unit (RGU) is greatly enhanced by the addition of associative cache memories. The cache memories store a set of calculated physical base addresses and the associated object lengths. Each time a logical reference to a memory segment is translated to its physical address, the entries of the cache are updated using a least-recently-used algorithm (Ref 7:4). Then, whenever an object is referenced, its logical address is compared to the addresses stored in the cache. If there is a match, the physical address is rapidly calculated using the information for that cache entry and the slower translation process is by-passed (Ref 19:517). This object-based addressing mechanism is also implemented within the IP since, as discussed previously, the GDP and IP structures must be compatible if a standard processor interface and communications protocol are to be used.

3.3.3 Interface Processor Architecture. The unique architecture of the iAPX 43203 Interface Processor (IP) chip allows an efficient interface to be maintained between the 432/600 Central System and each Peripheral Subsystem. The 43203 contains two independent functional units, the data acquisition unit (DAU) and the microinstruction unit (MEU). The AP controls the operation of both of the units. The AP requests service from the IP by writing the operands and the opcode of a

432 macro-operator into the IP's control object which is stored in the 432's main memory. Access to the control object is obtained through the DAU's control window. An access to the special opcode displacement location in the control window, which is detected by the DAU's control logic, is a signal that the writing is complete. The DAU then sends an interrupt to the MEU to initiate execution. Upon receipt of an interrupt, the MEU jumps to a service routine and subsequently reads the contents of the control object, executes the opcode and writes its status back into the control object. The status may be read by the AP, again through the control window, to determine if the desired function has been successfully completed. The DAU and MEU each contain two unique circuits which permit these complex functions to be performed efficiently (Ref 20:522).

The DAU is composed of an address mapping unit and a data transfer unit whose primary elements are a special static address mapping content addressable memory (CAM) and a byte packing buffer, respectively. These two units provide the means through which the AP accesses and manipulates the objects which are stored in the 432's main memory. In order to maximize the data transfer rate between the AP and main memory and still keep chip size at a minimum, the DAU was designed to provide four windows for high-speed data transfer and a fifth window for control and status information transfers. The address mapping unit calculates the physical address of a 432 object being accessed by the AP by adding the address displacement within the window of interest (the subsystem address) to the base address of the associated object in the 432 Central System. The CAM is used to determine if the subsystem address is within

the address range of one of the five windows. This address matching function was the limiting factor in the unit design since the 16-bit address must be compared to the base address of each of the 5 windows and a match acknowledgment generated, all within 35 nanoseconds. The CAM was designed to permit addresses to propagate through the circuit asynchronously so that the elapsed time is a function only of the time required for the address to propagate through the circuit. After a match has been acknowledged, the CAM operation is synchronized with that of the rest of the chip via an external synchronization signal, the physical address is calculated and the access is completed (Ref 20:523, 524).

The sophisticated data transfer unit of the DAU performs the data accesses. A special byte packing buffer circuit was designed for use with the first window to permit high-speed transfer of large files or data blocks. This buffer reduces AP access time by prefetching and postwriting data which is being accessed in the 432's memory. Eight byte packets are transferred by the buffer which acts like a series of eight 16-bit shift registers. The design results in high-speed transfers and is independent of the actual AP architecture. The other windows perform unbuffered transfers, sharing a buffer bypass register which provides temporary storage. The bypass register and the buffer are nested together between the input and output bus. Their operation is controlled by the condition of a series of status flip-flops which are set as the buffer and register are used (Ref 20:525).

The MEU, unlike the DAU which is primarily composed of logic circuitry, operates under microprogram control. The MEU is responsible

for IP initialization, interrupt and fault handling and executing the 432 instructions as directed by the AP. The MEU operates as a five stage internal pipeline, allowing one microinstruction to be executed during each machine cycle. The MEU is locked into an infinite wait loop until an interrupt is received and a microinstruction sequence is initiated. After an interrupt is requested and acknowledged, the address of the sequence is computed. The ROM is then accessed, the microinstruction decoded, the operands fetched and the status of the 432 interface is determined. The data is then moved into the ALU where the necessary manipulations take place. The last step in the execution cycle is updating the 432 interface status and resuming the busy-wait loop. The actual circuitry includes an interrupt controller, an address sequencer, a 2K by 16-bit microcode ROM, a microinstruction bus and decoder and a 16-bit ALU. As with the DAU, two of these circuits possess unique designs which permit optimal performance within the size limitations. The interrupt controller is implemented in an ordered structure called a storage logic array which is compact and does not require any external connections or any other logic support. The other special circuit, a 2K by 16-bit ROM in which two bits are stored per cell, provides a high density storage media for the microcode (Ref 20:526, 527). This design is consistent with one of the major goals of the overall iAPX architectural design, removing as many of the system functions as possible from the software.

3.4 iMAX Operating System

The object-oriented architecture of the iAPX 432 and the anticipated operational environment of the 432/600 both greatly

influenced the structure of the iMAX operating system. Since the 432/600 is intended to function as an embedded computer system, the attainment of a range of applicability rather than "user friendliness" was stressed. The result was the creation of library of software packages which can be used by the system designer to build a system tailored specifically for his needs.

A number of the basic operating system functions, which in conventional systems are implemented in the kernel of operating system software, are performed by the iAPX 432 hardware. iMAX provides the remaining functions of this kernel as well as a variety of high level functions which support different types of applications. The lowest levels of iMAX are primarily concerned with the management of the system objects in order to provide the applications programmer with a high level computing environment. Since the hardware performs similar functions, instead of the typical master/slave relationship between the hardware and software, within the 432/600 a cooperative relationship is established (Ref 21:399).

The nature of the iAPX 432 hardware and the programming language in which the majority of iMAX is written, Ada, both require that iMAX possess a well-defined, modular structure. A modular structure provides numerous advantages including user configurability, extensibility and the enforcement of protection mechanisms. The current version of iMAX (V2.01) can be configured as either a minimal or fully functional operating system. Minimal iMAX is a subset of full iMAX which provides a simple multiprocessing execution environment. In this form, iMAX provides the operators necessary for the support of the hardware defined

objects, including operators for the creation, alteration and destruction of all types of system objects. Minimal iMAX also provides AP support software for the Debugger Peripheral Subsystem. The Debugger, which is an integral part of the 432/600 Cross Development System, is discussed in the next section. Full iMAX supports virtual memory management, an object filing system, storage reclamation and compaction, run-time process management, extended typing, fault handling and AP support software for multiple user-configured Peripheral Subsystems (Ref 17:MIN-1).

All of the iMAX operating system software, except for the iMAX AP support software, is supplied as a collection of Ada packages which can be readily modified and extended by the systems programmer to meet the needs of a specific application. The object code of a 432 applications program is directly linked with the code of the configured iMAX system. No special operating system calls or interfaces are required. With no special supervisor state, a protection mechanism which prevents user created or modified operating system modules from corrupting the other iMAX routines is essential. The Ada packaging mechanism requires that an execution environment be declared for each module, guaranteeing static module interface protection. Since the standard call instruction creates an access environment (activation record) for the called procedure using the protection domains established during compilation, a corresponding level of protection is provided during execution. This unique module level of protection enhances the reliability of the 432/600 computer systems which attain a high degree of flexibility due

to the configurability and extensibility characteristics of iMAX (Ref 21:399-403).

The iMAX Attached Processor (AP) support software is required whenever independent Peripheral Subsystems other than the Debugger Subsystem are implemented. This software is supplied as a combination of Ada packages, PL/M-86 modules and ASM-86 routines. The Ada packages provide for the procedural interface between the Central System operating system and the AP operating system, for IP window management and for transfer of data between the AP and the 432 memory space. The PL/M-86 and ASM-86 software is linked to the AP's real-time operating system software using a configuration routine. This software is used for initializing the IP and the 432 Central System and it provides terminal handling capabilities, as well. At the present time, the only AP support software available from Intel requires that the iRMX-88 real-time multitasking operating system be used as the AP's operating system.

3.5 Ada Programming Environment

Ada, the Department of Defense (DoD) high level programming language, is the high level language currently being supported by the 432/600 computer systems. Ada was the result of a design effort aimed at the development of a general purpose programming language for embedded systems which would incorporate the most current concepts in language design. These concepts would allow the designers to address the problems inherent in large-scale software development so that programmer efficiency could be improved and development costs lowered. The new language was to be adopted as a DoD standard and, as such, alleviate the software transportability problems currently plaguing government

agencies and consequently further reducing software development costs by eliminating duplication of effort (Ref 22:405).

The similarity between the 432's architecture and several of the major features of Ada was the motivating factor in the selection of Ada as the primary language for the 432/600 systems. The 432's object-oriented architecture directly supports such features as access protection for packages, automatic maintenance of the activation record stacks, and garbage collection and supplies multiprocessor support mechanisms that simplify concurrent multitasking. Another important factor which was considered was the enhancement of the power and versatility of a 432/600 which would be the result of using Ada's multitasking facilities in conjunction with the 432's transparent multiprocessing capabilities. If an Ada program is written to correctly implement multitasking, it will execute correctly on either single or multiprocessor 432 systems. With a multiprocessor system, execution time is much improved since the Ada tasks are shared by a group of processors with no increase in software overhead (Ref 22:405-407).

The architecture of the 432 supports several advanced programming concepts which are not implemented in standard Ada. For example, the 432 provides for user-defined, small protection domains and run-time type checking via extended type objects. Thus, implementations of procedures can be selected or modified at execution time or programs can be written to deal with totally unknown procedure implementations and data structures. Features such as these are not generally available in languages designed for use with the typically static embedded systems. Ada can support dynamic applications only through recompilation of the

modules which are unknown or changing. In order to fully exploit the power of the 432 architecture, an extended version of Ada had to be created for the 432. Full standard Ada is the heart of this superset which allows users to dynamically change the access rights of other users and to manipulate entities whose definitions are compiled or changed at some point during program execution. Full type protection is still provided for all system objects by the domain protection facilities of the hardware. In short, the extensions to Ada expand its applicability to program development for dynamic systems (Ref 22:408).

Ada programs for 432/600 computer systems are created in a cross-development environment, as shown in Figure 13. The Ada Compiler System (ACS), which resides on a host computer, translates the source code into compiled modules and creates source and error listings. The resulting individual environment files can also be combined into single integrated environment files with the interfaces for the entire group being defined within the file. The compiled program units and the required iMAX modules are then linked into a single program and a load image is produced. The linker program verifies the compatibility of the modules by comparing the ACS generated timestamps, resolves intermodule references, reports errors and summarizes the linker operation. The linker can also link new modules with previously linked programs and can output revision files which are used to maximize system efficiency during downloading. Communications software is used to download either the loadable code image or a revision file from the host to a local workstation called the Debugger. Downloaded files are stored on disk before being executed in a 432/600 system. Revision files are used to

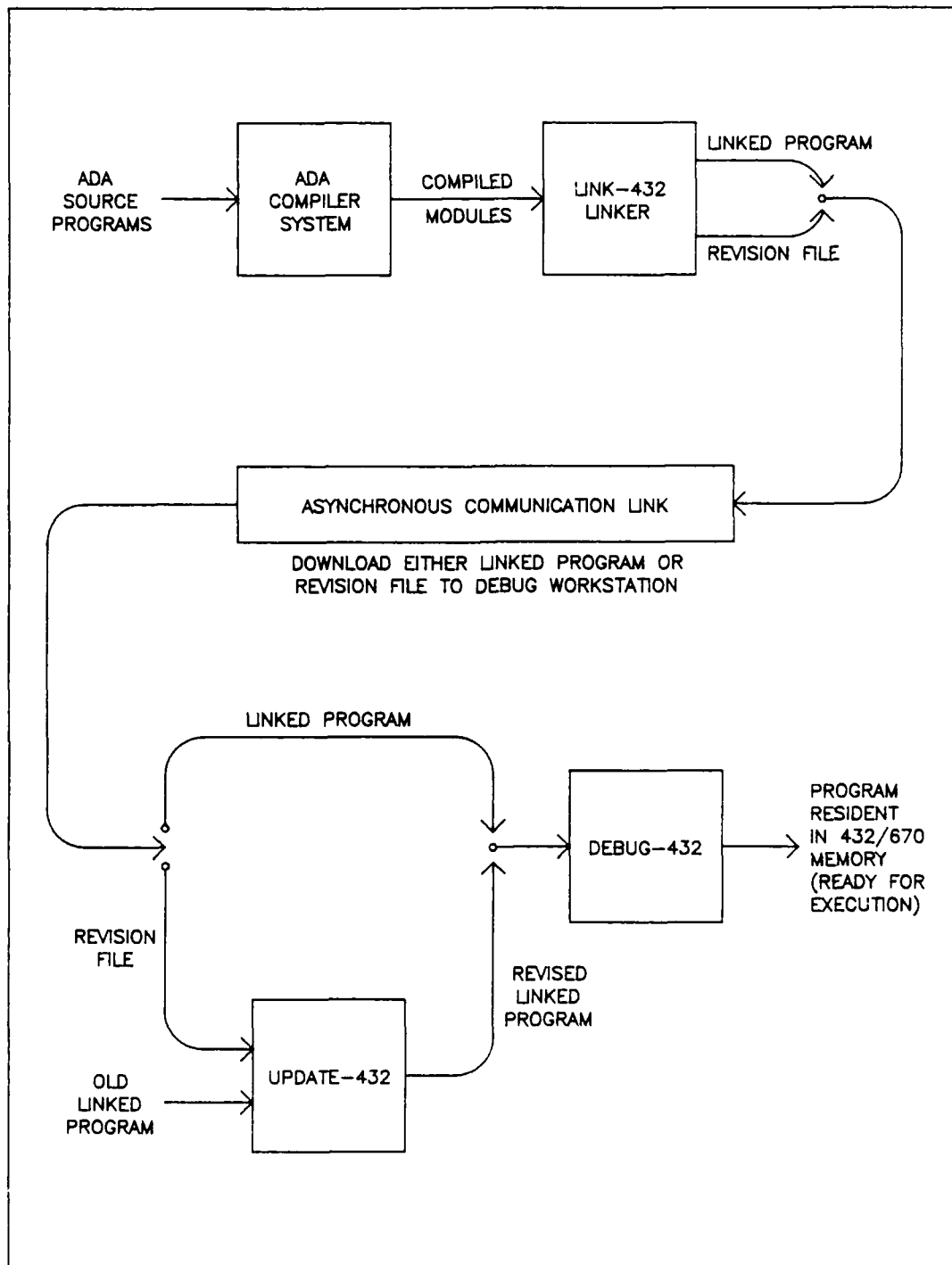


Figure 13. Program Development in the 432/670 Cross-Development Environment (Ref 23:1-8)

update previously downloaded code images before execution. In addition to its storage and updating functions, the Debugger also performs 432/600 system initialization, loads the object code image and starts program execution. The Debugger also provides the means for the user to monitor and modify program execution through the use of breakpoints and 432 memory examination. All system input/output must be performed through the Debugger station when minimal iMAX is used. In addition, Intel supplied Debugger software is used for performing diagnostics tests on the 432 hardware (Ref 23:1-9 - 1-12). A complete list of the 432 cross-development system software tools and a compatibility chart for the different versions is given in Appendix A. The release or version of a software package is extremely important since the 432/600 is still in the testing and development mode. For example, the ACS V1.01 release is an incomplete implementation of standard Ada which includes the 432-extensions discussed previously. A detailed description of operation of the cross-development system and its associated software can be found in References 23, 24 and 25. The actual implementation of AFIT's 432/670 Cross-Development System (CDS) is discussed in next section.

3.6 The AFIT Cross-Development System

The 432/600 computer system which was given to AFIT for research purposes is a model 432/670 Cross-Development System (432/670 CDS). Figure 14 shows the three main components of the 432/670 CDS: a VAX 11/780, an Intel Series III Microcomputer Development System and the 432/670 Mainframe. The VAX 11/780, running under either the VMS or UNIX operating system, functions as the host computer. The Series III,

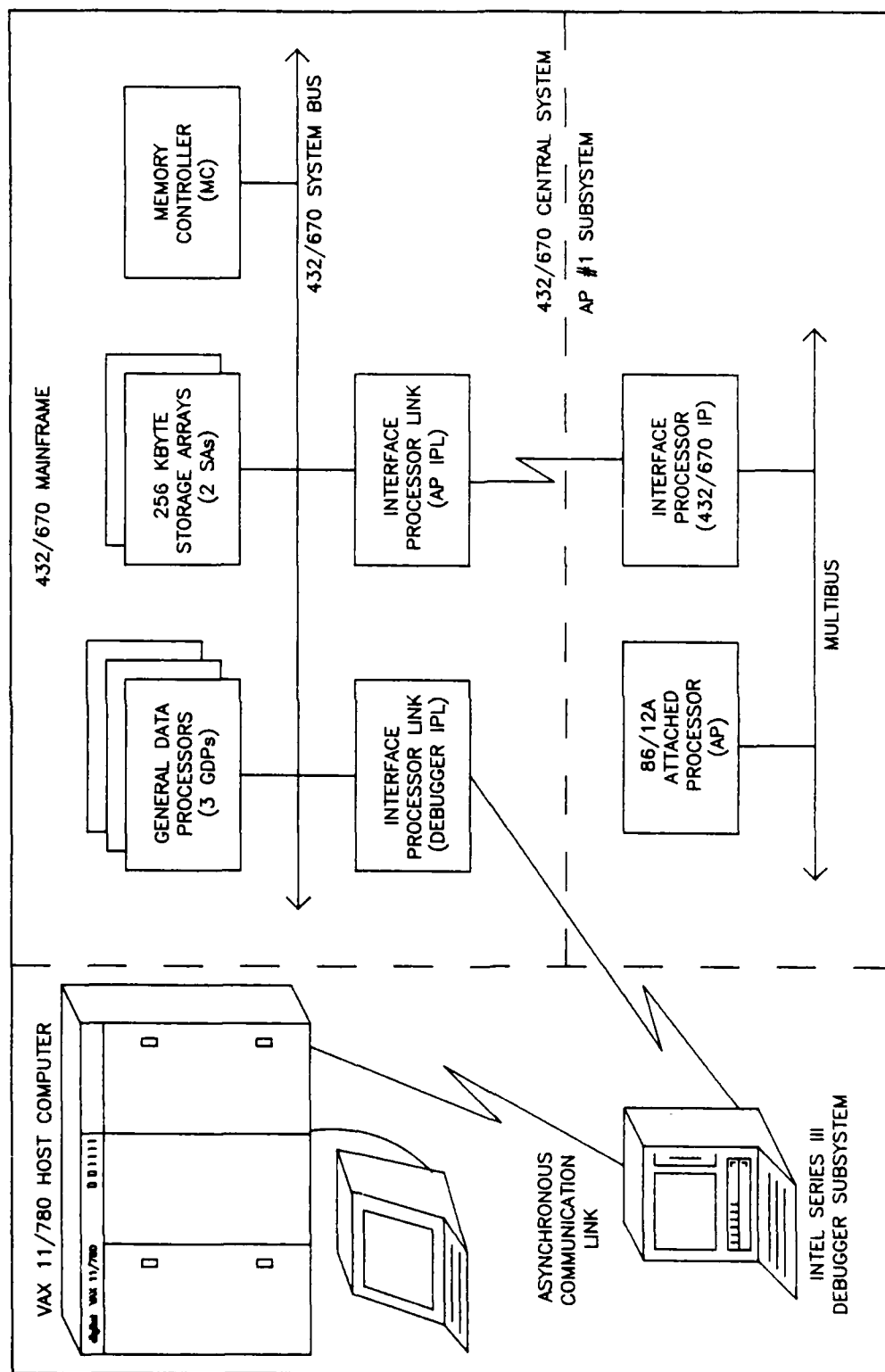


Figure 14. AFIT's 432/670 Cross-Development System Configuration

running Intel's ISIS II operating system, is the Peripheral Subsystem of the 432/670 which serves as a local workstation. This system, also called the Debugger Subsystem, is used to initialize the 432/670, load the executable code and monitor and modify program execution. Code files are transferred between VAX and the Series III using an asynchronous communication link software package. The 432/670 Mainframe is the "target" of the Cross-Development System. The mainframe contains a 12-slot System Bus and a 6-slot Multibus. The 432/670 Central System components include three General Data Processor (GDP) boards, a Memory Controller (MC) board, two Storage Array (SA) boards with a total of 512K bytes of RAM and two Interface Processor Link (IPL) boards. One IPL board is dedicated to the Debugger Subsystem; the other links the Central System to a Peripheral Subsystem supported by the Multibus. This subsystem contains an Intel 86/12A board with 64K bytes of RAM and 16K bytes of EPROM. A user-configurable real-time multitasking operating system, iRMX 88, is supplied as the operating system for the 86/12A Attached Processor (AP). A more detailed description of the components of the 432/670 CDS is presented in Appendix A.

CHAPTER 4

REQUIREMENTS

4.1 Introduction

The interface between the Intel 432/670 computer system and the MIL-STD-1553B data bus will provide a means of significantly improving the processing capability of current aircraft avionic systems. The functional requirements levied upon the interface are determined by the needs which the 432/670 is intended to fill and by the operational environment of embedded avionic computer systems. The requirements also reflect the particular constraints imposed by the availability of 432/670 hardware and software. In this chapter, the functional requirements of the interface are considered at the global and system levels. The global and system requirements are defined for all aspects of the interface design. For the system level, requirements are defined separately for the computer subsystem and the bus interface hardware. The separate treatment is necessary since the interface design spans several subsystems each of which possesses distinctly different hardware and software. System level requirements levied upon the entire system would necessarily be very general and should more appropriately be classified as global requirements.

4.2 Global Requirements

Global requirements are the highest level of functional requirements. These requirements are basically abstractions dealing with the characteristics of the idealized system. Their primary purpose

is to aid in the selection of the general design criteria which govern the entire design effort. These global requirements are detailed below.

4.2.1 Flexibility. Flexibility in a system design allows the same system to be used for a number of different applications with different 1553B data bus implementations. This is particularly important in this project since no specific applications have as yet been identified. Flexibility also helps cut life-cycle costs since fewer types of systems need to be designed and maintained. Excessive flexibility is to be avoided since overly general systems are usually difficult to use and expensive to maintain and modify.

4.2.2 Reliability. Reliability is a particularly important consideration in avionics computers since failures can result in loss of expensive equipment and possibly lives. Embedded systems also tend to be more difficult to repair due to accessibility problems.

4.2.3 Simplicity. Simplicity aids in system integration and enhances system reliability and maintainability since there would be fewer parts to fail and fewer to fix when failure does occur.

4.2.4 Maintainability. Maintainability refers not only to the ease of repairing a system, but the ease of upgrading a system as well. Maintainability is especially important in software development since software development costs are currently a major system expense.

4.2.5 Cost-Effectiveness. Cost-effectiveness involves minimizing life-cycle costs of a system. Cost-effectiveness figures are the result of a trade-off between development and maintenance costs and cost savings achieved through system versatility. The more expensive versatile system which can be expanded, easily modified to meet new

requirements and/or used for another application is likely to prove more cost-effective over the life of the system than would a cheaper, less flexible version.

4.3 System Requirements

The intermediate level functional requirements are system requirements. These requirements apply to all aspects of the design of a given system; however, they are more concrete in nature than are the global requirements. System requirements are also more specific, being derived in part from the description of the problem and from constraints such as those imposed by equipment availability and the operational environment. The system level requirements for this project, discussed below, are directed toward a general avionic system application.

4.3.1 Computer System Configuration. The AFIT configuration of the Intel 432/670 computer system, as described in Section 3.6 must be used since it is the only available 432-based computer system. The assumption will be made that a 10M byte Winchester disk drive is included with the Debugger Subsystem since only a minimal amount of code can be developed for the 423/670 Central System (CS) with only floppy disk drives available. The restrictions listed below are implied by the configuration of the 432/670 system as described in Appendix A.

4.3.1.1 432/670 Central System Software. All software development must be compatible with the Intel version of Ada. The current version of the compiler is an incomplete implementation of MIL-STD-1815A with several extensions added to allow the programmer to take advantage of the 432's object-oriented architecture. The size of the iMAX operating system, including Attached Processor (AP) and Debugger interface drivers

must not exceed 512K bytes since the entire operating system must be memory resident during system operation. Memory requirements for future application programs are not addressed since defining specific applications is outside the scope of this project. Reserving memory for these applications is not a major concern, since the 432/670 memory space is expandable by simple addition of Random Access Memory (RAM) or Read Only Memory (ROM) boards.

4.3.1.2 Attached Processor Software. All code for the Attached Processor #1 (AP1) subsystem must be written in either ASM-86 (8086 assembly language) or in PL/M-86, a higher level language which allows for some modular constructs. The iRMX-88 user-configurable operating system supplied with AFIT's system must be used as the basis for the AP1 software. The configured AP1 system software must be able to initialize the Intel 86/12A processor board and the 1553B bus interface hardware, transfer data between the 432/670 CS and the 1553B bus and drive the bus interface in accordance with MIL-STD-1553B protocol. The design should not preclude subsequent transfer of 432/670 CS initialization functions to the AP1 system. The total size of this AP1 code must not exceed 16K bytes. Total RAM usage during operation is limited to 64K bytes.

4.3.1.3 Attached Processor Hardware. The Intel 86/12A board residing on the 432/670 Multibus (AP1) is the only peripheral processor board available for hardware development work since the Debugger Subsystem Multibus is fully populated. All hardware to be interfaced to AP1 must reside on the 432/670 Multibus. The 86/12A board imposes a serious constraint upon software development since in its basic configuration it can support only 16K bytes of EPROM for the operating

system (Ref 26:1-4). This is adequate for most iRMX-88 applications; however, the iMAX AP Support software and the drivers for the 1553B bus interface must also reside in the EPROM. Although no coding is included in this project, the limited amount of EPROM, as well as the relatively small amount of on-board RAM (64K bytes), must still be taken into consideration during the software design phase.

4.3.2 MIL-STD-1553B Data Bus Interface Configuration. The DAIS implementation of the 1553B data bus, described in Section 2.5, is to be used in the design of all interface hardware and software. The RTU and associated subsystem designs must conform to the requirements stated in MIL-STD-1553B and Air Force Notice 1 to MIL-STD-1553B; DAIS compatibility will be given priority in the event of conflict. Detailed information concerning the specific requirements listed below can be found in Chapter 2. Notice 1 is discussed on pages I-14 and I-15, Reference 7.

4.3.2.1 Redundancy. Notice 1 requires Air Force systems to operate as dual standby redundant buses. The DAIS implementation uses one twisted-pair bus cable and one fiber optic cable. Interface to both is required. Within the RTU, the dual redundancy must extend through the word processor circuitry (analog transmitter/receiver and encoder/decoder) as a minimum.

4.3.2.2 Bus Coupling. Transformer coupling is required for electrical interfaces, as per Notice 1. Only a generic design of the fiber optic interface is required, since no specifics on the DAIS implementation were available at the time this project was undertaken.

4.3.2.3 Terminal Type. A terminal capable of providing all standard remote terminal unit functions is required. The remote terminal hardware which supports the broadcast command transmission may be used, but the broadcast mode is not to be implemented.

4.3.2.4 Transmission Method. Serial digital pulse code modulation at 1.0 MHz is required. Short term variations cannot exceed ± 100 Hz and long term variations must not exceed ± 1000 Hz. Manchester II bi-phase level encoding is to be used.

4.3.2.5 Bus Protocol. Standard 1553B protocol is to be implemented. Mode codes and message formats will be strictly as implemented on the DAIS and described in Figure 7 (Chapter 2).

4.3.2.6 Error Management. Word and message validation and a built-in-test (BIT) mode must be provided by the hardware. Invalid data, parity errors, word size errors, message length errors, word discontinuity, no terminal response errors and built-in-test (BIT) errors must be detected. All errors are to be reported via the BIT word. The BIT word format is to be implemented as shown in Table 2 (Chapter 2).

4.3.3 Operational Environment Constraints. The system proposed in this project is intended for use in an aircraft avionic system. In general, embedded avionic computers subsystems must be small, efficient and rugged, being able to withstand temperature, humidity and pressure fluctuations, shock and vibration and increased levels of dirt and possibly radiation. The system hardware must also be maintainable in the field; troubleshooting must isolate the malfunctioning boards which are then replaced in the field. These requirements can be applied only to

the 1553B interface hardware design portion of the project, since the 432/670 configuration is invariant. Then, since the interface hardware can be easily ruggedized by the substitution of the militarized versions for the commercial parts, only the requirements to minimize physical size and power consumption need be considered.

CHAPTER 5

DESIGN METHODOLOGY

5.1 Introduction

The 432/670 computer is a complex system containing several subsystems with distinctly different hardware and software. These subsystems are involved to varying degrees in the interfacing of the 432/670 to the MIL-STD-1553B data bus. Prior to developing the design of the general purpose interface, the term "interface" must be clearly defined with respect to subsystem involvement to ensure that the scope of this study is understood.

An interface between a conventional computer system and the MIL-STD-1553B would be considered to include the remote terminal unit (RTU) hardware, the software, if any, which executes on the terminal and the device drivers which are added to the computer's operating system to facilitate data transfer. Since the 432/670 cannot be directly interfaced to another system, the conventional interface must be expanded to include Attached Processor #1 (AP1), its software and the 432/670 Central System (CS) resident software for handling communication with a non-Debugger AP system. Therefore, for this study, the interface between the 432/670 computer system and the 1553B data bus includes the RTU hardware and any RTU resident software, the AP1 hardware, the iRMX-88 operating system and drivers for communicating with the RTU and 432/670 CS and the iMAX operating system and drivers for communicating with the AP1 Subsystem.

The complexity of the interface defined above precludes the creation and implementation of the interface as a single project. As discussed in Section 1.3, the objective of the design portion of this study is to produce system level hardware and software designs which are used in evaluating the viability of interfacing the 432/670 computer system to the MIL-STD-1553B bus. The hardware and software designs are presented in Chapters 6 and 7, respectively; however, a certain degree of overlap does occur, since inter-relationships between the two must be considered at each phase of the design. The design effort is based upon structured design methodology, which is discussed in the following sections.

5.2 Fundamentals of Structured Design

The system development life cycle consists of three distinct phases: analysis, design and implementation. During the analysis phase, the problem statement and the design requirements, which state how the system must work and what constraints are placed upon the design, are generated. During the design phase, these requirements are turned into the precise specifications, usually hardware schematics or pseudo code, needed for the implementation. The goal of a design effort is to optimize efficiency, reliability, maintainability, flexibility and utility of the system within the imposed design constraints. The use of structured design techniques to "engineer" a design solution has gained acceptance in the engineering world since typically the resulting systems are superior in quality to those designed using other, more haphazard, methods (Ref 27:3 - 12).

Structured design methodology provides a systematic top-down approach to solving the design problem. The design phase progresses through three distinct levels. The high level design is very general since it presents the system in its entirety. At the intermediate or system design level, the high level design is partitioned into a set of modules, often called subsystems in hardware designs, which are then progressively decomposed until sufficient definition for the detailed design effort is attained. The lowest, or detailed, design is then a precise statement of how to implement the system. Although the formulation of both quality high level and rigorous detailed designs is critical to the success of the design effort, the most emphasis must be placed upon the system level design process. The system level design plays a vital role in linking the other two design levels. The structured design techniques, the heart of the structured design methodology, therefore address the system level design effort in depth.

Structured design is based upon the principle that "... implementation, maintenance, and modification would be minimized if the system could be designed in such a way that its pieces were easily related to the application, and relatively independent of another." (Ref 27:19). The initial partitioning of the system decomposition should group highly interrelated parts of the same piece of the system and place them together, rather than unrelated pieces of the system. The system level design is organized with respect to the functional requirements of the system. The organization is generally based upon the functional requirements of the system rather than upon the physical requirements of the system.

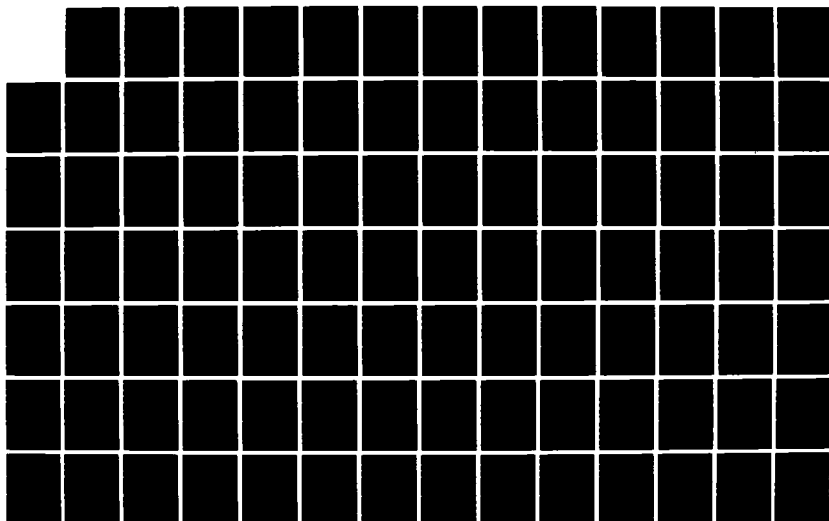
AO-A109 041

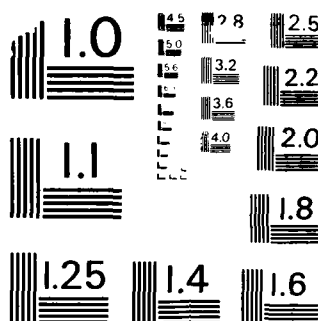
AN INVESTIGATION OF THE INTERFACING OF THE INTEL IAPX
432/670 COMPUTER SY.. (U) AIR FORCE INST OF TECH
WRIGHT-PATTERSON AFB OH SCHOOL OF ENGI.. L M BLACK
SEP 07 AFIT/OE/ENG/075-1 F/G 12/6

2/4

UNCLASSIFIED

ML





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

so small that new interconnections which prevent separate solution are introduced (Ref 27:17 - 19).

One way to look at the decomposed system is as a set of black boxes, each of which performs a required function, but whose internal structure is invisible to the whole (Ref 27:22). Each box, or module, is independent and can be designed, implemented, debugged and integrated into the system with a minimal amount of interaction with the other boxes. All interactions are in the form of a well-defined set of inputs which enter at a single point and a set of outputs which exit at a single point. This system can be realized by designing to minimize coupling between the modules and maximizing the cohesion within the modules.

Coupling and cohesion are central to the theory of structured design. Coupling is a measure of the degree of interconnection between modules or the amount of information about one module needed to understand another. In an ideal loosely coupled system, a designer or programmer needs only a minimal amount of information about other modules when designing, implementing, debugging or modifying a given module. Four factors influence the degree of coupling between modules. The most important factor is the type of connection between modules. Minimally connected systems, characterized by a single interface through which control and data are bi-directionally transferred, have the lowest coupling. To achieve minimal coupling, all input/output must be passed as arguments or parameters and all transfers of control must be conditioned to return control to the original activation point. A normally connected system also exhibits good coupling. In this type of

system there may be more entry points as long as each is minimal with respect to data transfers, control may return to an alternate entry point as long as the activating module defines the alternates or non-conditioned transfers of control may return control to a normal entry point. A pathologically connected system, characterized by unconditional transfers to labels within other modules or explicit references to external data, usually shows a marked degree of coupling. The second factor influencing coupling is the complexity of the interface as determined by the number of different items being passed. Regardless of how much data is passed, the higher the number of different items, the greater the coupling. The type of information flow is also important. Data flows produce less coupling than control flows, which produce less than hybrid flows of both data and control. The fourth factor concerns when the binding connections are made since the later values are bound to variables, the more readily they are changed. Thus, binding at execution produces the least coupling, while binding at coding time produces the most. The use of global variables also increases coupling (Ref 27:76 - 94).

The concept of cohesion is closely related to that of coupling. Coupling and cohesion are relatively correlated, in that minimizing coupling yields approximately the same results as maximizing cohesion. In practice, cohesion is usually the simpler concept to work with. Modular cohesion is a measure of how closely its constituent elements are related. Cohesion is effected by grouping elements possessing related properties, that is, according to certain associative principles. Seven levels of cohesion, each identifiable by a distinct

associative principle, are used in grouping elements into modules. These levels, in increasing order of cohesive strength are: coincidental, logical, temporal, procedural, communicational, sequential and functional association (Ref 27:95 - 98).

Coincidental cohesion is the result of random association of elements and is usually seen when trying to force-fit existing code into a design. Logical cohesion is created by grouping elements that fall into the same logical class, such as all input or edit functions. Grouping elements which occur during the same time period, such as at system start-up, results in temporal cohesion. A module has procedural cohesion if it exhibits no higher form of cohesion and its elements are grouped by virtue of belonging to the same algorithm, iteration, decision process or sequence of steps. Procedural cohesion is likely derived from modularization based on flowcharts. The fifth level of association, communicational, is the first level which shows some degree of relationship to the problem. Elements associated in this manner all operate on the same set of input data or produce the same output data. Elements are sequentially associated when the input to the module is processed sequentially through the elements in the module, with the output of each element being the input to the next. The highest, or best level of cohesion is functional cohesion, which groups all elements essential to the performance of a specific function. The operational definition of functional cohesion is that a module is functional cohesion if it is not one of the other forms of cohesion and its elements are related by any of the lesser forms of cohesion. The degree of cohesion is not solely a function of the associative principle used by the designer. The introduction of

side effects, such as error trapping, which occur in special instances marginally lowers the effective cohesiveness of a module. In determining the desirability of adding side effects, as in selecting the type of association for a module, the designer is faced with making trade-offs between optimizing the ease of implementing, debugging, maintaining and modifying the system through structured design techniques and meeting other requirements such as minimizing CPU time or memory use (Ref 27:98 - 125).

5.3 Structured Design Tools

The popularity of structured design has led to a proliferation of analysis methods and graphical means of presenting designs based upon the inherently abstract design concepts. Personal preference of the designer and the type of system involved usually drive the choice of presentation method. The widely accepted Yourdon implementation (Refs 27, 28) was chosen for this project since it offers a choice of analysis and presentation methods and since it can be adapted for use in hardware design. The primary Yourdon design tools, data flow diagrams, structure charts and the data dictionary, as well as transform and transaction analysis and the selected hardware and software design methodology are discussed briefly in the following sections. Reference 27 contains detailed descriptions of structure chart conventions and the analysis methods, while Reference 28 describes data flow diagrams and data dictionaries in more detail.

5.3.1 Data Flow Diagrams. The data flow diagram is a non-procedural model of the information flow within the system. The data flow diagram is typically the first graphical representation of the system to be

derived from the system specification or problem statement. Together with the data dictionary, it describes the system processes and data interfaces in detail sufficient to allow the system to be partitioned into its component modules (Ref 28:24).

The data flow diagram is composed of four basic elements, as shown in Figure 15. The data flows are shown as named vectors (X,Y,Z) between the process bubbles (P1,P2), data source and sink boxes (S) and the data files (F) which are represented by straight lines. Convention dictates that data flow names be unique, hyphenated and descriptive. Data flows from files need not be named. Data flows never pass control information or activate processes. Process names must adequately describe the transformation of the incoming data flow into the outgoing flow. A file can be any form of temporary data storage and must be described in detail in the data dictionary. The data flow linking the file to a process normally will show only the direction of the net data flow unless major bi-directional flows occur. The sources and sinks provide

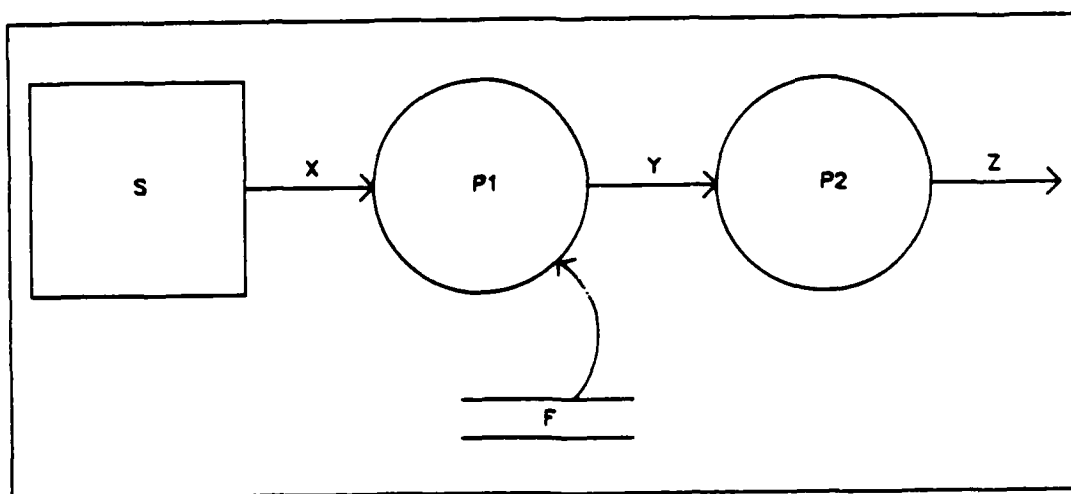


Figure 15. Basic Data Flow Diagram (Ref 28:51)

general information concerning the system's external connections and are generally insignificant (Ref 28:51 - 59).

Data flow diagrams are design tools, and as such, should be kept to a manageable size. Diagrams exceeding a page should be partitioned or leveled using top-down analysis, as shown in Figure 16. The top level diagram, called a context diagram, shows only the net inputs and outputs and the system process and serves to delineate the scope of the project. Diagram 0 is the partitioned version of the context diagram. Intermediate levels are added, with one diagram for each decomposed bubble, until a set of unpartitioned functional primitives is obtained. The diagrams are numbered using the number of the parent bubble, which in turn was derived from its diagram number with a period and unique local number appended (Ref 28:72 - 77). A number of rules must be followed when leveling data flow diagrams to ensure consistency. A detailed description of the process to be followed can be found in Reference 28.

5.3.2 Data Dictionary. The data dictionary is the essential document which gives meaning to the data flow diagrams. While primarily a tool for concisely describing the components of the data flow diagrams, the data dictionary can easily be expanded to include more detailed definitions of design elements as well as any other pertinent design/project information. Entry formats may vary depending upon type of element and upon project requirements, but should allow for as concise a definition as possible (Ref 28:126).

5.3.3 Structured Design Diagrams. A structure chart is a hierarchical model which reflects the relationships of the component

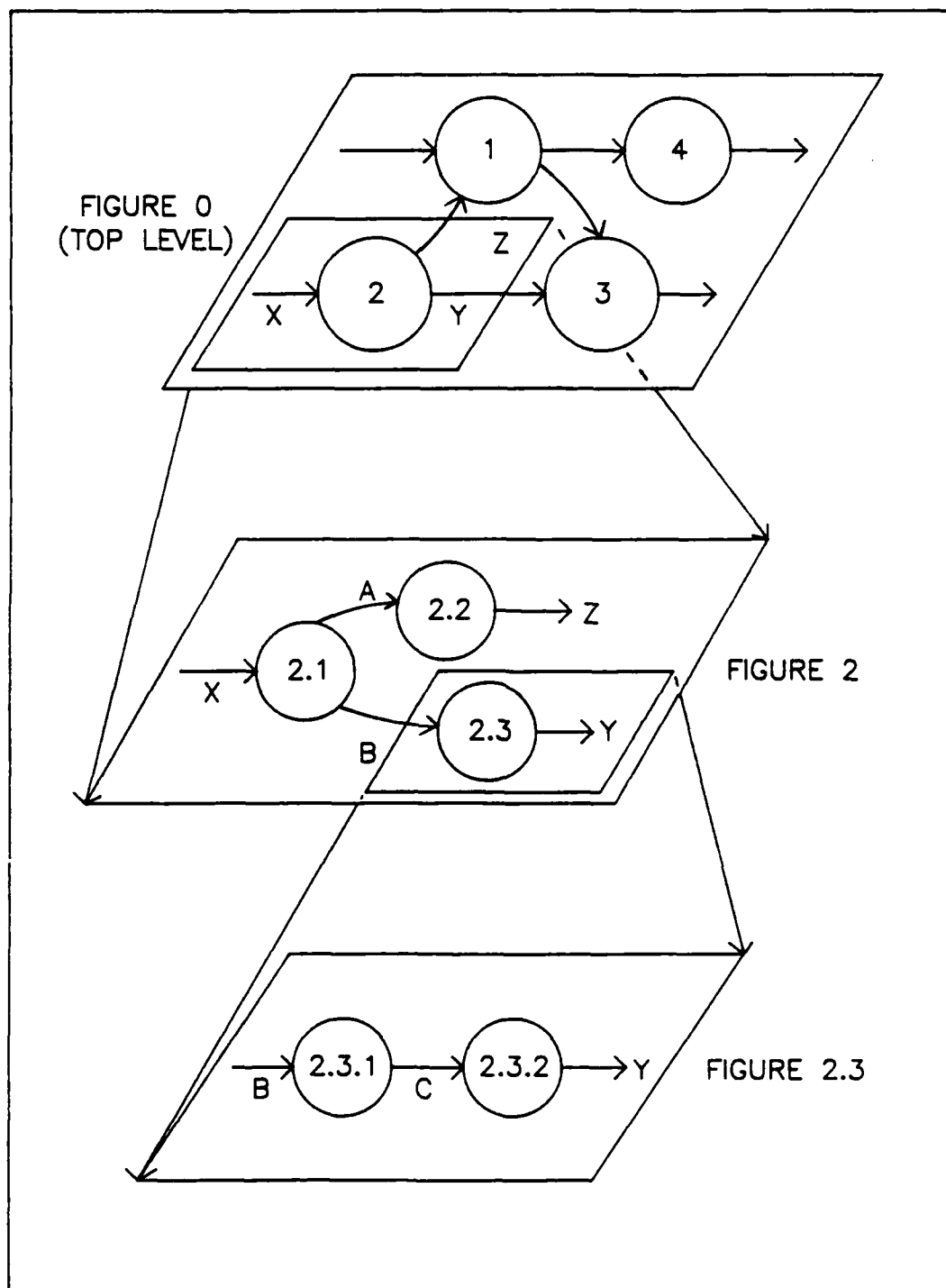


Figure 16. Data Flow Diagram Leveling (Ref 28:72)

modules of a system. The structure chart presents important information concerning the allocation of functions to the various modules, the modular interfaces and the system morphology, or shape, but timing, order of execution and procedural flow information cannot normally be deduced (Ref 27:45 - 57). By convention, modules in the structure chart are placed in approximately the same order from left to right as they would be executed. As illustrated in Figure 17, data flows in or up, called afferent flows, are usually placed on the left with flows down and out, called efferent flows, are on the right. Transform flows occupy the center. Coordinate flows, which redirect data, can exist in afferent, efferent or transform branches. The primary data flows are normally linked by the top level module, which is called an executive module if its sole function is to control and coordinate data flow. The modules shown in the structure chart are normally independent of the other modules; that is, they are not subsets of their calling modules. Structure charts therefore cannot adequately show the physical organization of the software. Another type of diagram, the hierarchy chart, better portrays structural relationships between modules. In the hierarchy chart, successive levels are refinements of the higher level modules. Although the hierarchy chart is usually considered as a substitute for structure charts, and, as such, is not included in the Yourdon methodology, it is useful in clarifying designs in which the organization of software modules and placement of supplied packages must conform to pre-established guidelines. A combination of both types of diagrams may be necessary to completely define the system software design.

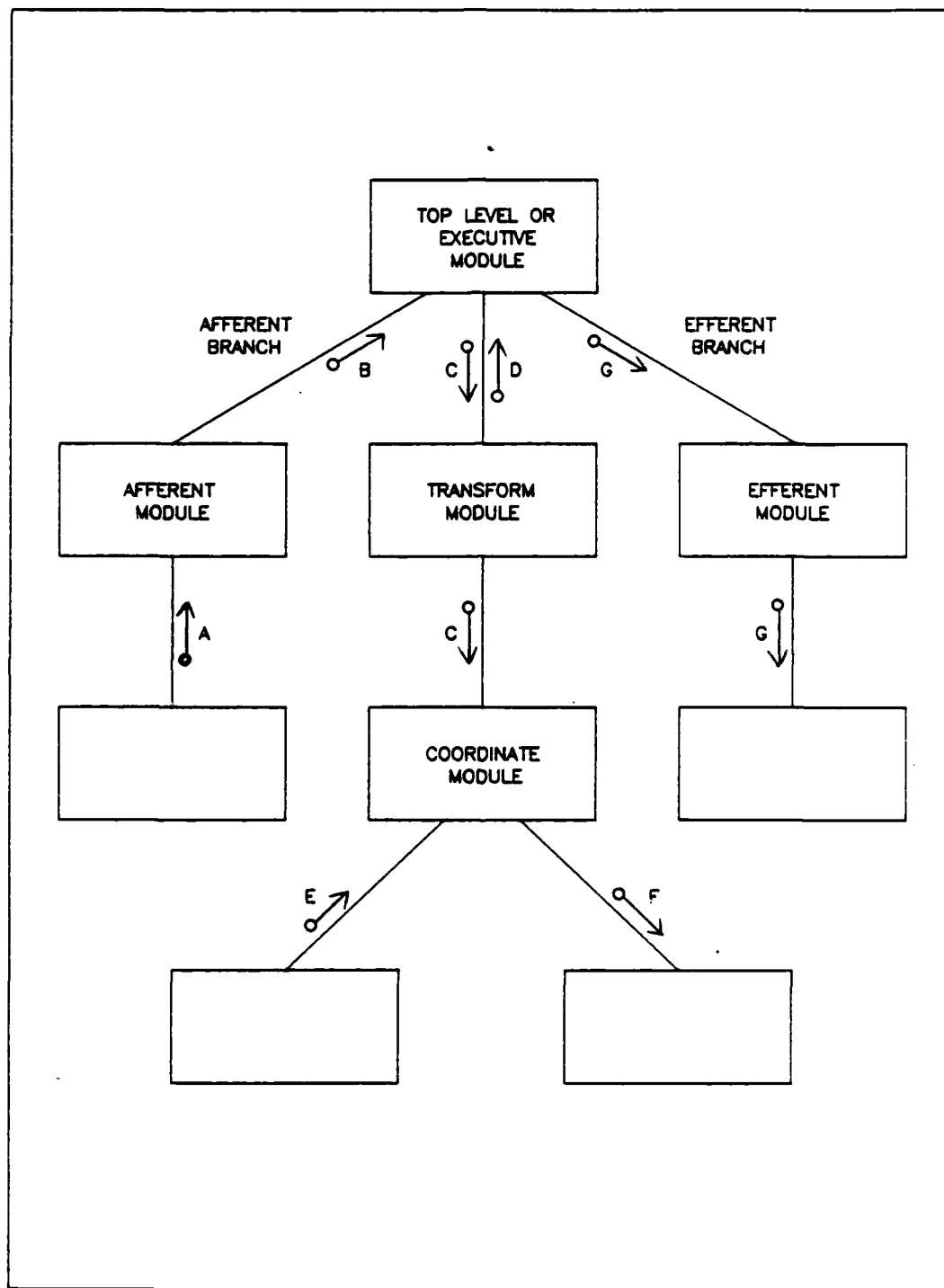


Figure 17. General Structure Chart Morphology (Ref 27:137)

5.3.4 Analysis Methods. Most well-designed, cost-effective software systems share a number of common morphological traits. These systems tend to be balanced, neither input nor output driven. They are usually highly factored, with top level modules being decision-making elements and low level modules being processing elements. The optimal span of control, or fan-out, of a module is between three or four for most systems. The scope of effect of the controlling module, the set of modules containing processing elements affected by a decision made by the module, is almost always a subset of its scope of control. Fan-in to a given module is also significant; maximizing fan-in decreases the degree of duplication of code (Ref 27:134 - 146). Systems possessing the design features described above also naturally lend themselves to optimized coupling and cohesion. Transform and transaction analysis are the two design strategies which are most frequently used to derive initial system level designs possessing the above characteristics.

Transform and transaction analysis are top-down design strategies whose purpose is to identify primary system processing functions and their associated high level inputs and outputs. These information flow models lead to structures which are frequently fully factored, possessing numerous intermediate level modules which control and coordinate the processing performed by their subordinate modules. The transform analysis strategy consists of four steps: restating the problem as a data flow diagram, identifying and partitioning the diagram to show the afferent and efferent branches and all central transforms, performing first level factoring to identify the immediate subordinates of the top level module in the structure chart and factoring the first

level subordinates to create the initial system design. The success in optimizing the lower level factoring is often due more to the experience of the designer, since, particularly for the transform branches, correctly distributing the decision processing functions while taking into account the above design heuristics and cohesion/coupling considerations is still an empirical process. The initial design is closely related to the data flow diagram in structure. The design often requires additional restructuring and refinement to optimize the design to reflect the real-world problem, design constraints and trade-offs (Ref 27:171 - 182).

Transaction analysis follows the same basic procedure as transform analysis, with the recognition that transaction-centered data flow diagrams are mapped into a different type of structure. While a transform changes the form of the input data to produce the output data, a transaction center analyzes an incoming transaction, initiates action depending upon transaction type and controls processing of the transaction. Standard structure for a transaction center uses four levels below the dispatch module: the transaction processor, the transaction level, the action level and the detail level. The analysis steps are somewhat more complicated than those for transform analysis; however, the same design heuristics and cohesion/coupling considerations should still be applied (Ref 27:202 - 207). The details of the analysis process are discussed in Reference 27.

Transaction analysis is rarely applicable to entire systems, but rather to portions of the system. Transactions are commonly found in the afferent or efferent branches of real-time process control, data

acquisition or interactive systems and in engineering applications. Care should be taken to identify transaction centers and treat only those portions of the system using transaction analysis, since research has shown that while easier to organize, transaction-centered designs are more more difficult to implement (Ref 27:217).

5.4 Hardware Design Methodology

The advantages of using top-down, structured design for hardware as well as software systems seem obvious, especially when a project involves parallel hardware and software design efforts. Unfortunately, structured design techniques were developed primarily for use in designing software systems and cannot be rigorously applied to hardware design. Of the design tools available, hierarchy charts are the most readily adaptable to hardware. The basic design techniques are applied in modularizing the hardware system. For hardware systems, the modules resulting from the top level partitioning are more appropriately termed subsystems and the diagrams themselves structure charts (referring to physical structure). The degree of decomposition required by the various subsystems depends upon the complexity of their assigned functions and upon any requirements levied by pre-existing hardware which must be incorporated into the system. For example, a module composed entirely of an existing circuit board needs no further analysis at this level since the determination of appropriate jumpering and switch settings for board operation are usually detailed design concerns. Other modules require supplemental block diagrams, analogous to software data flow diagrams, to clarify design decisions or to demonstrate the means by which certain requirements are met. The lack of

a specified format for such block diagrams is a major difficulty encountered in adapting the structured design techniques to hardware systems. Simple block schematics showing circuit and signal types served as substitutes. In several instances, where project requirements dictated that more detail than usual be included in the system level design, critical chip or chip category selections were included in the block diagrams.

5.5 Software Design Methodology

The software design methodology chosen for this project is the Yourdon methodology described in Section 5.3. The problem definition suggests that a combination of transform and transaction analysis methods is required to correctly deal with the diversity of processing mechanisms required. These include iRMX-88 operating system, with associated drivers which control the interface to a data bus with command/response protocol and the object-based message transfers to the 432/670 Central System, the basic 432/670 iMAX operating system and the message processing software resident on the Attached Processor. The resulting system level software design is described by a set of data flow diagrams, the associated data dictionary and the hierarchy charts which show the software organization. The structure charts would then be created during the detailed design phase, which is not addressed in this project. Figure 18 shows the conventions for the symbols used in the data flow diagrams and hierarchy charts presented in Chapter 7 and Appendices F and G.

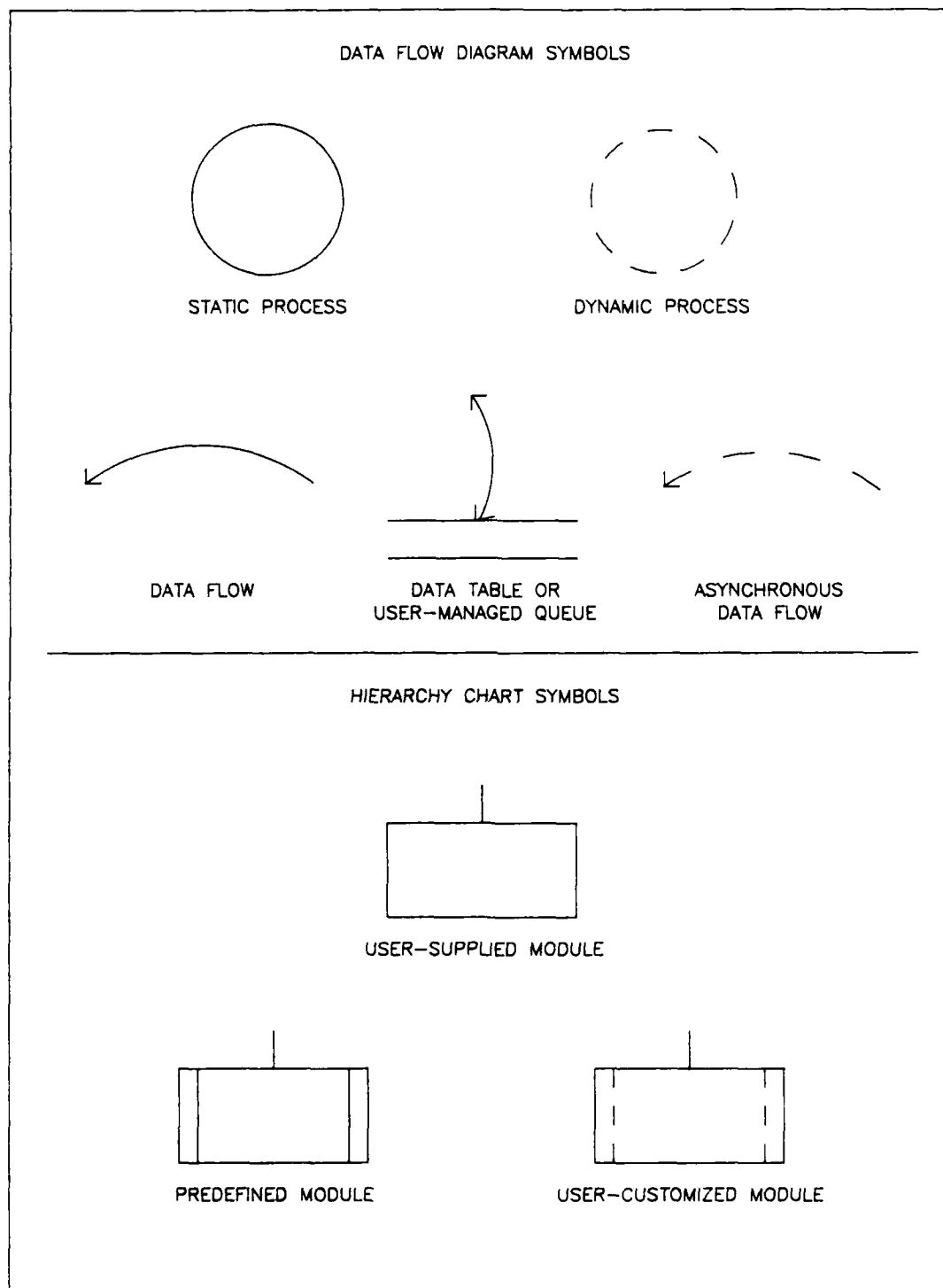


Figure 18. Symbol Conventions for Software Design Diagrams

CHAPTER 6

HARDWARE DESIGN

6.1 System Concept

The primary objective of the design effort is to provide a means of integrating the 432/670 computer system into current MIL-STD-1553B based avionic systems without impacting the operation of the existing system, in other words, to design a 432/670 Avionic Subsystem. Since defining the relationship between the two systems being interfaced is essential, a broader system concept is presented in lieu of a high level interface hardware design. The global requirements are applied to the conceptual design in the same manner as they would be applied to a more typical high level design.

The 432/670 Avionic Subsystem design must allow for both avionic system operation and for maintenance. Special provisions must be made for the maintenance functions since the 432/670 functions strictly as an embedded computer. As shown in Figure 19, the system hardware includes the 432/670 Central System (432/670 CS), the interface hardware, designated the Attached Processor #1 (AP1) Subsystem and the Debugger Subsystem. Included in the diagram is an optional connection to a serial (RS-232C) video display terminal which is supported by AP1. The operational and maintenance configurations are identical except for the physical connection to the Debugger. The Debugger is used during mission preparation for reloading the 432/670 CS software into memory, unless iMAX and the application software are stored in ROM. The

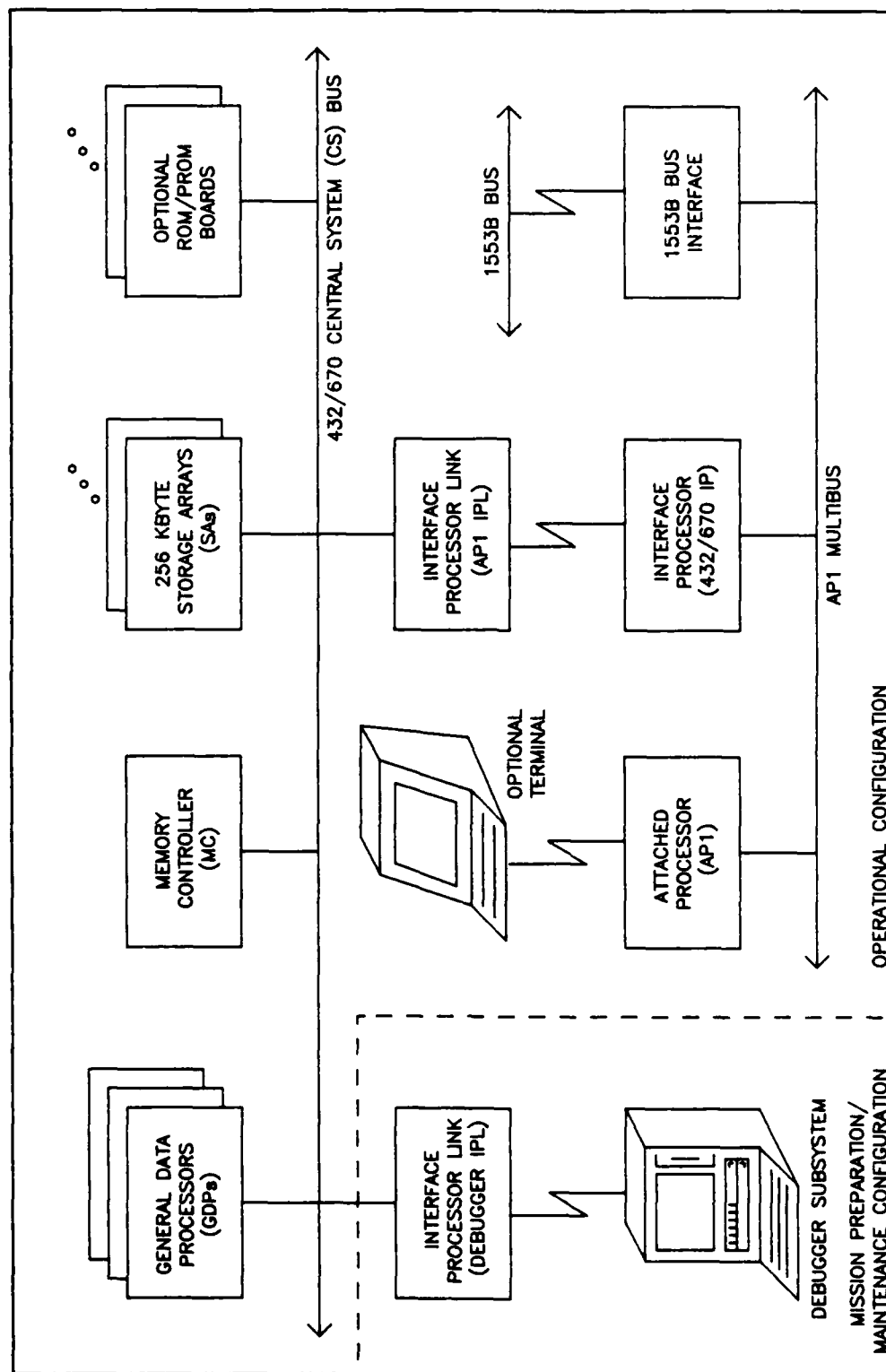


Figure 19. 432/670 Avionic Subsystem Conceptual Design

user-transparent multiprocessing capability of the 432/670 allows the Debugger to be connected when required by merely plugging the Interface Processor Link (IPL) board into the system bus and connecting the cable to the Debugger. If the iMAX operating system is configured to support the Debugger, no modification to the software is needed. The use of the Debugger for maintenance is the most viable approach since it is equipped with all of the hardware and software required for diagnostic testing of the 432/670 CS hardware and software. The Debugger also supports the Intel iSBC-957 Interface and Execution Package and the In-Circuit-Emulator (ICE-86), one or both of which is required for troubleshooting the AP1's Intel 86/12A board. Testing of AP1 as a remote terminal is easily accomplished by replacing the 1553B bus cable connection with a connection to a bus exerciser unit capable of emulating bus traffic. An added advantage of this design is that the maintenance configuration is identical to that required for the initial system development and for the testing of new applications programs.

The 432/670 Avionic Subsystem conceptual design meets all of the global requirements described in Chapter 4. The system is flexible in that the same system configuration can be used for avionic system operation, maintenance and software development. A variety of applications software can be supported due to the power of the 432 processors. The system is reliable; the user transparent multiprocessing provides processor redundancy. Malfunctioning General Data Processor (GDP) boards are automatically detected and bypassed by the process dispatcher with the only possible impact being a slight increase in processing time. Since time critical applications must be

supported by the AP, this impact should be negligible for most applications. The operation of the system is simple, requiring no operator intervention except during maintenance. The system hardware is easily maintained using the diagnostics which run on the Debugger Subsystem to identify malfunctioning modules which are easily replaced by board swapping. Software development and maintenance must be performed on a VAX computer. Since configuration management regulations normally preclude flightline software modification, there should be no resulting operational impact. An additional Debugger can be used for downloading the software for transfer to storage media. The cost-effectiveness of the system design is the result of using existing Debugger hardware and diagnostics to avoid incurring development costs and of using a single hardware configuration for operation, maintenance and development.

6.2 System Level Design

The 432/670 Avionic Subsystem is readily portrayed using an adaptation of a software structure chart. The structure chart shows the hierarchical relationship of the modules/subsystems and the functional partitioning of the system, as well as defining the subsystem interfaces to a limited degree. Unlike software structure charts where subordinate and parent modules are usually independent, connected only by control and data flows, the subordinate modules in a structure chart are subsets of their parents. The decompositions serve to further define the functions of the higher level modules. No signal flows are shown; if required to clarify the design at this level, they are included in the more detailed block diagrams of the modules.

The modular, extensible nature of the basic 432/670 system hardware facilitates the first level partitioning of the 432/670 Avionic Subsystem into its three component subsystems. As shown in Figure 20, these three subsystems are designated as the 432/670 Central System (432/670 CS), the AP1 Subsystem and the Debugger Subsystem. The 432/670 CS and Debugger Subsystem modules are existing (predefined) 432/670 subsystems which require no design effort other than a brief discussion of their assigned functions. The focus of the design effort is the AP1 Subsystem module.

6.2.1 432/670 Central System. The 432/670 CS module is composed of three General Data Processor Boards (GDPs), the Memory Controller (MC), a minimum of two Storage Array (SA) memory boards, optional Read Only Memory (ROM) boards for permanent storage of iMAX and application software and the Interface Processor Link (IPL) boards for AP1 and the Debugger. These boards reside on the 432 system bus as shown within the 432/670 CS partition in Figure 19. The 432/670 CS provides the power to support computationally intensive applications as required by the aircraft avionic subsystem. Its assigned functions are thus:

1. Provide a powerful user-transparent multiprocessing capability adaptable to a variety of signal processing and analysis applications.
2. Provide the communication link to the AP1 Subsystem.
3. Provide the communication link to the Debugger Subsystem during software development, maintenance and mission preparation.

6.2.2 Debugger Subsystem. The Debugger Subsystem consists of an Intel Series III Microcomputer Development Center and the Interface Processor (IP) board needed for communication with the 432/670 CS. As

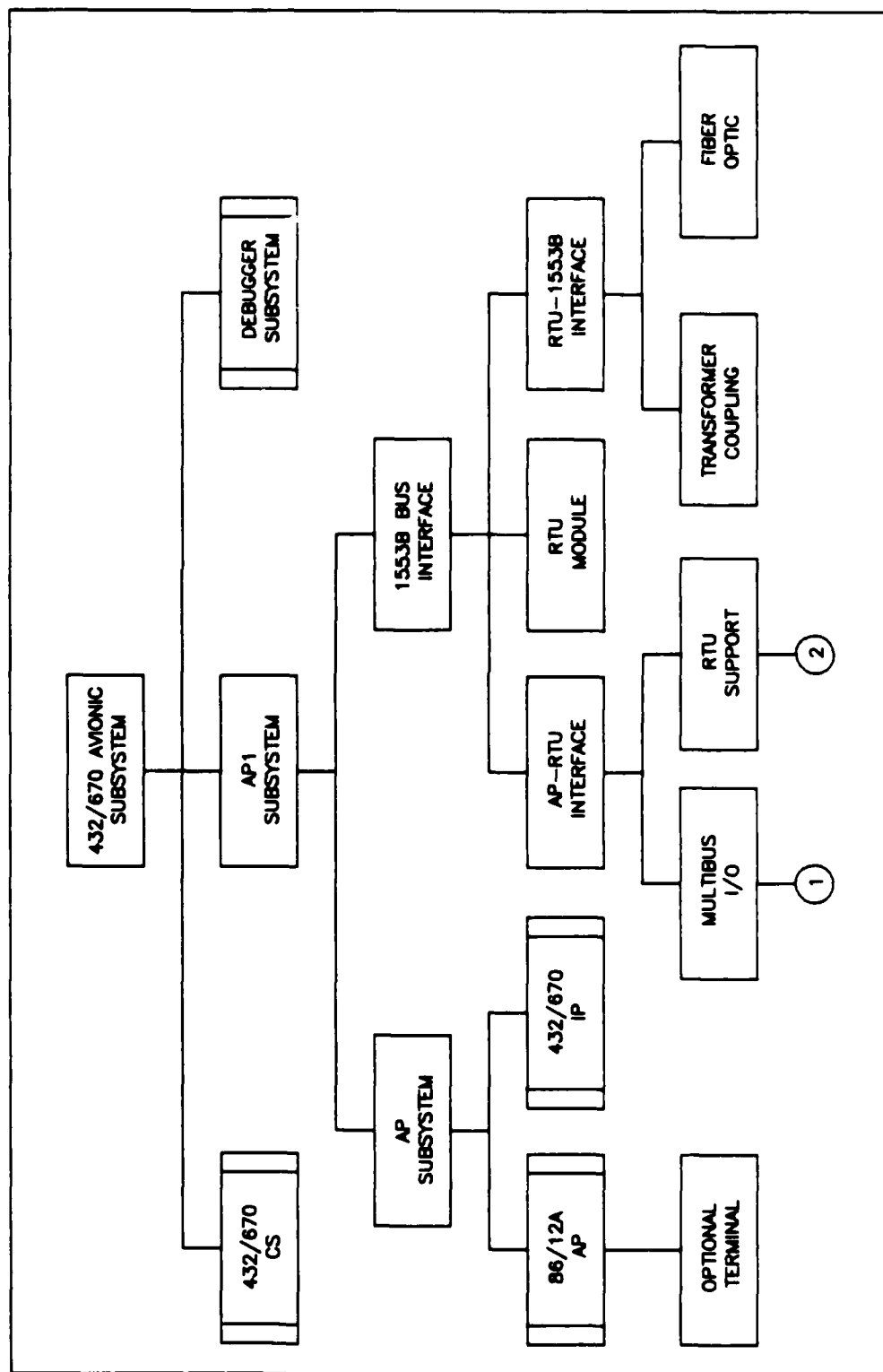


Figure 20. 432/670 Avionic Subsystem Structure Chart

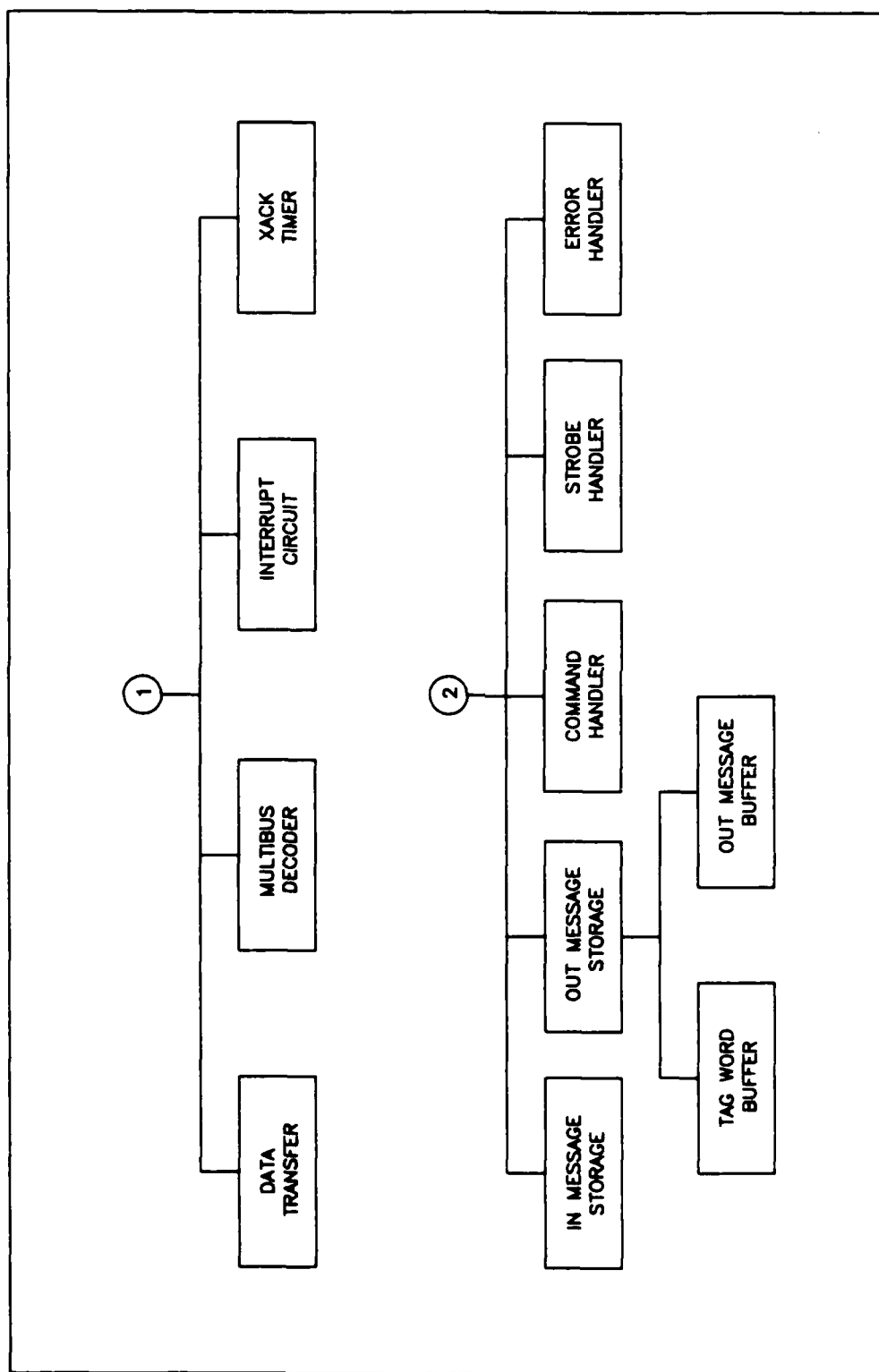


Figure 20. 432/670 Avionic Subsystem Structure Chart (Cont.)

discussed in Section 4.3.1, the 10M byte Winchester disk drive is assumed to be included in the AFIT Debugger configuration. The Debugger provides an interactive interface for the 432/670. In the operational environment, the Debugger performs the following functions:

1. Initializing the 432/670 CS hardware and loading the iMAX operating system and application software into 432/670 CS memory prior to each mission, unless the optional ROM boards are used as storage.

2. Executing the diagnostic software as required for hardware troubleshooting and scheduled maintenance.

The remaining Debugger functions support operating system and applications software development. These include:

1. Downloading 432/670 CS code from the VAX computer system and updating existing code modules if only revision modules are supplied.

2. Loading executable images into 432/670 CS memory.

3. Initiating and monitoring execution of 432/670 CS code and providing all standard functions required to debug code.

4. Supporting debugging of API code via the iSBC-957 or ICE-86.

6.2.3 API Subsystem. The API Subsystem performs two major functions in the 432/670 Avionic Subsystem. The primary function of this subsystem is to provide the interface between the 432/670 and the MIL-STD-1553B data bus since the 432/670 architecture precludes direct connection of peripherals. The API Subsystem also serves as a pre-processor for all data transferred between the 1553B bus and the 432/670 CS. Pre-processing of data destined for the 432/670 CS is essential if the timing requirements dictated by the 1553B bus protocol are to be met. The time delays inherent in passing data through the 432 IP windows

are excessive, especially since the polling which must be performed by the 432/670 CS processes in order to detect incoming service requests is controlled by the system's round-robin protocol. Remote terminal timing constraints require responsiveness available only through a closely coupled processor and peripheral hardware. Pre-processing the data also reserves all the 432/670 CS processing power for the computationally intensive applications by off-loading data management functions to the under-utilized 86/12A AP board.

Partitioning of the AP1 Subsystem hardware produces two functionally cohesive high level modules. The AP Module contains the 86/12A AP and 432 IP, the board set which provides the 432/670 CS access to its peripherals. The 1553B Bus Interface (1553BBI) Module is composed of all of the hardware necessary to interface the AP Module to the 1553B data bus. These modules and the operation of the AP1 Subsystem hardware are described in detail in the following sections.

6.2.3.1 AP Module. The AP module provides the means for the 432/670 CS to communicate with its two input/output devices, the 1553BBI and the optional video display terminal. As such, the AP module is responsible for performing four major functions:

1. Initializing its own hardware and software and that of the 1553BBI Module upon power-up or system reset.
2. Optionally initializing the 432/670 CS. Due to EPROM limitations on the 86/12A AP board, the optional 432/670 CS ROM boards must be added to allow storage of application program information.
3. Providing responses to bus controller commands as required by the bus protocol for commands not supported by the 1553BBI.

4. Transferring data between the 432/670 CS and the 1553BBI and providing buffering as required.

5. Providing an RS-232 serial interface for the optional video display terminal.

The AP Module components, as shown in Figure 20, are essentially predefined. The 432 IP board is used as supplied with the system. The 86/12A board requires only jumpering and connection to the optional terminal via a standard RS-232C cable. The jumper configuration is implementation specific, depending upon the types of peripheral interfaces desired. Since, for this project, the AP is dedicated to a single peripheral, the basic board configuration can be driven by the need to optimize the operation of the 1553B interface. These configuration requirements are summarized in Appendix B.

6.2.3.2 1553B Bus Interface Module. The 1553BBI Module provides the physical and logical interface between the AP Module and the 1553B data bus as implemented by the DAIS. This module is readily decomposed into three distinct submodules: the AP-RTU Module which interfaces the AP Module to the remote terminal unit (RTU) hardware via the Multibus, the RTU Module which performs all standard remote terminal functions as required by MIL-STD-1553B, Air Force Notice 1 and the DAIS implementation and the RTU-1553B Interface Module which provides the physical connection between the RTU and both the twisted-pair and fiber optic bus cables. The three modules reside together on a single Multibus card. Using a single card minimizes size and power consumption and decreases design complexity, since only a single Multibus interface is needed. This configuration is also simpler to maintain in the

operational environment, since if any 1553BBI problem is identified, the board can be replaced immediately, with no further diagnostic testing being required to identify the malfunctioning module. The principle disadvantage to this approach is that when the three modules are combined into a single physical unit, the modular interfaces tend to become ill-defined during the design effort. This violation of structured design principles frequently leads to increased system complexity and decreased reliability and maintainability. As can be seen in the block diagram in Figure 21, care has been taken to preserve the integrity of the modules comprising the 1553BBI. The modules are described in more detail in the following sections.

6.2.3.2.1 RTU Module. The implementation of the RTU Module is critical to the successful operation of the API Subsystem and, in fact, seriously impacts the design of the AP-RTU module hardware and the API Subsystem software. A variety of large scale integration (LSI) and hybrid smart 1553B RTU chip sets with different degrees of capability, size and power requirements are available. Based upon the system level and global requirements, the following criteria were used as guidelines in selecting the appropriate RTU chip set:

1. MIL-STD-1553B hardware compatibility. Considerations include matching transformer and terminal input impedance, waveform integrity, noise generation and rejection, common mode rejection and fault isolation specifications.
2. The degree of hardware implementation of RTU protocol functions and the complexity of required support hardware and software.
3. Built-in-test (BIT) word format compatibility with the DAIS.

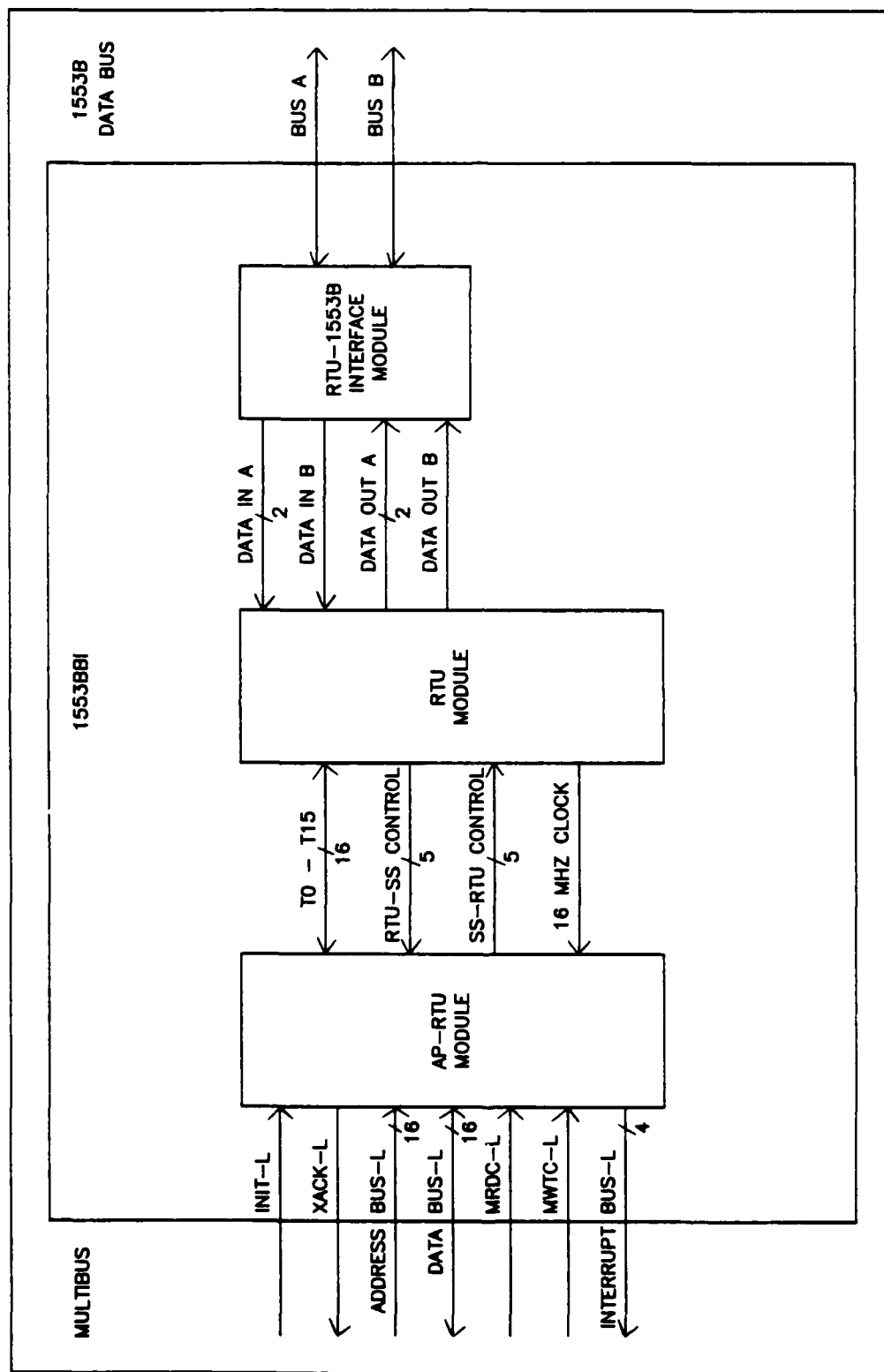


Figure 21. 1553B Bus Interface (1553BBI) Module Block Diagram

4. Compatibility with Multibus power. The Multibus supplies regulated $\pm 12V$ and $\pm 5V$ power.

5. Size, power consumption and complexity of the chip set hardware, with hybrid and LSI implementations being preferred for improved reliability and maintainability. The RTU and all required support hardware should fit on a single Multibus card.

6. The supplier should be an accepted avionic industry source to assure continued technical support and availability of parts.

7. Estimated cost of chip set and support hardware.

The chip sets evaluated fell into two basic categories: dumb and smart depending upon the degree of intelligence attributed to the terminal. The "dumb" terminals provide the required electrical compatibility with the 1553B bus and include basic transceivers and Manchester II encoders/decoders. Since only minimal protocol handling capability is provided, the amount and complexity of the subsystem terminal support hardware and software required to create a fully functional RTU is excessive. A dumb terminal implementation is slightly cheaper and does provide the only means to meet the requirement to match the DAIS BIT format, since the BIT word of most of the "smart" terminals cannot be modified by the subsystem. These terminals rely upon the bus controller (BC) software to handle differences in BIT formats between the various implementations of RTUs on the bus. The advantages provided by a smart terminal far outweigh the advantage of being able to control the BIT format, since some the customization of BC software is required to support non-standard applications such as the 432/670 Avionic Subsystem.

The smart chip sets perform most RTU functions without outside intervention, effectively presenting a much simpler interface problem to the subsystem designer. The most difficult aspect to evaluate when comparing the various smart terminal chip sets is the complexity of the subsystem support hardware and software needed to meet their different, but frequently stringent, timing and synchronization requirements. Descriptions of these requirements seem to be written for experienced 1553B subsystem designers, tending to be both inadequate and esoteric. Comparisons of system capabilities and implementation requirements based upon such documentation may not be totally accurate.

The chip set chosen to perform the AP1 Subsystem functions is an industry accepted combination manufactured by ILC Data Device Corporation (DDC) and Micro Circuit Engineering (MCE), whose chips are distributed in the United States by DDC. This chip set, consisting of five basic components, meets or exceeds all MIL-STD-1553B requirements, including requirements for reliability and maintainability. These LSI chips are low power and are Multibus compatible. The chip set implements all standard protocol functions with a minimum of subsystem intervention being necessary, without including an excessive number of unnecessary functions which would drive up the cost. RTU support hardware includes only the bus interface hardware described Section 6.2.3.2.2 (RTU-1553B Interface Module), a 16 MHz clock oscillator, two simple time out circuits and a high speed transfer circuit required for receiving continuous blocks of 32 data words. A description of the RTU components, a block diagram of the RTU showing the basic connectivity between the components and its external interfaces, diagrams of the

support circuitry and notes concerning the design are included in Appendix C. The external RTU timing requirements are discussed below with the AP-RTU Module design, where their major impact is evident. For information on the internal timing of the RTU chip set, see References 29 through 31. References 29 through 31 are also the sources for the RTU specific information and timing diagrams presented in the remainder of this chapter.

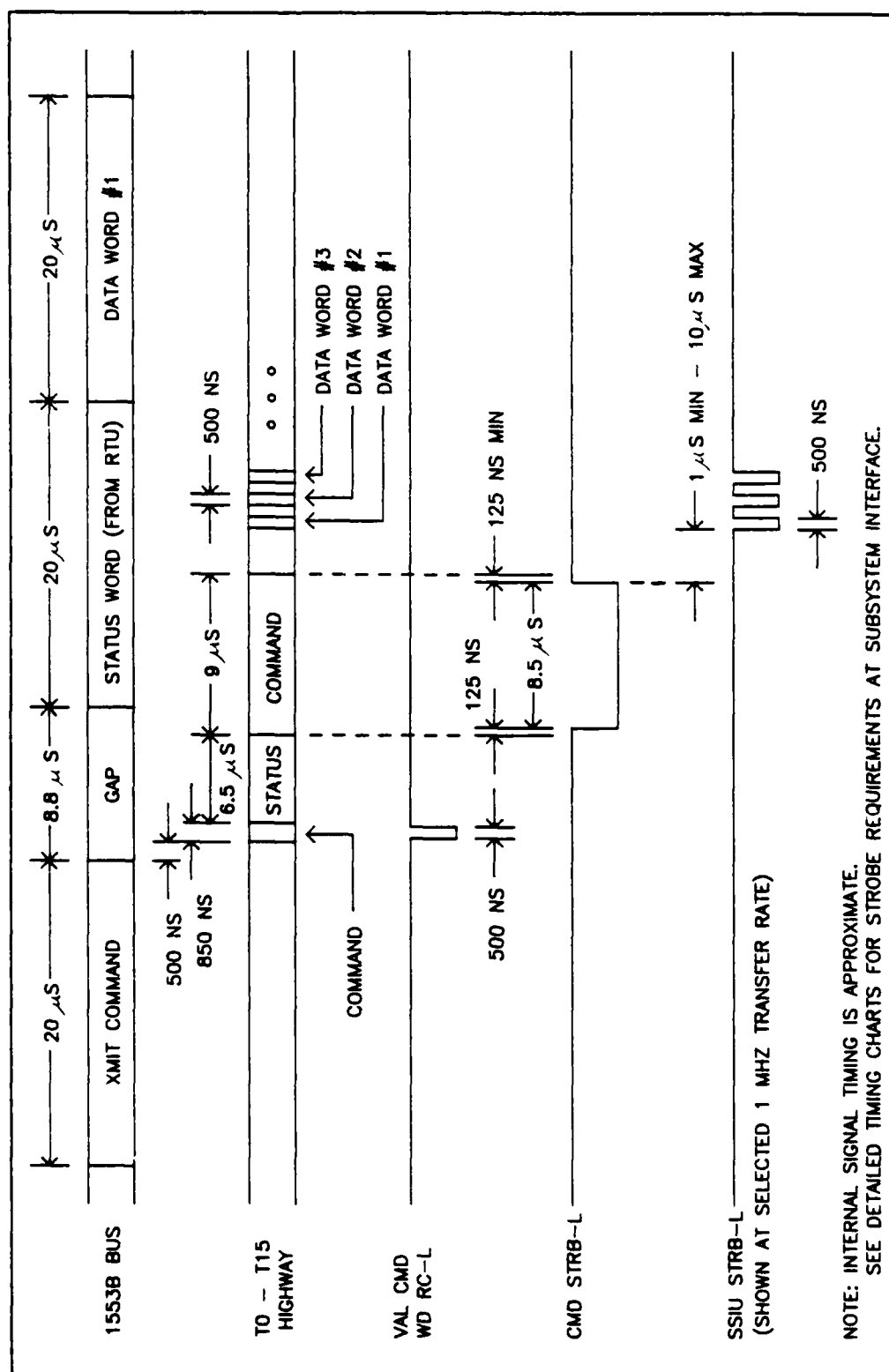
6.2.3.2.2 RTU-1553B Interface Module. The RTU-1553B Interface Module provides the physical connection between the RTU and the dual redundant 1553B buses. This module is composed of two functionally related, but totally independent submodules. The Transformer Coupling Module provides the electrical interface to Bus A, the twisted-pair bus cable, according to the specifications included in Air Force Notice 1. The design, included in the RTU-1553B Interface Module Block Diagram in Appendix C, is very detailed to show that both these requirements and the specific interface requirements of the RTU chip set have been met. The Fiber Optic Module design, also included in the block diagram, is very generic, since, as stated in Chapter 4, the fiber optic implementation of the Bus B was not defined at the start of this project. The RTU-1553B Interface Module design, as shown in the block diagram, meets the requirements of Air Force Notice 1, which specifies that dual redundant operation and electrical isolation of the remote terminal unit must extend from its interface to the redundant buses through at least the word processor circuitry.

6.2.3.2.3 AP-RTU Module. The AP-RTU Module is responsible for interfacing the RTU to the AP via the AP1 Subsystem Multibus, and, as

such, represents the bulk of the hardware design effort. The AP-RTU Module consists of two primary modules: the Multibus I/O Module which provides the actual Multibus interface, including address/control decoding, data transfer, transfer acknowledgment and interrupt handling, and the RTU Support Module which provides the subsystem message handling capability required to support the RTU. The timing constraints imposed upon the subsystem, which from the perspective of the RTU, includes all of the AP1 Subsystem beyond the RTU boundary, drive the major design decisions, especially those concerning partitioning of message handling functions between AP1 software and AP-RTU Module hardware. These general transmit, receive and mode command timing constraints, shown in Figures 22, 23 and 24, respectively, are discussed briefly below. The detailed timing constraints are discussed in Appendix E.

The 1553B data bus operation is based on an asynchronous command/response protocol. As described in Chapter 2, all RTU functions are initiated by receipt of a command word from the BC which must receive acknowledgment in the form of transmission of the status word by the RTU. The RTU automatically receives and stores the command and any associated data words, performs all 1553B error-checking functions and transmits the status word within the required time period. The RTU also handles most mode commands internally; for this application only the transmit vector word mode function requires subsystem intervention. Valid command words are transferred from the RTU's Encoder/Decoder to the Protocol Sequencer for storage during a 500 nanosecond prior to the transmission of the status word. While the command word is available on the T0 - T15 16-bit Data Highway which spans the RTU and provides

the data path to the subsystem, the Protocol Sequencer generates a VAL CMD WD RC-L (Valid Command Word Received) signal. After status word transmission, the RTU officially notifies the subsystem of receipt of a valid transmit or receive command by sending the CMD STRB-L (Command Strobe) signal for 8.5 microseconds, during which period the contents of the RTU command register are gated onto the T0 - T15 Data Highway. As shown in Figure 22, for a transmit command, the subsystem must generate the first SSIU STRB-L (Subsystem Interface Unit Strobe) transmit control strobe and gate the first word to be transmitted onto T0 - T15 within 1.0 to 10.0 microseconds following the trailing edge of CMD STRB-L. Subsequent words and strobes must be supplied at a rate between 62.5 KHz and 1.0 MHz. As shown in the detailed timing diagrams in Appendix E, SSIU STRB-L must be very closely synchronized to the 1 MHz clock signal produced by the RTU Protocol Sequencer. Receive timing can be controlled by either the RTU or the subsystem. The 0.5 MHz rate of the RTU-controlled transfers is inadequate to support the continuous reception of 32-word blocks of data required by a data processing system. Control can be passed from the RTU's 0.5 MHz DATA STRB-L (Data Strobe) to the subsystem's EXT SO-L (External Strobe Out) signal by adding a high speed transfer circuit to the RTU. As shown in Figure 23, the DATA STRB-L signal is used to initiate the receive sequence, but is ignored after the subsystem assumes control. Few external timing constraints are imposed upon the subsystem-controlled transfer; however, to optimize performance the delay in initiating the transfer should be minimized (less than 500 nanoseconds), and, ideally, the transfer should proceed at the maximum allowable rate of 2.0 MHz.



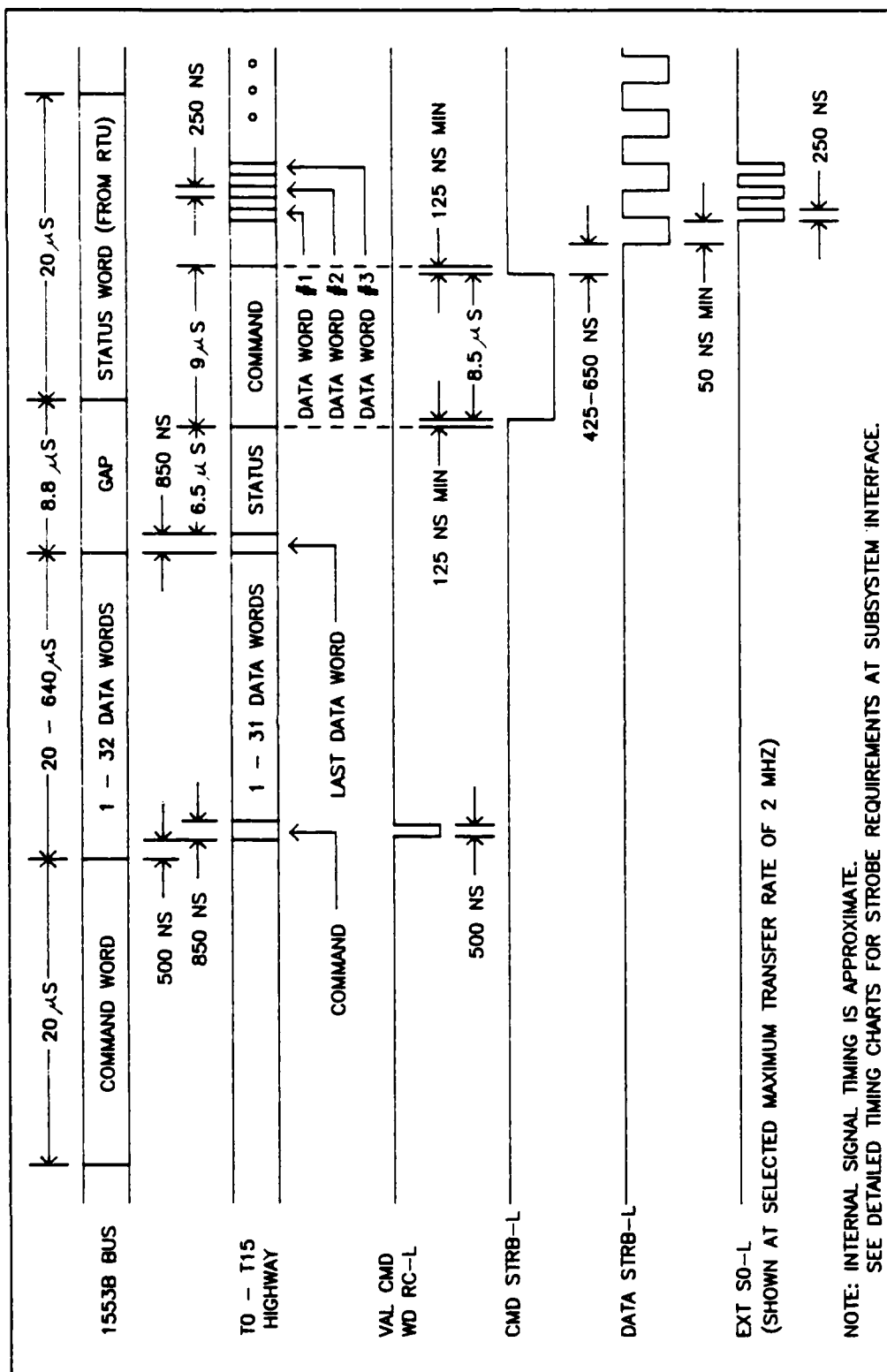


Figure 23. General Receive Command Timing Considerations

The transmit vector word mode command timing, shown in Figure 24, is quite stringent, requiring that the subsystem gate the vector word onto the highway within 250 nanoseconds following the leading edge of the EN VEC WD-L (Enable Vector Word) signal. Note also, that for mode commands, no CMD STRB-L signal is generated, since subsystem intervention in mode command processing is expected to be minimal. Whenever the subsystem does not meet the RTU's timing requirements, the RTU generates an error condition and terminates the current sequence, which must be completely repeated. Frequent errors degradation of the bus system performance and impact the operation of other subsystems which upon the 432/670 Avionic Subsystem for critical processing (Ref 29:71 - 131).

The timing requirements imposed upon the subsystem by the RTU drive the basic design of the AP-RTU Module. The first, and most critical, step in designing this module was selecting an appropriate mechanism for transferring data between the RTU and the AP Subsystem. The first approach to be analyzed was based on a standard interrupt-driven system, in which the AP Subsystem is notified via interrupts to transfer data to or from the RTU's First-In-First-Out (FIFO) buffer. The AP-RTU Module in a standard interrupt-driven system would consist of an address/control decoder, a data transfer circuit to buffer the data lines, an interrupt handler and a transfer acknowledge (XACK-L) circuit. A preliminary examination of the timing indicated that port-driven input/output, which is restricted to 8-bit transfers by the 86/12A board, would be slower than memory-mapped input/output with its 16-bit transfer capability. Memory-mapped input/output has the additional

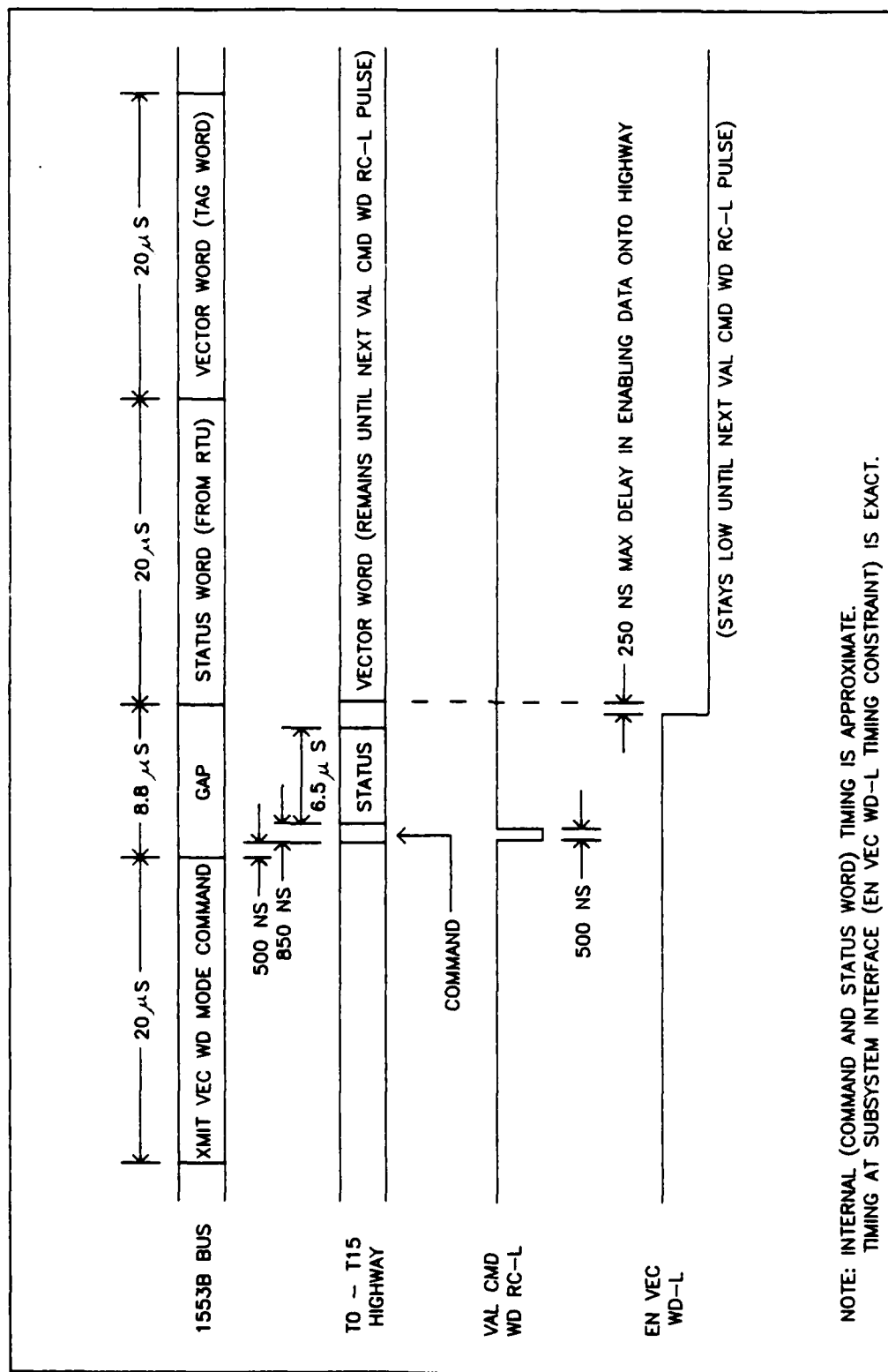


Figure 24. Transmit Vector Word Timing

advantage of being simpler to implement for this application, since the RTU requires concurrent access to the full 16-bit data path. The operation of this system is relatively simple. To allow adequate time for decoding the command word, the early VAL CMD WD RC-L signal is used to generate the service request interrupts to the AP Subsystem. Assuming that the system assigns the highest priority level to these interrupts and that a non-bus vectored interrupt scheme, in which all interrupt vector addresses are generated by the AP, is used, the interrupt processing time can be found by adding the following delays:

1. The 8259A Programmable Interrupt Controller's delay in generating an interrupt to the CPU after receiving an interrupt over the bus. The maximum delay is 350 nanoseconds (Ref 32:B-122, B-123).

2. The CPU latency, or time for the CPU to recognize an external interrupt request, which depends upon the remaining execution time of the currently executing instruction. In the worst case condition, the latency could span two instructions and would involve multiply, divide or rotate instructions. The fastest instruction execution time is 400 nanoseconds if the instruction is in the queue, and 1 microsecond if not (Ref 32:2-22 - 2-24).

3. The CPU's interrupt response time to an external maskable interrupt. The CPU requires 61 clock cycles (at 200 nanoseconds per clock) from the time the interrupt is recognized to begin execution of the service routine (Ref 32:2-56).

Assuming an average 1 microsecond CPU latency, the AP would take an average of 13.5 microseconds to start executing the interrupt service routine. If the service routine is simple, the command word can be

successfully read by the AP while CMD STRB-L is low, since this period occurs 6.5 to 15.5 microseconds after the trailing edge of the 500 nanosecond VAL CMD WD RC-L signal. Addition of auxiliary storage to allow latching the command on the 1553BBI would not significantly alter the basic transfer mechanism and would ensure a successful command word transfer to the AP, given worst case processing delays.

The primary difficulties with the standard interrupt-driven system arise during the subsequent data word transfer, including vector word transmission. The minimum time for transferring a data word between the AP and the 1553BBI can be calculated using the following instruction execution and transfer delay times:

1. MOV (16-bit), memory to register: 9 clocks plus EA (effective address calculation, 5 to 12 clocks).

2. MOV (16-bit), register to memory: 8 clocks plus EA.

3. Latency for gating read/write requests of the Multibus via the 8288 Bus Controller. Due to advanced command feature of the Intel 8288 Bus controller, the only delay normally encountered when operating in the 86/12A's maximum mode is that due to the addition of processor wait states while awaiting the XACK-L response from the 1553BBI. This delay cannot exceed 65 nanoseconds, and can, with a flexible XACK-L circuit implementation, be minimized (Ref 32:A-37 - A-38, A-182).

Using the extremely oversimplified case where the data word transfer is accomplished with only one memory-to-register and one register-to-memory transfer, and assuming that optimal effective addressing is used (5 clocks), the minimum data transfer time for a memory-mapped read or write is approximately 5.5 microseconds. This delay falls within the

1.0 to 10.0 microsecond initial response window for data transmission and is adequate to support the subsequent transfers which must occur at a rate between 62.5 KHz and 1.0 MHz; however, due to the timing variations in this transfer process, 1553BBI-resident hardware would be needed to handle the stringent transmit synchronization. The RTU does not impose any synchronization requirements for data reception, but the required data rates are much greater. The 2.0 MHz transfer rate, which allows the RTU to continuously receive 32-word data blocks, requires a 500 nanosecond read delay. Even when using the RTU's internally-generated, default transfer rate of 0.5 MHz (via DATA STRB-L), the delay is still 2.5 times too long. The other timing problem is that encountered in processing the transmit vector word mode command. This problem arises due to the lack of the 8.5 microsecond additional processing time afforded by the CMD STRB-L generation and the associated transfer of the command to the subsystem. As shown in Figure 24, the AP must read and decode the command and write the vector word to the 1553BBI within approximately 7 microseconds, since only a 250 nanosecond delay following EN VEC WD-L is tolerated by the RTU. If this were the only system timing problem, it could be circumvented by using a write-ahead scheme in which the AP anticipates the command, writes the vector word to a 1553BBI latch and allows the 1553BBI hardware to control the transmission. However, since the basic data transfer problems are much more far-reaching, an alternative data transfer mechanism is required.

Several popular approaches to handling high-speed data transfers between peripherals and CPUs were next considered. These included direct memory access (DMA) transfer, which would require additional

synchronization hardware and a memory management mechanism, microprocessor-controlled transfer using an on-board microprocessor, PROM-based software and dual-ported memory, and microprocessor-controlled transfer using a custom LSI microcodable controller and dual-ported memory. These approaches were rejected, having been determined to be unnecessarily complicated or too expensive to implement for a non-production (quantity one) system. The common element detected in the above approaches, off-loading the time-critical processing, was incorporated into the modified interrupt-driven mechanism developed to meet the system timing requirements. The hardware design for this mechanism is discussed briefly below, with more detailed block diagrams and design notes describing critical aspects of the design being included in Appendix C. Appendix D contains the Hardware Signal Dictionary, where the signals used in the designs shown in Appendix C are defined. Appendix E contains the detailed timing diagrams which prove that the 1553BBI design meets the timing constraints imposed on the subsystem by the RTU chip set. Section 6.3, 432/670 Avionic Subsystem Concept of Operation, describes the relationships and interactions between the system hardware and software, as well as discussing the system operation from the perspective of the 1553B Bus Controller.

The coupling of the basic interrupt-driven data transfer mechanism with message handling hardware gives the AP-RTU Module the ability to meet the strict timing requirements which the RTU levies upon the subsystem. As can be seen in the structure chart in Figure 20, with this approach the AP-RTU Module functions are assigned to two principal

modules: the Multibus I/O Module and the RTU Support Module. As the name implies, the Multibus I/O Module is responsible for performing all functions required to interface a peripheral device to a standard Multibus. As discussed in the previous analysis, due to speed considerations, memory-mapped input/output and a non-bus vectored interrupt scheme are necessary. Four submodules are used in the interface design. The Multibus Decoder Module decodes the AP's address and control signals and from them generates the function signals called AP Commands (AP CMDS), which comprise one of two sets of control signals which drive the RTU Support Module operation. The Data Transfer Module buffers the 16-bit DAT0 - DATF data bus which extends from the Multibus interface to the RTU Support Module. The XACK Timer Module generates the XACK-L memory transfer acknowledge signal which informs the AP that the 1553BBI has accepted incoming data or that stable data is available on the data bus for reading. The Interrupt Circuit Module provides non-bus vectored interrupt handling services for the interrupts available to the 1553BBI. This scheme requires that the module latch the appropriate line low until the AP signals compliance with an interrupt acknowledge, sent in the form of an AP CMD. A maximum of eight prioritized interrupts can be controlled by the AP's master 8259A Programmable Interrupt Controller (PIC), since when operating in this mode, the PIC directly generates the service routine addresses. Four of these interrupts are reserved for supporting the 432/670 IP, the system clock timer and the serial input/output for the optional terminal. Proper assignment of the four interrupts allocated to the 1553BBI is critical to the operation of the AP1 Subsystem, since only through these four

signals can the AP monitor the RTU Support Module service requests and operational status.

The RTU Support Module performs all of the time critical message handling functions for the subsystem, including decoding 1553B command words and controlling the data transfers across the subsystem/RTU interface. The module provides the local storage capability required to support the high speed data transfers, as well as providing an error handler used to notify the AP of non-1553B message handling errors and the BC of 432 Avionic Subsystem problems. These functions are assigned to a set of five submodules. These submodules include the In Message Storage Module, responsible for storing incoming messages, the Out Message Storage Module, responsible for storing outgoing messages and message identifiers, the Command Handler Module, which decodes the 1553B command words and generates a set of control signals known as RTU Commands (RTU CMDS), the Strobe Handler Module, which generates all the properly synchronized external (to the RTU) timing signals and the Error Handler Module which provides the error handling capability mentioned above. Since the design of this set of modules is intimately related to the 432/670 Avionic Subsystem concept of operation, a more detailed description of the role each is assigned relies heavily upon an understanding of the system operation. Further discussion is therefore deferred to the following section, where the message handling function of the subsystem is discussed.

6.3 432/670 Avionic Subsystem Concept of Operation

The goal of this design effort is to design and analyze a general purpose interface between the MIL-STD-1553B data bus and the 432/670

Computer System which can be adapted for use in a variety of avionic system applications. To provide maximum flexibility for supporting these unspecified applications, complete hardware and software design independence is desirable. Unfortunately, the speed and synchronization requirements which necessitated assigning the message handling functions of the API Subsystem to the hardware made this goal unrealizable. As a result of this redefinition of the subsystem architecture, the subsystem concept of operations, whose development was originally intended to be an integral part of software design, became a driver for a major portion of the RTU Support Module design. Upon analysis, the impact of having the concept of operation drive a portion of the hardware design, instead of merely reflecting constraints imposed by the hardware, was found to be minimal. The concept of operation presented in this section effectively ties together the related portions of the software and hardware designs.

The 432/670 Avionic Subsystem operation is best understood by following the processing of a data message as it flows through the 1553B bus system. The normal message processing flow is summarized in the essential process model shown in Figure 25. The message processing begins in the 1553B BC which is responsible for controlling all message traffic on the bus. The BC is also responsible for translating information being routed between subsystems which use incompatible data word formats. In order to maximize bus performance and compensate for the different operational requirements of the subsystems, portions of the BC software must be customized for each bus implementation. The BC software/data tables must include information concerning subsystem

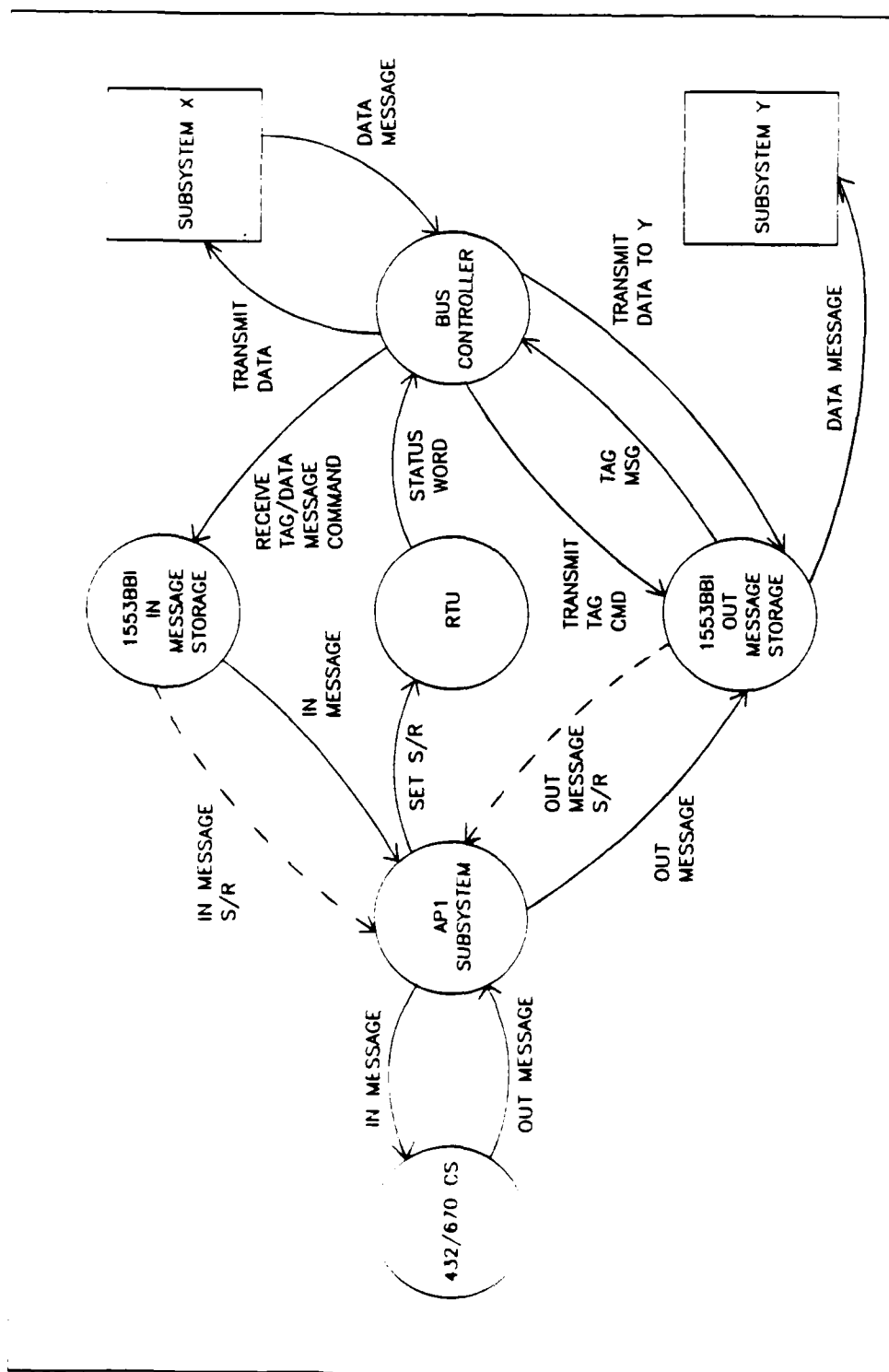


Figure 25. Essential Process Model

polling sequences, polling intervals and data routing. In order to monitor subsystem status and provide data translation services, built-in-test (BIT) word formats and subsystem peculiar data formats must also be provided. Complex subsystems may impose additional requirements on the BC software. The 432/670 Avionic Subsystem, whose function is solely to furnish data processing support to other bus residents, requires external process management control which can be provided only by the BC.

The key to designing efficient BC executive software is identifying and capitalizing on the cyclic nature of subsystem service requirements. The majority of the BC resources are dedicated to regulating major and minor bus cycles, monitoring subsystem status and responding to subsystem service requests. Resources to support processing of memory intensive ancillary functions, such as process management for a particular subsystem, may not be available. Subsystems whose operation depends upon frequent unscheduled service requests are often insupportable since disruptions to the normal bus cycles can seriously degrade system performance. The process control management scheme for the 432/670 Avionic Subsystem therefore must be a simple, low overhead process that operates on a non-interference basis with other BC executive functions. The process management scheme outlined in this section fulfills these requirements and, if implemented correctly, should be easily supported by the BC.

The 432/670 process management scheme requires that all data messages being sent to the 432/670 Avionic Subsystem for processing be routed through the BC for classification. Based upon the source and any

additional identifying information, the BC determines the type of processing required, extracts data and reformats the message data as necessary and assigns a unique process identifier called a tag word to the message. The tag word contains three fields, a 5-bit message word count field, a message continuation bit and a unique 10-bit process identifier (PID). The PID uses several bits to identify the type of 432/670 CS processing desired. The exact number of bits required for this purpose depends upon the number of programs the 432/670 CS must host to support the application. The remaining bits function as a supplemental identifier to allow a unique PID to be assigned. The format of this portion of the PID can be selected so as to convey additional information, such as a time stamp, to the BC.

Priority-FIFO process scheduling is used by the management scheme. The process priority must be passed to the 432/670 Avionic Subsystem via the PID. Priorities can be associated with the program numbers if all requests for the same type of processing have equal priority. Otherwise, the priority must be encoded in the supplemental identifier in some manner. After classifying a message and generating its tag word, the BC combines the two and adds the resulting message(s) to the queue of outgoing 432/670 messages. Two methods of message/tag word handling are available, depending upon the length of the message. For messages of 31 words or less, the data words are appended to the tag word to form a new message, while for 32-word messages, the tag word must be transmitted as a separate message. Whenever the tag word is transferred separately, the tag word message must be transmitted first. Some types of commands may be sent to the 432/670 prior to the

subsequent transmission of the data message; however, pre-emption by higher priority requests after tag word transmission is not permitted. Neither are commands which alter the system status, unless an error condition requires system reset. In this case, the tag word must be retransmitted. This single and double message handling scheme is completely transparent to the 432/670 Avionic Subsystem.

The BC uses the standard 1553B protocol to transfer the data messages to the 432/670 for processing. The BC sends a receive command and its associated data words to the 432/670 RTU. In the two message case, the tag word message contains only the one data word. All functions required by the MIL-STD-1553B protocol, including all message error checking, mode command responses except for vector word transmission, and status word transmission, are handled internally by the RTU and require no subsystem intervention. The subsystem involvement begins when the RTU Support Module detects a VAL CMD WD RC-L (Valid Command Word Received) signal. This signal, intended for internal RTU use, triggers the RTU Protocol Sequencer's latching of the valid command which is available for 500 nanoseconds on the internal T0 - T15 data highway. This signal is used by the subsystem to initiate the RTU Support Module message handling mechanism.

The Command Handler Module controls all subsystem involvement with the RTU. The Command Handler Module monitors the VAL CMD WD RC-L signal, and latches the command word from the T0 - T15 bus when the signal goes low. The command buffer output, which is continuously available until updated upon receipt of the next valid command, is decoded to provide a set of control/enable signals to the remainder of the RTU Support

Module. The T (Transmit) and R (Receive) signals are derived from bit 10. The Subaddress Decoder generates subaddress signals SUB A, SUB B, SUB C, SUB D, MODE (Mode Command, bit patterns 00000 or 11111), and WRAP AROUND CMD (Wrap Around Command, pattern 11110) from bits 9 through 5, the command subaddress/mode field. Taken together, all of these signals except WRAP AROUND CMD are considered the RTU CMDS. SUB A through SUB D are implementation dependent bit patterns used to enable data word receipt and transmission, and transmission of the old tag word, the old message and the contents of the error registers, respectively. MODE is used only in conjunction with the EN VEC WD-L (Enable Vector Word) signal to transmit the current tag word; all other mode commands are ignored. WRAP AROUND CMD is provided to allow the BC to initiate a wrap around test sequence if the normal hardwired wrap around enable is not desired. This signal sets the WRAP EN (RTU Wrap Enable) SS-RTU CONTROL (Subsystem to RTU Control) signal. In addition to its primary function of generating the subsystem control/enable signals, the Command Handler also allows AP control of WRAP EN, provides command word access to the AP upon receipt of an AP RD CMD (AP Read 1553B Command Word) and passes the CMD WD CNT (Command Word Count, bits 4 to 0) to the other RTU Support Submodules.

Incoming data messages are transferred from the RTU's FIFO to the In Message Storage Module under the control of the EXT SO-L (External Strobe Out) receive strobe from the Strobe Handler Module. Strobe timing is critical since data words must be transferred from the RTU's FIFO to the subsystem at a 2.0 MHz rate. Since the Strobe Handler is responsible for generating both the EXT SO-L and the SSUI STRB-L

(Subsystem Interface Unit Strobe) transmit strobe, a series of gates is used to ensure correct generation and routing of the strobes. The R and NOT MODE signals provide the control for the 2.0 MHz EXT SO-L, which is derived from the RTU's 16 MHz clock when triggered by the RTU's DATA STRB-L (Data Strobe) signal. An additional gate is used to stop the strobe generation when the requisite number of pulses has been sent. This gate is opened by the trailing edge of the 8.5 microsecond CMD STRB-L (Command Strobe) which is output by the RTU upon the latching of the last data word by its FIFO. Use of the CMD STRB-L to open the gate protects the subsystem from inadvertent transfer of incomplete messages since the signal will only be generated when a complete, valid message has been successfully loaded into the FIFO, terminating the RTU's receive sequence. The MIN/MAX (Minimum/Maximum Count) signal of the up-down counter, which was loaded with the CMD WD CNT when the CMD STRB-L arrived, is used to close the strobe gate. The EXT SO-L signal is sent to the RTU where it is used to enable the data words onto the T0 - T15 Data Highway and to the In Message Storage Module where it controls latching of the words into its In FIFO Buffer.

Data reception within the In Message Storage Module is enabled by SUB A and triggered by EXT SO-L. The tag word and its associated message are stored sequentially in the next available locations in the module's 16-bit wide FIFO buffer. This FIFO not only allows the subsystem to meet the RTU's timing requirements, but also provides the extra storage capacity necessary to support a relatively high processing volume. This additional storage allows most of the limited storage resident on the AP board to be used for outgoing message storage and alleviates storage

contention problems on the AP. Message transfer to the AP is a two step process which is interrupt-controlled to minimize the backlog of messages. Rapid transfer is necessary since, between the time the BC sends the message and the time the data is transferred to the 432/670 CS, process scheduling is strictly FIFO. The In Message Storage Module communicates its status to the AP via two interrupts. The IN BUFFER FULL (In Buffer Full, Interrupt 1) interrupt is a high priority interrupt used to inform the AP that no more messages can be accepted. The IN BUFFER FULL is one of several error conditions which must be reported to the BC; the reporting mechanism is discussed later. The IN MSG S/R (In Message Service Request, Interrupt 3) interrupt is used to notify the AP that messages are awaiting transfer. This interrupt is generated whenever the input buffer is not empty. This is the highest priority non-error related interrupt available to the subsystem. The AP responds to the interrupt by issuing a AP RD MSG (AP Read Message Word). This signal is generated by the Multibus Decoder Module and is, as are all of the AP CMDS, derived from the combination of the standard signals which control Multibus memory-mapped input/output (MRDC-L, MWTC-L) and the associated address. The AP RD MSG signal enables the word in the first location of the buffer onto the AP-RTU Module's DAT0 - DATF data bus; the Multibus I/O Module controls the rest of the transfer as required by the Multibus protocol.

The AP is responsible for message pre-processing and for initiating the transfer of data to the 432/670 CS. In response to the IN MSG S/R interrupt, the AP reads the first word in the buffer and verifies that the word is a tag word. The verification mechanism, which necessarily

provides only a limited error detection capability, involves ensuring that the tag word structure and the PID conform to the application-dependent set of rules governing its generation. Invalid tag words flag an error condition. Following the reading of a valid tag word, the AP extracts the message word count and reads all associated data words from the FIFO by issuing the appropriate number of AP RD MSG commands. To compensate for any delays in transferring the data message when the tag word is transmitted separately, the AP will not initiate the data word transfer loop until a second IN MSG S/R interrupt is received. After the message transfer is complete, the AP moves the tag and message data words to a block of memory reserved for incoming message storage. Transfer of subsequent messages from the 1553BBI is inhibited until the message has been moved to temporary storage, at which time the IN MSG S/R interrupt is cleared.

Transfer of the message from the AP to the 432/670 CS is handled by a set of message transfer tasks using the 432 IP window management facility. This facility allows the AP to treat the transfer as memory-mapped I/O writes (and reads) to the reserved IP memory space. Object-oriented operations are simultaneously supported on the 432/670 CS side. Monitor tasks executing on both the 432/670 CS and the AP continuously poll the message transfer status to determine when a message is waiting to be read or when the window is free for the next write operation, respectively. Since priority management is not warranted for the relatively few incoming messages in the temporary AP storage, transfers to the 432/670 CS are handled in a strictly FIFO manner.

The 432/670 CS message processing is initiated when the incoming message monitor task detects the arrival of a new message object in the IP window. Space is allocated for the message object and the tag word is decoded to determine message priority. The message is then added to a priority/FIFO queue to await processing by its application program. The output data object contains the original tag word and all processed (outgoing) data words. Since no restrictions are placed on the number of data words which can be generated by the processing of an incoming message, the output data object must be reformatted into a set of outgoing message objects. Each of these objects contains the tag word and 32 or fewer data words. The continuation bit in the tag word is set when multiple output messages are produced, to preclude the BC from flagging an error upon detection of messages with duplicate PIDs. The sequential order of the data words is preserved during the reformatting process and when the outgoing message objects are added to the priority/FIFO queue for transfer back to the AP.

The transfer mechanism for returning the messages to the AP is analogous to that for incoming message transfers. After accepting a message from the IP window, storage is allocated and the message is added to the priority/FIFO outgoing message queue. The AP uses a low-overhead dynamic memory management scheme based on linked-lists to manage the limited amount of memory available for output message storage. With this scheme all of the AP memory allocated to output message storage is divided into 34-word blocks which contain the tag word, 32 words for data storage and a pointer to the next message. Blocks are assigned to either the free block table or to one of the

priority message tables, depending upon their current status. A separate priority message table is used for each message priority level allowed by the application. This scheme provides an efficient memory management mechanism and serves as the basis for the priority-FIFO process management for the output message queue.

Messages in the output message queue cannot be directly transmitted to the BC due to the stringent timing constraints imposed upon the subsystem by the RTU. The transfer mechanism chosen for data message transmission involves several steps. Following transmission of the previous message, the Out Message Storage Module sends an OUT MSG S/R (Out Message Service Request, Interrupt 5) interrupt to the AP. The AP writes the tag word associated with the next message to the 16-bit Tag Buffer in the Tag Word Buffer Module. The AP then sends an AP CL MSG (AP Clear Message Buffer) command to clear the 32-word Out Message Buffer and writes the message to the buffer using a series of AP WT MSG (AP Write Message Word) commands. Clearing the buffer before writing is a safety measure which prevents problems that might arise if the FIFO is left in an unknown state following transmission of a message of less than 32 words. After the message is written to the Out Message Buffer, the AP sets and latches the S/R-L (Service Request) line which in turn sets the S/R bit in the RTU status register. After the next status word transmission (either following a command or in response to a transmit status mode command), the BC responds to the request by sending a transmit vector word mode command to the 432/670.

The transmit vector word command is decoded by the Command Handler as are the normal receive and transmit commands, but instead of a CMD

STRB-L, an EN VEC WD-L signal is generated by the RTU. MODE and T act as enables for the tag word transmission function, for which the EN VEC WD-L signal serves as the trigger. The resulting RD TAG WD-L (Read Tag Word) signal gates the tag word onto the T0 - T15 Data Highway and latches the word into the Old Tag Buffer which is part of the transmission error recovery mechanism. Upon completion of the vector word transmission, the BC decodes the tag word, determines the number of words to be transferred and, based on its data tables, selects the appropriate routing for the message. If no translation of the data is required, the BC issues an RTU to RTU transfer command, otherwise, an RTU to BC command is sent. The subsystem handles both types of transfers in the same manner, since, at the subsystem interface, RTU to RTU transfers are indistinguishable from transfers routed through the BC.

The transmit command processing parallels that of the receive command. After command decoding, SUB A and the SSIU STRB-L are used to clear the S/R-L latch and to control the unloading of the Out Message Buffer. In order to meet the RTU's synchronization requirements, the Strobe Handler Module derives the SSIU STRB-L from the 1.0 MHz Protocol Sequencer clock. T, NOT MODE and VAL CNT (Valid Transmit Command Word Count) serve as the strobe generation and routing controls. The VAL CNT signal, generated by the Tag Word Buffer Module, is used as a cross-check to ensure that the number of words the system is directed to transmit is equal to the number actually in the buffer. If necessary, CNT ERR (Transmit Command Count Error, equivalent to NOT VAL CNT) flags an error to the BC via the AP, informing the BC that transmission of the

wrong message was requested. CMD STRB-L is again responsible for opening the strobe gate. The gate is controlled in the same way as for a receive command, except that a delay in opening the strobe gate is needed to ensure that the first strobe does not occur before the 1.0 microsecond minimum time. As with the tag word, backup buffer storage is provided for the message itself. The Out Message Buffer unload control also serves as the load control for the Old Message Buffer, which was previously reset by CMD STRB-L.

The Old Tag Buffer and Old Message Buffer play a critical role in error recovery for the subsystem. Backup storage on the AP is impractical since managing such a retransmission request would be a relatively complex and time intensive task. The addition of backup storage on the 1553BBI provides a simple solution to the problem. Since the subsystem controls the message transmission via SSIU STRB-L, the transfer to the backup buffer can be forced to completion even when a transmission error occurs. In this case, the BC can request retransmission of the tag word by sending a one word transmit command using subaddress B. After decoding the old tag word, the BC sends the transmit command using subaddress C. Except for being controlled by a different set of enables (SUB B or SUB C), the entire sequence, including SSIU STRB-L generation, is handled in exactly the same manner as was the original transmit. The Old Message and Old Tag Buffers will hold the message indefinitely as long as the BC does not send an intervening normal transmit command (SUB A, T), which clears and overwrites the old buffer contents.

The BC needs subsystem status information to effectively control bus operation. Each RTU must have BIT and 1553B protocol error management capability. Since the 432/670 Avionic Subsystem is intended as a general support system, additional error and status information is required to aid the BC in its process management task. The error management scheme in this design is flexible, to allow easy modification and extension to support a variety of applications. The 432/670 RTU handles all 1553B protocol and RTU hardware related errors without subsystem intervention. The RTU Support Module hardware and the AP software both flag process management related errors, and depending upon the implementation, may also be able to detect some subsystem hardware errors as well.

The Error Handler Module is used to inform the BC of the subsystem errors. The module contains two 8-bit registers, an AP1 Subsystem hardware error register, (SS ERR Register) and an AP error register (AP ERR Register). Up to eight AP1 Subsystem hardware errors can be handled without incorporating an encoding scheme into the system. Errors such as CNT ERR are latched individually by the module, which then generates the high priority SS ERR (Subsystem Error, Interrupt 0) interrupt to the AP. The AP responds by sending an AP WT SS ERR (AP Write Subsystem Error Register) signal to write the individual latch contents into the SS ERR Register. For an AP initiated error sequence, such as for In Buffer Full, the AP writes an 8-bit error code to the AP ERR Register, using the AP WT AP ERR (AP Write AP Error Register) signal. Following either of these two commands, the AP sets and latches the SS FLAG-L (RTU Subsystem Flag) input to the RTU status register, which sets the SS FL

bit in the RTU's status register. Upon detecting SS FL, the BC reads both error registers by sending a transmit command to subaddress D, which is handled as described above. Transmission of the error registers clears the individual hardware error latches. The AP monitors SS FLAG and clears the SS ERR Register by sending another AP WT SS ERR command after the latches have been cleared. The AP also uses its monitoring capability to determine when and if to update error codes in the AP ERR Register. Since codes which reflect multiple error conditions can be devised, conditional update of the register contents may be desirable.

A wide variety of error coding schemes can be implemented using this design. For example, the simplest way to handle an outgoing message storage bottleneck on the AP is to allow the effects of the overload to ripple back. The 432/670 CS would hold its output data objects while continuing its normal processing. Upon exceeding its storage capacity, no further incoming messages would be accepted. The In Message Buffer would continue to accept messages until full, at which time an In Buffer Full interrupt is generated and the BC is notified of the problem by the error handling mechanism. Incoming transmissions would then be halted until the error condition is cleared. Early flagging of the problem could easily be implemented by monitoring the status of the various storage bottleneck points and by devising an appropriate set of error codes to be written to the AP ERR Register. The flexibility inherent in the AP error handling mechanism is due to the ease of modifying the subsystem software. The hardware error handling mechanism is likewise extensible. If more than eight hardware

error codes are required to adequately convey the subsystem status to the BC, more individual latches and a simple encoding circuit can be added to the Error Handler Module.

6.4 Hardware Design Analysis Summary

The 432/670 Avionic Subsystem hardware design and concept of operations discussed above meet all requirements stated in Chapter 4. As discussed in Section 6.1, the high level system concept meets all of the global requirements. Two significant points must be made with regard to the analysis, however. In evaluating the requirement that the system be cost-effective, the utility of integrating the 432/670 computer into an avionic system is not considered, since the project proposal stipulated the use of the 432/670. The utility of implementing the system is addressed in the conclusions in Chapter 8. Likewise, the simplicity of the overall concept, and not the 432/670 system itself, is used to determine compliance with the requirements.

The system level design of the 1553BBI likewise meets the global requirements. The selection of a smart, industry-standard, highly reliable RTU chip set is instrumental in allowing the requirements for simplicity, reliability and maintainability to be met. Again, primarily due to judicious selection of the RTU chip set, all but one system level requirement is satisfied by the 1553BBI design. The RTU automatically handles all 1553B protocol requirements, essentially moving the design interface to the boundary between the RTU and the AP-RTU Module. The strict timing and synchronization requirements imposed by the chip set at this, the subsystem, boundary are met by the RTU Support Module hardware. A thorough analysis of the timing requirements and the design

timing was performed to verify that the design does all of these critical timing and synchronization requirements. The timing diagrams and analysis are presented in Appendix E. The chip set also meets or exceeds all of the 1553B electrical requirements, as well as meeting the requirements for Multibus power compatibility, low power consumption and small size (single board implementation possible). The Multibus I/O Module was designed to Multibus interface specifications, and does in fact conform to the Multibus interface specifications. The AP-RTU Module design uses only standard, readily available, relatively inexpensive components. Chip functions were identified rather than specific chips to allow optimal selection based on cost and the availability of military standard implementations. The complete design meets all 1553B and Air Force Notice 1 requirements, and is based upon the AFIT 432/670 computer system configuration. The only DAIS-related requirement not met is that to support its non-standard BIT word format. The disadvantages inherent in the use of a dumb RTU implementation far outweighed the ability to support variable BIT formats, especially since bus controller software can be, and generally is, used to compensate for this type of format incompatibility.

The overall 1553BBI hardware design is general in nature, being able to meet the needs of a number of different avionic system applications, while still providing the implementer the flexibility to incorporate any special features desired. Since all of the time-critical message handling functions of the subsystem are implemented in hardware and since memory-mapped input/output is supported, the 1553BBI design has no processor-dependent features. When implemented, the

1553BBI board could easily be hosted by any Multibus-based processor subsystem. The design is also consistent with Air Force system support methodology, which emphasizes on-equipment hardware maintenance and software configuration modification.

CHAPTER 7

SYSTEM SOFTWARE DESIGN

7.1 Introduction

The software required to provide the functionality discussed in the Section 6.3, the 432/670 Avionic Subsystem Concept of Operation, is quite complex. The system software executes on two different processor subsystems, under the control of separate, user-customized operating systems. Special software is required to control the windows which are the sole means of communication between the two subsystems. The application, or user software, includes drivers for the 1553B Bus Interface (1553BBI) hardware, the 1553B message management software resident on both subsystems and the application programs which perform the actual message data processing. Due to the nature of the computer system architecture, an exceptionally high degree of cohesion is required between the user software and the respective operating systems. Any software design effort must therefore address both user software and operating system design.

The primary objective of the system software design effort is to provide a solid framework upon which a follow-on detailed design effort can be based. Two major factors influenced the selection of the design presentation tools. Identifying the essential components, including operating system modules, communication support packages and user software, and determining the physical organization of the system are critical aspects of this design process. Unfortunately, the nonstandard

multi-processing/multitasking 432/670 system architecture does not lend itself well to standard presentation methods. As discussed in Chapter 5, some liberties were taken with the standard Yourdon design methodology in order to provide a complete, clear design. Hierarchy charts were used in conjunction with the data flow diagrams at both the top and system design levels. Structure charts, which are more applicable to sequential processing systems, were not used at either of these two design levels since both exhibit the numerous multiprocessing/multitasking characteristics of the software. The lack of symbols for portraying these characteristics and for identifying software packages requiring user customization led to the introduction of several new graphics symbols for use in the data flow diagrams and hierarchy charts. The symbol conventions are shown in Chapter 5 (Figure 18), where the software design methodology is discussed in more detail.

7.2 Top Level Design

The primary function of the 432/670 Avionic Subsystem is to process data for subsystems on the bus which lack the 432/670's computational power. As shown in Figure 26, the software context diagram, the data processing software perceives the 1553BBI to be the source of the 1553B data messages. The 1553B hardware characteristics and protocol are essentially transparent to both the 432/670 CS and the AP, since the 1553BBI's RTU Support Module handles all of the time-critical message transfer functions and presents a much simpler interface to the 432/670 software. Access to the data messages is provided through a memory-mapped input/output scheme. Control signals generated by the software drive the interface and status signals allow for continuous monitoring

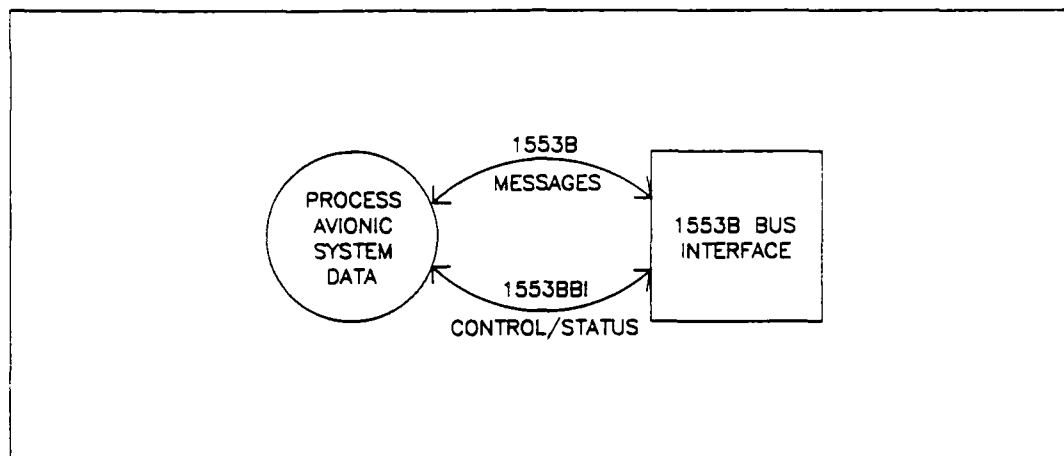


Figure 26. Software Context Diagram

of 1553BBI activity and error conditions. Despite the simple, direct nature of the interface between the 1553BBI hardware and the Avionic Subsystem software, the software itself is quite complex. This complexity is dictated by the architecture of the 432/670 architecture, which precludes direct connection between the CS and its peripherals.

The introduction of a peripheral subsystem for performing all input/output operations for the 432/670 CS adds an extra set of functions to the subsystem software. The top level data flow diagram, shown in Figure 27, identifies four primary software processes: system initialization, reception and transmission of 1553B messages, transfer of the messages to and from the 432/670 CS and the actual data processing. The reception and transmission of the 1553B messages are actually performed by the RTU Support Module hardware; it is included as a process in this diagram to emphasize that, although actually implemented in hardware to allow the timing requirements imposed by the RTU chip set to be met, these are normally software functions. The

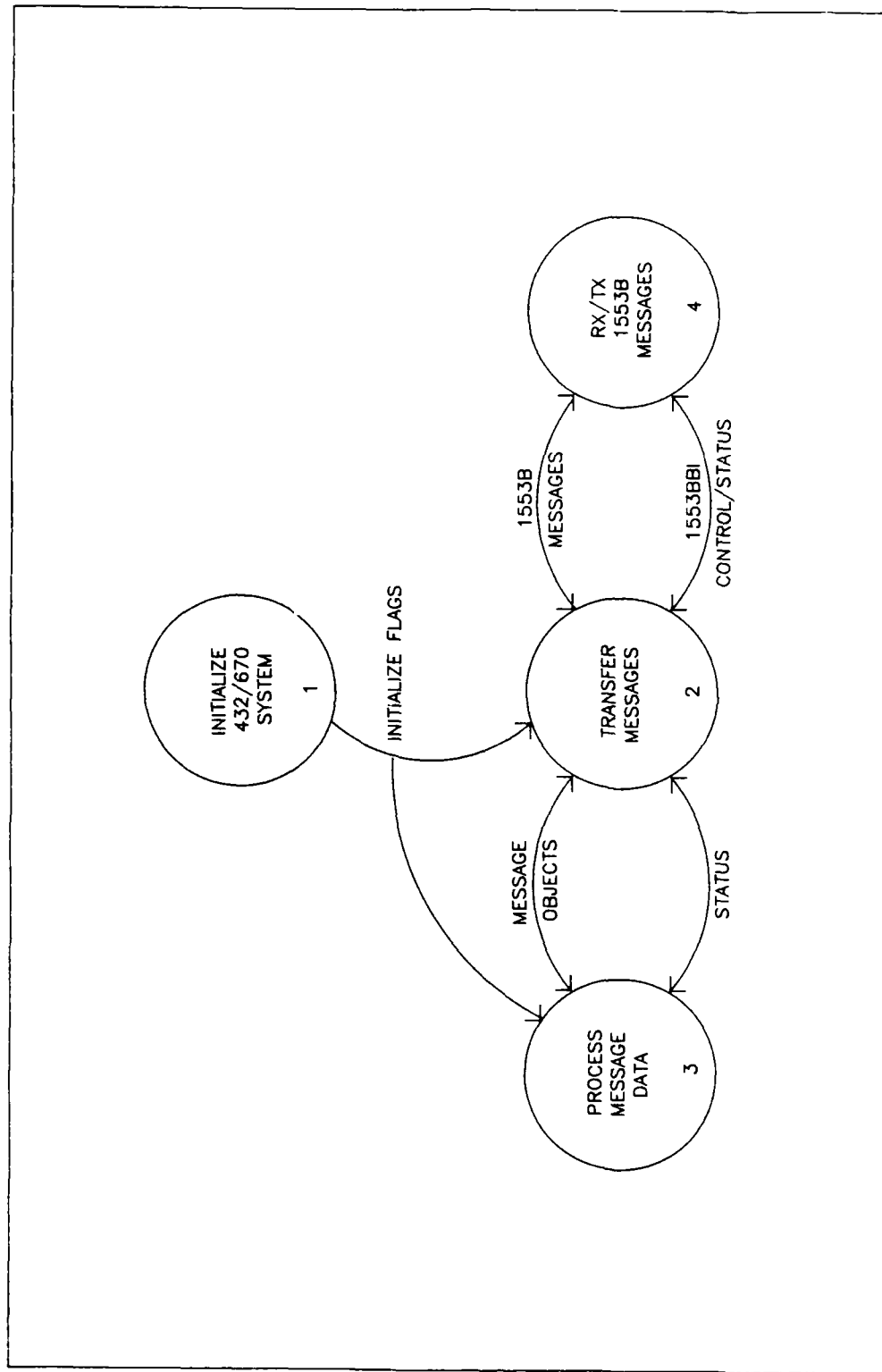


Figure 27. Top Level Software Data Flow Diagram

transfer messages process encompasses all transfers required to move the messages between the 1553BBI hardware and the application programs hosted on the 432/670 CS. The message data processing, the most straightforward of the software processes, involves decoding of the tag word associated with the data message to determine the appropriate application program, and starting the actual application process. The system initialization process, like the message transfer process, is quite complex. Separate initialization of the 432/670 CS and AP1 hardware is required, and all processes resident on both systems must be explicitly started. The initialization process involves a combination of interrelated hardware, system software and user software functions, as does the transfer process. As shown in the top level software hierarchy chart in Figure 28, the physical division between the system and user software is likewise obscure at the two highest level.

The 432/670 Avionic Subsystem software is basically organized by subsystem residency. The Debugger software is supplied with the cross-development system and is only marginally of interest in this design effort. The Debugger software provides tools which allow the user to interactively control system loading and start-up, and to observe and control program execution. The AP1 Subsystem software, hosted by the AP1 peripheral subsystem, consists of an executive and input/output (I/O) controller software which performs all data transfers between the 1553BBI and the 432/670 CS. The 432/670 CS software is composed of the user-customized iMAX operating system and all application processes defined by the user. This software is discussed in greater detail in the system level design which is presented in the following section.

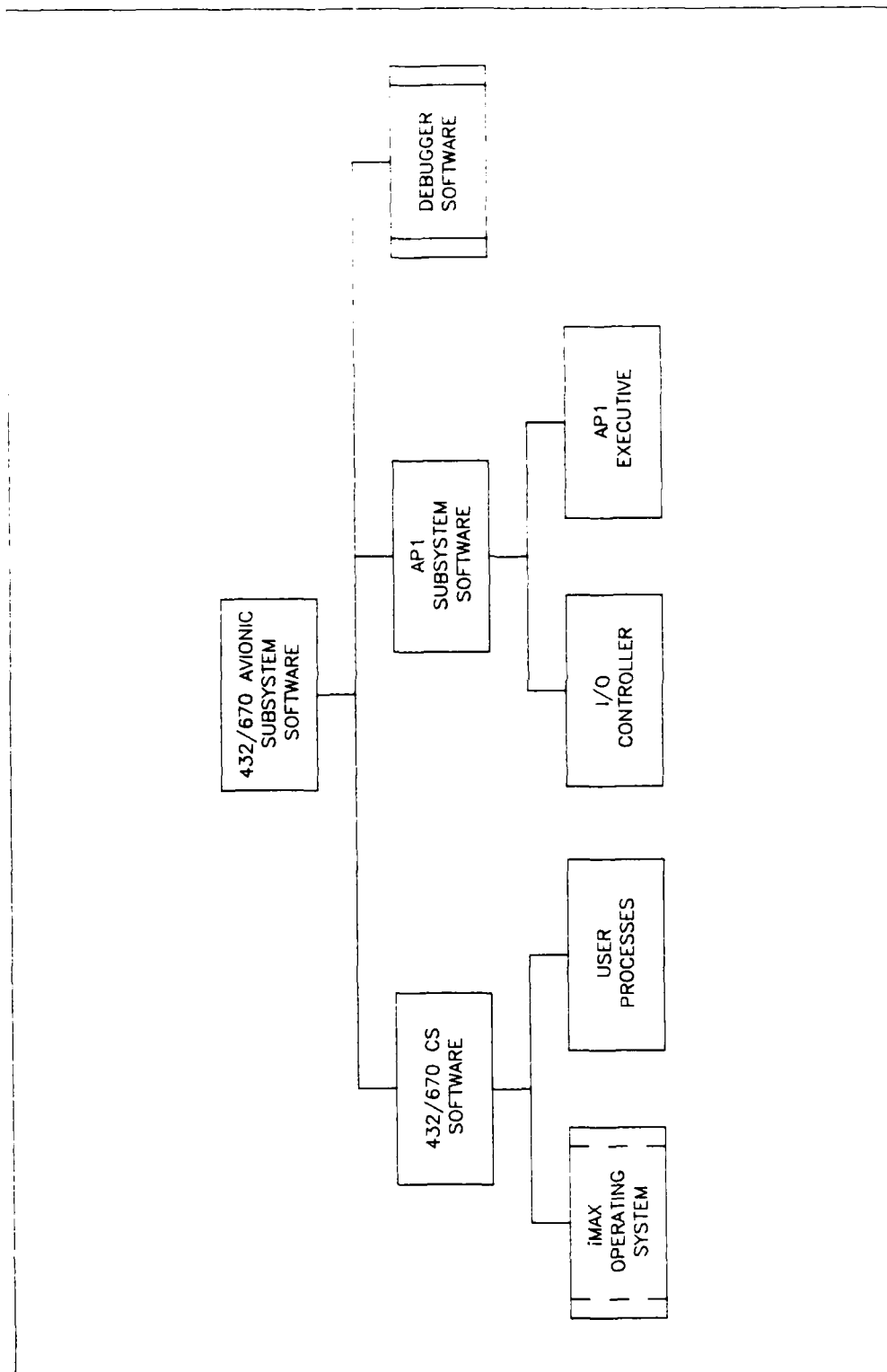


Figure 28. Top Level Software Hierarchy Chart

7.3 System Level Design

The system level design of the 432/670 Avionic Subsystem software involves the design of two user software/operating systems configurations which execute independently on the separate processor subsystems. Connectivity between the 432/670 CS processes and AP1 processes is obtained through AP-resident support software provided with iMAX. This support software cooperates with iMAX, providing an asynchronous communication link between the subsystems via the AP1's IP windows. Since the nature of this communication link drives much of the system software design, the transfer mechanism, as perceived by the software, is discussed briefly below.

The software which drives the flow of data through the IP windows effectively masks the details of each subsystem's operation from the other. Since the 432/670 CS perceives an IP window as an asynchronous communication port, the standard interprocessor communication facility supported by iMAX is used for sending and receiving the required data object references. The actual control of the data transfer is provided by the IP's window management facility, which is provided in the iMAX AP Support software package. This software essentially extends the AP's instruction set through the addition of IP control facilities. The IP thus resides both physically and logically in the AP subsystem. The IP is assigned a contiguous block of up to 64K bytes in the AP's unpopulated memory space for use by its five windows. Each window is allocated a set of addresses, called a subrange, within this 64K byte block. The AP is then able to access the windows through memory-mapped input/output operations. Memory addresses are mapped into objects by

memory writes to addresses in a window's defined subrange. The CS passes data object contents to the AP by writing the object reference to the port object associated with the window, with the AP subsequently reading the data via a sequence of memory reads. This mechanism also supports operations on object refinements, that is, it allows read and update access to particular fields within the object (Ref 13:1-4 - 3-4). This mechanism is particularly useful in passing transfer status information between the CS and AP.

The system level data flow diagram shown in Figure 29 illustrates how the system software controls the flow of the 1553B messages between the 1553BBI and the 432/670 CS. This diagram supplements the hardware-oriented subsystem essential process model (Figure 25, Chapter 6), which served as the basis for the 432/670 Avionic Subsystem concept of operations. As with the process model, in order to provide a relatively simple, clear presentation of the design, some details of the software operation must be simplified or completely omitted from the data flow diagrams. The impact of these simplifications, which are described below, is minor, since they involve operational mechanisms which are transparent to the user software.

Data transfers across the CS-AP subsystem boundary are shown from the perspective of the CS, since objects are more easily designated than memory references. This convention is adopted with the understanding that the transfer process portrayed in the data flow diagrams is an over-simplification of the actual transfer mechanism which was discussed previously. Similarly, since asynchronous communication via ports and mailboxes is a standard means of communication between processes (CS)

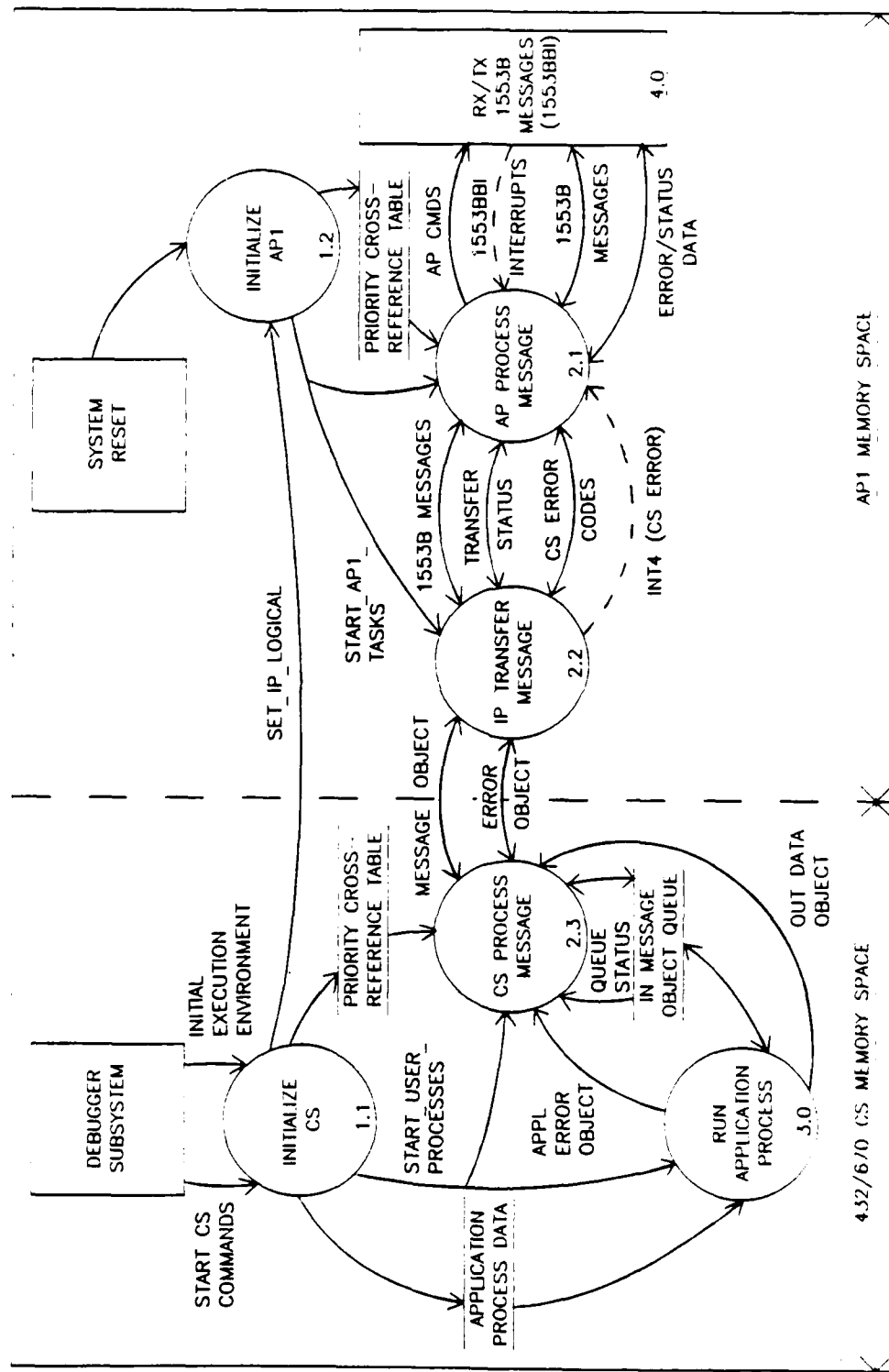


Figure 29. System Level Software Data Flow Diagram

and tasks (AP), respectively, all system-managed ports and mailboxes are omitted from the data flow diagrams. The priority/FIFO queues are shown since they must be managed by user software. Two conventions relating to the decomposition of the data flow diagrams were also adopted. Since all CS static processes and AP tasks are started only during the system initialization, the respective startup flags are shown at the system level, but are not propagated down into the lower level design diagrams which involve only message processing functions. For a similar reason, composite data flows are normally shown at the higher levels, while only the individual components of the composite flow are shown in the lower level diagrams. The names of all composite flows and their components were chosen to facilitate tracking during the design decomposition. Appendix H contains a data dictionary in which the contents of the composite flows are defined.

The system level data flow diagram, shown in Figure 29, presents a composite of the first level decompositions of processes shown in the top level data flow diagram. The relationships between the significant system processes are more apparent in this diagram than in a more typical set of diagrams in which each process in the top level diagram is expanded individually. Decompositions of Run Application Process (3.0) and RX/TX 1553B Messages (4.0) are not included since their lower level processes are actually equivalent in detail to the other third level processes. Note also that on the system level diagram RX/TX 1553B Messages (4.0) is shown appropriately as a source/sink since the software functions are handled by the 1553BBI hardware.

The system level software processes fall into two primary categories: initialization processes which occur at system reset and are subsequently suspended, and the operational system processes. Separate initialization processes are required for the CS and the AP1 Subsystem. The Initialize CS Process (1.1) is actually composed primarily of hardware functions executed under Debugger control. The CS hardware must be initialized, the initial execution environment loaded into memory and the Debugger IP set into the logical addressing mode prior to system startup. Under normal operational conditions, after the Debugger issues the start command, the remainder of the process occurs without operator intervention. The first GDP is started, the initial LMAX data structures built, the remaining GDPs and AP1 IP started, then the static user processes are started. The user process which builds the Priority Cross-Reference and Application Process Tables, which contain the application program execution priority and application program startup information, respectively, is also included in the process. The Initialize AP1 Process (1.2) runs independently of the CS process. This process, which is initiated by the hardware system connected to the LPMX 88 operating system, initializes the AP1 hardware, the LMAX AP Support functions, builds the AP's version of the Priority Cross-Reference Table, and initializes the message storage data structures before starting the AP user tasks. The only communication between the CS and AP1 during initialization is through the hardware Set_IP_Logical signal which enables logical addressing for the AP1 IP's windows. Following completion of the initialization processes on the two subsystems, these processes are essentially suspended. No provision

for further operator intervention is included in the design, although communication through the optional operator terminal is not precluded by the design. Since the Debugger software is used in development and maintenance activities, no user software is required to support these functions.

The operational system processes transfer the 1553B messages between the 1553BBI and the 432/670 CS, and perform the actual data processing functions. The AP Process Message Process (2.1) handles all message processing on the AP and collects the subsystem error information which is used to report subsystem problems to the bus controller. This process is responsible for performing the 1553BBI device driver functions, including servicing interrupts and controlling the physical transfer of 1553B messages and error information between the AP memory and the 1553BBI hardware storage. Since the IP Transfer Message Process only performs the actual transfers through the IP windows, AP Process Message also includes polling routines which monitor the status of the window transfers, in order to determine when to actually perform the transfers to and from IP Transfer Message.

AP Process Message provides storage for incoming and outgoing messages, as well as for the current subsystem error codes used by the process in creating the composite error codes sent to the bus controller. Incoming messages are held in a small temporary storage area for subsequent transfer to the IP Transfer Message Process (2.2). This storage space is limited since the 1553BBI In Message Storage Module provides the primary storage. A strictly FIFO mechanism is used for the transfer, since priority management is not warranted for the few

messages supported. For outgoing messages, a priority/FIFO queue is the storage vehicle. This queue is implemented using a set of linked lists. During API system initialization, all memory allocated to outgoing message storage is partitioned into 34-word blocks. Each block holds up to 32 16-bit data words, the tag word and a next-block pointer. Initially, all blocks are placed in a free-space list. Lists are also established for each message priority level. When a message is received from the CS (via IP Transfer Message), an empty block is removed from the free-space list, the message is stored in the block, and its priority is determined using the tag word contents and the Priority Cross-Reference Table. The block is then added to the end of the list holding the messages of that priority level. The queue is serviced by removing the top (oldest) message from the highest priority linked list. This storage scheme was selected due since it uses a low overhead storage management mechanism which directly supports the priority/FIFO queuing. The comparison of the overhead required for this scheme versus a scheme based on variable length blocks was predicated on the assumption that, due to the data processing function of the 432/670 Avionic Subsystem, most messages would be 31 or 32 words long. For similar size messages, using fixed-size blocks requires less memory and processing support.

The IP Message Transfer Process (2.2) transfers the 1553B messages and CS error data across the peripheral subsystem interface. This process accepts incoming 1553B messages in the form of data words from AP Process Message, passes the data through a dedicated window using the mechanism previously discussed, and makes the object reference available

to CS Process Message (2.3). The same process occurs in reverse for outgoing 1553B messages and CS error data. Since four data transfer windows are available, each of the three transfer types is supported by a dedicated window. Transfer status information provides the handshaking necessary to ensure that the source process does not send new data before the current data have been accepted and that the receiving process knows when new data are available. The handshaking is implemented through a transfer status field which is alternately updated by the sending and receiving processes. Consistent with the transfer mechanism, the two processes perceive that status field differently. Transfer of an incoming message from the AP to the CS can be used to clarify the transfer process. AP Process Message monitors the transfer status memory location within the incoming message window address subrange until the code indicating that CS Process Message has accepted the previous incoming message (by transferring the object reference) is detected. AP Process Message then sends IP Transfer Message the addresses of the next group of message words to be transferred and updates the associated status field to indicate that a new message is being sent. IP Transfer Message performs a memory-to-memory transfer (data word addresses to window subrange addresses) for the message words and status fields. On the CS side of the interface, the message words and status field are components of a single message object. This object has a fixed format, containing the tag word, 32 data words, and the status field. By treating the status field as a refinement object, CS Process Message can monitor and separately update the transfer status after detecting and transferring the object reference for newly arrived

message objects. The transfer procedures for outgoing messages and error data follow the same principles. The object structure for all 1553B message objects is identical to facilitate processing by the CS. The error object structure differs, however, containing an error code field instead of a message field. The other significant difference in the error transfer mechanism is that due to the serious nature of CS errors, IP Transfer Message monitors the error status field and generates an interrupt (INT4) upon detection of a new error. IP Transfer Message also updates the status field to indicate acceptance after passing the error code to AP Process Message.

CS Process Message (2.3) performs all of the message management functions for the CS. These include accepting or sending messages and error information through the IP windows, as discussed above, storing message objects before and after processing, and collecting error information to relay to the AP. Message processing by the CS is all on a priority/FIFO basis. After accepting an incoming message object, the object is held in a temporary storage area, while permanent storage is requested and allocated from a pool reserved for incoming messages. The transfer status field in the window is not updated until the object is moved to permanent storage, so that if no storage is available, the flow of incoming messages stops. Object storage is partitioned at system initialization to provide a equitable distribution between incoming and outgoing storage so as to prevent input/output deadlock. The partition size is based upon the expected average ratio of the number of outgoing messages created per incoming message, and is application dependent. Since objects are actually moved by moving their references, the

partition is implemented using two tables. Each table contains a list of the objects whose allocation is charged against its partition. As an object is removed from the incoming queue by Run Application Process (3.0), its allocation is removed from the incoming partition table. Following processing, assuming slots are available, the outgoing data object is reformatted to create a set of 1553B message objects, which are added in consecutive order to the outgoing queue. The references from the resulting objects are then added to the outgoing partition table. Both incoming and outgoing queues are priority/FIFO queues, and are managed in the same manner as the AP Process Message priority/FIFO queue, except that the linked lists consist of object references and next-object pointers. The respective partition table replaces the free block list for each queue.

The incoming message object queue feeds Run Application Process. Run Application Process decodes the message tag to determine the application program to perform the processing, and reads the process information from the Application Process Data table. Run Application Process uses this information to build an instantiation object for the process, which is created dynamically. The application process is then destroyed after its output data object is transferred back to the static portion of Run Application Process. The output data object is subsequently transferred back to CS Process Message for reformatting and addition to the outgoing message object queue. This dynamic process management is used since a change in the mix of incoming messages can be supported without changing the CS initial execution environment. With totally static processing, all process must be declared at

initialization, and only a single instance of the process can exist. This dynamic process-based processing scheme allows for a much more efficient, as well as flexible, maintainable system.

The process flow described above is illustrated in the set of data flow diagrams included in Appendix F. In order to provide more insight into the design of the software required to achieve this functionality, the remainder of this chapter is devoted to a discussion of the software organization, the required components and their source, and the basic design and required features of the user-customized operating systems which execute on the CS and AP1 subsystems. The system level modules, which were introduced briefly during the discussion of the high level system organization, serve as the basis for the subsequent discussion. The system level decomposition of the software is included in the top level hierarchy chart in Figure 28. Appendix G contains a further decomposition of the system level modules. This set of charts provides more required insight into the complex structure of the 432/670 Avionic Subsystem software. The definitions of the modules shown in these charts are included in the Data Dictionary in Appendix H, along with the descriptions of the processes and data flows referenced in the data flow diagrams.

7.3.1 432/670 Central System Software. The 432/670 CS software is composed of two major modules, the iMAX Operating System (1.1) and User Processes (1.2). The iMAX operating system is a user-configured operating system which must be customized prior to use in a particular application. As discussed in Chapter 3, iMAX is supplied as a set of Ada package specifications and bodies. In customizing iMAX, the

designer must first determine whether the minimal or full iMAX configuration is to be used and decide if any special features, such as dynamic process management, are to be included. For this application, a full iMAX implementation is required since neither communication with a non-Debugger peripheral subsystem nor dynamic process management are supported by minimal iMAX. All full iMAX packages should be included to allow flexibility in implementation. The TEXT_IO package, which is provided with iMAX (although not part of iMAX), is needed to support the Debugger and optional terminal input/output functions. With the exception of dynamic process management, which is required, the optional packages are all small. The advantages gained from the increased flexibility they provide far outweigh the minor impact on the code size. After choosing the iMAX configuration, several primary iMAX packages must be customized. The Processors and User_Processes package bodies must be written by the user in the format specified in the package shells provided with iMAX. These packages are used during system initialization to start the GDPs and the user static processes, respectively. The IP_Processors body must also be modified to add the AP1 peripheral subsystem. Code must be added to this package, which is also used during initialization, to start the AP1 IP and put it into the logical addressing mode (Ref 17:CON-1 - CON-13). All code for the user static processes which are started by User_Processes must be provided by the user. For this design, all of this code is included in the User Processes module.

7.3.2 AP1 Subsystem Software. The AP1 Subsystem software structure parallels that of the CS software, with an operating system

module and an application code mode. The AP1 Executive (2.1) provides the standard operating system functions, including system initialization. The I/O Controller (2.2), composed primarily of user-supplied code, controls the AP's communication with the CS and the 1553BBI, and provides the AP message processing and management functions. These two modules are discussed in more detail below.

7.3.2.1 AP1 Executive. The AP1 Executive is based on the user-configurable iRMX 88 Real-Time Multitasking Executive. iRMX 88 is supplied as a set of modules which include a nucleus, free space memory manager, terminal handler, disk file and input/output system and a bootstrap loader for booting from floppy disk. The minimum size of a configured system, not including the user code, is about 1.4K bytes. The systems which implement disk input/output functions exceed 64K bytes, and require the use of the megabyte version of iRMX 88. Designing an iRMX 88 system requires careful consideration of a number of factors. PROM and RAM availability are major drivers in the design process, since trade-offs involving added utility and macro-level functional interfaces versus size (Ref 34:1-1, D-1) must be made. The limited amount of PROM on the AP board (16K bytes) dictated that a mid-level operating system be designed. The modules selected for use in the AP1 Executive include the nucleus with interrupt and timed wait support, the free space manager, and the terminal handler. The input/output system was excluded both due to its size (64K bytes) and its low degree of utility. Since the 1553BBI is considered a custom device, the only advantage gained by using the input/output system is access to a standard interface based on three I/O calls. The user must write the

code to respond to these calls, thereby further increasing code size. The use of memory-mapped I/O is a viable alternative that requires only a small amount of user code to implement. In addition to selecting modules, the type of addressing must be determined. Since the system size is less than 64K bytes, either megabyte or non-megabyte can be used. The megabyte version offers the advantage of access to several macro-level system calls in the free space manager which simplify the memory management process. This added utility comes at the cost of address pointer size, 4 bytes versus 2 bytes for the non-megabyte version (Ref 34:1-1 - 3-15). The determination of the preferred addressing mode is actually an implementation-related decision which should be based upon expected message throughput. Systems with higher message storage demands should use the non-megabyte addressing scheme.

The system initialization functions are grouped with the executive functions. These functions are divided into two categories. The standard API initialization tasks include initializing the 86/12A board, starting the AP support software which provides the IP window management, building the Priority Cross-Reference Table used by the queue manager, and initializing the outgoing message queue's message blocks and linked lists. The second set of initialization tasks is responsible for initializing the 432/670 CS in the absence of the Debugger Subsystem. Implementation of API-controlled CS initialization is very desirable since it provides restart capability during mission execution. The design of this optional module is not included, however, the functions to be performed are basically the same as those for Debugger-controlled initialization.

7.3.2.2 Input/Output Controller. The design of the I/O Controller is significant since this software performs a multitude of functions. The I/O Controller consists of three major software modules, as shown in Figure G.4, Appendix G. The AP Message Processor is user-supplied code which implements the AP Process Message functions described previously. The IP Transfer Server, also supplied by the user, controls the passage of data through the IP windows. The IP Transfer Server relies upon the iMAX AP Support Software to operate. This software, written in PL/M-86, is supplied with iMAX. The iMAX AP Support Software performs two major functions, IP management and AP initialization. This package also provides a standard operating system interface for AP subsystem designers, through the AP Executive Calls package. AP software, which is written in PL/M-86 and which takes advantage of the interface provided by this shell, can be ported between different AP Executive implementations. The IP Controller software, consisting of the AP Executive Calls and the IP Function Facility which provides the window control functions, does not require user-modification. The AP Initialization module is configured by the user, who selects the appropriate AP initialization routine (APINIT) and AP initial task (INITLZ), based upon required terminal support. For this design, single terminal support versions were selected in order to support the optional operator terminal (Ref 17:IOI-1 - IOI-27). Detailed information concerning the various iMAX AP Support Software packages is included in Reference 17, Chapter IOI, Input/Output Implementation.

The structure of the software required to implement the 432/670 Avionic Subsystem avionic system data processing function is quite

complex, since the application software must be closely integrated with the numerous operating system and input/output support packages required for the operation of the 432/670 computer system. As mentioned previously, a more detailed look at the system software design is presented through the data flow diagrams and hierarchy charts presented in Appendices F and G, respectively. The processes, data flows, and software modules referenced in these diagrams are defined in the Data Dictionary in Appendix H. These two complementary sets of diagrams describe the design of both the application software and the two operating systems needed to perform the 432/670 Avionic Subsystem processes, as well as clarifying the manner in which the various modules must be organized to achieve the required functionality.

7.4 Software Design Analysis Summary

The design of the message processing software presented in this chapter is based upon sound structured design techniques, and meets all applicable global and system level design requirements. The flexibility and cost-effectiveness aspects of the design are reflected in its direct applicability to any type of avionic system implementation. No modification to the code is required to customize the software to support different types of data processing required by the other avionic subsystems, since this information is in a separate data dictionary. In addition, no analysis of the expected data processing rates is required, since the dynamic creation/deletion of processes allows for any mix of processing requirements. The design is also improved, since, with the dynamic creation/deletion of processes, the process mix exactly matches the requirements.

table-driven nature of the application processing portion of the software simplifies changing and/or adding application programs, enhancing system maintainability as well. The use of accepted structured design techniques helps to ensure that the software design is relatively simple and straightforward and that the resulting software is reliable.

The software system level requirements are all met by the design, as well. These requirements stipulate that the the iMAX and iRMX 88 operating systems and the Ada and PL/M-86 languages, be used in the design. The system level requirement that the AFIT system configuration be used as the basis of the design effort also levied constraints on the software, in that available PROM on the AP1 board and RAM on both processor subsystems is restricted. The exact size of the software could not be determined, since its implementation was precluded by the absence of the Debugger hard disk. The following estimates of the code size, which assume proper implementation, show that this requirement can be met.

The iRMX 88 configuration uses approximately 4.9K and 5.1K bytes of PROM for the non-megabyte and megabyte addressing modes, respectively (Ref 34:D-1), with RAM usage for both being less than 400 bytes. The remaining 11K bytes of PROM and 63.6K bytes of RAM available for application processing use should be quite adequate to support any reasonably efficient implementation of the AP message processing software. The 432/670 CS RAM is restricted to 512K bytes, of which 233K bytes is required for iMAX after system initialization is complete (Ref 17:SI2-1). A major goal of the design was to minimize the amount

of the RAM used by the processing software, in order to provide more RAM for message storage and enhance system throughput. The dynamic application processing scheme discussed above supports this goal. Assuming a reasonably efficient implementation of the static processes design, in excess of 128K bytes, or a quarter of the system RAM, should be available for message storage. This analysis of memory use shows that implementation of the system software, despite the constraints imposed by the hardware configuration, is feasible.

The design analysis presented above shows that the requirements imposed upon the software design, as stated in Chapter 4, have all been met. As stated in the hardware design analysis summary, Section 6.4, the qualitative aspects of the design are evaluated with respect only to the portions of the design which are within the control of the designer. The simplicity, maintainability and cost-effectiveness of the 432/670 software are not considered since the use of this system was stipulated in the project proposal. A discussion concerning the disadvantages of interfacing the 432/670 computer to the 1553B bus for use in an avionic subsystem is included in the next chapter. Issues such as the complexity of the peripheral subsystem interface software are considered in that discussion.

CHAPTER 8

CONCLUSIONS AND RECOMMENDATIONS

8.1 Conclusions

The purpose of this project was to investigate the feasibility of interfacing the Intel 432/670 computer system to the MIL-STD-1553B data bus. The major objectives included developing a conceptual design for a military avionic subsystem based on the 432/670 computer system, defining the requirements for the subsystem, and investigating the feasibility of its implementation. The 432/670 Avionic Subsystem, whose design was the result of this investigation, meets all requirements stated for such a system. The feasibility of implementing the interface between the 432/670 computer system and the 1553B data bus was proven by designing the subsystem to a level at which the design could be evaluated based on the specific requirements. The design analyses for both the hardware and software, summarized at the end of Chapter 6 and Chapter 7, respectively, prove the feasibility of implementing the interface. During the course of the investigation, however, the author developed serious reservations concerning the use of the 432/670 in an avionic system. The analysis summaries hinted at the existence of such reservations, which are discussed below.

This project was proposed based on the premise that the performance of an avionic system could be enhanced by providing real-time access to mainframe computational power. This premise must still be considered valid, since, despite the sophistication of current avionic systems,

many avionic subsystems are little more than sensors which possess extremely limited computational power. A subsystem integrating these sensors and pre-processing their output data in real-time for use in the increasingly overburdened subsystems, such as the flight computers and electronic countermeasures systems, would have considerable utility. Unfortunately, the author feels that, due to limitations inherent in the architecture of the 432/670 computer system, a real-time processing capability would not be realized.

The limitations of the 432/670 computer stem from its architecture, in which a separate peripheral subsystem is required to perform all input/output operations. This subsystem adds an additional layer through which all data must pass prior to reaching the primary processor. The time lag inherent in passing both the incoming and outgoing data through this additional peripheral processor, the author believes, would be so great as to preclude real-time processing of the data to meet the needs of the sophisticated, high-speed, destination subsystems. The 432/670 Avionic Subsystem could be used in applications not requiring a real-time processing capability, such as data collection and pre-processing for post-flight data reduction. However, many processors currently available can provide the power required for such processing without the associated complexity. The non-standard architecture and the complexity of the software which arises from the introduction of the peripheral subsystem, would also cause significant problems in maintaining the system in the military environment. Training software and hardware maintainers to support a system with very limited utility in military applications would not be cost-effective and

... Support
... would be
... system
... generally
... both
... to the
... despite the
... since the
... a-kind
... systems based
... is readily
... specific design
... possible. The use
... hardware and software
... the implementation of the time-
... in hardware, required to
... the Multibus
... a much more general
... The 1553BBI hardware can be
... with only software
... The Multibus I/O Module is also
... allowing easy substitution of support hardware for any
... The software,
... can
... The basic multitasking structure of

the software is directly supportable since the Department of Defense requires that Ada be used for all software for embedded military computer systems. The message processing software on the AP can then be readily hosted on any processor which supports 16-bit memory-mapped input/output functions. The priority/FIFO incoming queue manager, as well as the application programs, would need to be rehosted on the main processor when the 432/670 code and the IP transfer functions are removed, but few other changes would be required. A recommendation for rehosting the 1553BBI hardware and support software is included in the next section.

8.2 Recommendations

The recommendations fall into two general categories. The first set of recommendations deals with suggested pre-implementation modifications to the 43/670 Avionic Subsystem design. These recommendations are as follows:

1. Use militarized or ruggedized versions of the 432/670 computer with airframe-compatible power interfaces (400 Hz, 220/110 volt, 3 phase AC power and ± 15 volt DC power). The AFIT 432/670 system is a standard commercial version which uses single phase, 110 volt AC power and ± 12 volt DC power, both of which are not readily available on military aircraft.
2. Increase the 432/670 RAM to a minimum of 2M bytes to optimize application processing, maximize message storage, and support the upgraded version of iMAX which requires a minimum of 1M byte of memory.
3. Use the AP1 Subsystem for initializing the 432/670 CS. This modification would allow the system to be reset without requiring access

to the Debugger Subsystem for reinitializing the 432/670 CS and reloading the software. Changes to the hardware configuration would include adding additional PROM to the AP1 Subsystem for storing the 432/670's initial execution environment, and modifying the AP1 initialization software to perform the CS initialization. Without this modification, the 432/670 could not realistically be used in an airborne application.

The above recommendations are not to be construed as a recommendation that the 432/670 Avionic Subsystem be implemented. As discussed in the conclusions above, the utility of the system is questionable. As an alternative, the author recommends that a follow-on study be performed to identify a more appropriate processor, perhaps based on the MIL-STD-1750A architecture, to replace the 432/670. The 1553BBI hardware should then be built and the software modified to allow testing and verification of the 1553BBI hardware design.

APPENDIX A

THE AFIT 432/670 CROSS-DEVELOPMENT SYSTEM CONFIGURATION

The 432/670 Cross-Development System Software Components

1. ACS 432 - The Ada Compiler System which translates Ada source programs into object code, generates source and error listings and builds combined files from individually compiled modules.
2. LINK-432 - The object file linker which links compiled modules into single executable programs.
3. iMAX 432 - The 432 Multifunction Application Executive which is a collection of different software components that implement the basic operating system functions for both the Central System and the Peripheral Subsystems. iMAX provides Ada packages for the 432 Central System and PL/M-86 modules for the subsystems.
4. ACL 432 - The Asynchronous Communication Link software which is used to transfer files between the VAX host and the Series III Debug Workstation.
5. DEBUG-432 - The program debugger which controls the execution of the 432 programs by loading the programs and providing the user with monitor and alteration capabilities. DEBUG-432 runs on the Series III under ISIS II.
6. UPDATE-432 - This package is used to combine revised object files created on the VAX host with linked files which have previously been downloaded to Series III storage devices.
7. DSP 432 - The 432 Diagnostic Software package is used in isolating malfunctioning boards in the 432/670. The diagnostics execute on the Series III iSBC 86/12A board, testing the GDP boards, the Memory Subsystem and the hardware link between the 432 and the Series III. Another set of 432 diagnostics is stored in PROM on each AP and is used to test the interface between the subsystem's IP and IPL.
8. iRMX 88 - The user-configurable, real-time multitasking operating system for the 8086 AP. iRMX 88 provides the basic operating system modules, written in PL/M-86, which must be interfaced by the user who also must supply device drivers and interrupt handling routines.

The 432/670 Cross-Development System
Hardware and Software Compatibility Guide

(Current as of 10 June 83)

Component	Original Configuration	Upgraded Configuration
432 GDP	R2.1	R2.1
432 IP	R2.0	R2.1 (AP #1 Subsystem) R2.0 (Debug Subsystem)
ACS 432	V1.00 (VMS)	V1.01 (VMS/UNIX)
LINK-432	V1.00 (VMS)	V1.01 (VMS/UNIX)
iMAX 432		
- 432 Code	V1.00 (VMS)	V2.01 (VMS/UNIX)
- AP Code	None	V1.00
ACL 432		
- Host pkg	V1.02 (VMS)	V1.02 (VMS/UNIX)
- Debug pkg	V1.01	V1.01
DEBUG-432	V1.00	V1.01
UPDATE-432	V1.00	V1.00
DSP 432	V1.00	V2.00
iRMX 88	None	V2.00

Note: The upgraded configuration is required for the implementation of any Peripheral Subsystem other than the Debugger.

The 432/670 Mainframe Hardware Configuration

(Current as of 10 June 83)

Card Cage Size: 18 slots total

Slots 1-12 for Central System/System Bus

Slots 13-18 for AP #1 Subsystem/Multibus

Slot Assignments:

1. Reserved for SA board
2. Reserved for Storage Array (SA) board
3. Reserved for SA board
4. Reserved for SA board
5. SA board/256K bytes RAM
6. SA board/256K bytes RAM
7. Memory Controller (MC) board
8. GDP #1 (Dedicated P#1 slot - see iMAX configuration)
9. GDP #2 board (P#2 slot)
10. GDP #3 board (P#3 slot)
11. Debugger IPL board (P#4 slot)
12. AP #1 IPL board (P#5 slot)
13. AP #1 IP board (R2.1)
14. AP #1 - 86/12A board, 64K bytes RAM, 16K bytes EPROM
15. Reserved for MIL-STD-1553B Interface board
16. Empty
17. Empty
18. Empty

The Debugger Subsystem Hardware Configuration

(Current as of 10 June 83)

Debugger: Intel Series III Microcomputer Development System

System Components:

8085 CPU, 64K bytes RAM

86/12A Resident Processor, 64K bytes RAM, 16K bytes EPROM

432 IP board (R2.0 -- due to be replaced with an R2.1)

3 - 64K byte RAM boards

Integral Single Density Eight Inch Disk Drive

2 - Double Density Eight Inch Disk Drives

Operating System: ISIS II V4.3

APPENDIX B

ATTACHED PROCESSOR HARDWARE CONFIGURATION REQUIREMENTS

A standard Intel iSCB 86/12A Single Board Computer is used as the Attached Processor (AP) in the AFIT 432/670 Computer System. Some jumpering is required to customize the board for each particular application. Although defining this jumpering is normally a detailed design or implementation concern, several decisions affecting the detailed board configuration had to be made during the course of the system level design effort. The hardware design for the AP1 Subsystem assumes the configuration discussed below for the 86/12A AP board. Detailed information can be found in Reference 26.

1. ROM/EPROM: Four 2732 EPROMS (200 nanosecond or faster) are used for the ROM/EPROM memory. The use of EPROMS is necessary to support software development, while the maximum 16K byte on-board ROM storage allows the most flexibility in designing the AP1 Subsystem software. Since this software is composed of the iRMX-88 operating system, the iMAX AP Support Software (for communication with the 432/670) and the 1553BBI drivers, available ROM storage space is a critical design factor. The higher speed chips and the No Wait Option jumpering are required to avoid generating unnecessary wait states.

2. Clock Signals: The Bus Clock signal BCLK-L and Constant Clock signal CCLK-L (9.22 MHz, 35- 65% duty cycle on both) are enabled onto the Multibus to provide the Multibus signals of the same name.

3. Bus Control: The Bus Arbitration logic is designed to allow the 86/12A to retain control of the Multibus at each transfer cycle without contention, until a higher priority bus master requests control. Since no other master resides on the bus, no time is lost through the automatic surrender of the bus at each transfer cycle.

4. MIN/MAX-L: MIN/MAX-L is tied low to enable Multibus communication.

5. 8253 Configuration: The 8253 Programmable Interval Timer is configured to act as a programmable baud rate generator for the serial input/output port (USART).

6. Failsafe Timer: The 86/12A Failsafe Timer is enabled to prevent the CPU from hanging in a wait state. Time-out then occurs whenever an acknowledge signal (XACK-L) is not received within 6.2 milliseconds.

7. Interrupts: The non-bus vectored interrupt scheme is used, providing eight prioritized interrupt levels, with INTO being the highest level. The following 8259A Programmable Interrupt Controller assignments are either required by the iMAX AP Support Software, single terminal implementation, (Ref 17:HDW-3) or were included in the AP1 Subsystem design:

INT0 - Subsystem Error (AP1, SS ERR)

INT1 - Optional Monitor (iMAX)

INT2 - System Timer (USART Support)

INT3 - In Message Service Request (AP1, IN MSG S/R)

INT4 - 432/670 IP Board Use (iMAX)

INT5 - Out Message Service Request (AP1, OUT MSG S/R)

INT6 - USART Receive-Ready (iMAX)

INT7 - USART Transmit-Ready (iMAX)

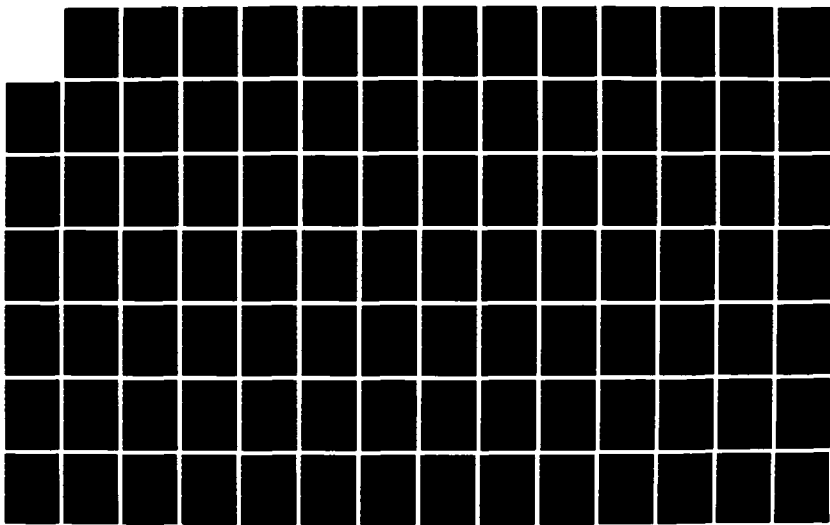
AD-A189 841

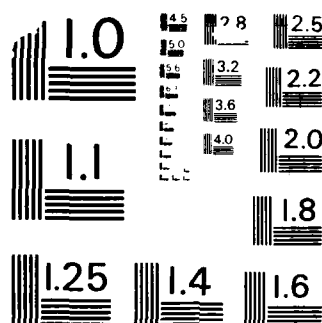
AN INVESTIGATION OF THE INTERFACING OF THE INTEL IAPX
432/670 COMPUTER SY. (U) AIR FORCE INST OF TECH
WRIGHT-PATTERSON AFB OH SCHOOL OF ENGI.. L H BLACK
SEP 87 AFIT/GE/ENG/875-1 F/G 12/6

3/4

UNCLASSIFIED

ML





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

APPENDIX C

API HARDWARE DESIGN BLOCK DIAGRAMS

Appendix C contains the block diagrams and applicable design notes for the 1553B Bus Interface (1553BBI) hardware design discussed in Chapter 6, Section 6.2.3.2. The module numbering scheme is basically the same as that described in Chapter 5 (Section 5.3.1) for Data Flow Diagram leveling. The top level module is assigned the module number C.0. Submodules within this module are assigned numbers C.1, C.2, and C.3. These numbers, which are located in the lower right corner of each submodule, are used as the submodule figure numbers. The block diagrams are ordered so that the submodules are placed immediately following their parent modules whenever possible. A set of general design notes precedes the block diagrams, and when necessary, design notes for individual modules are included immediately following the pertinent block diagrams. These notes describe any assumptions made concerning specific characteristics of components included in the design, i.e. the FIFO buffers, as well as providing any necessary clarification of design decisions and implementation requirements. Unless otherwise noted in the design notes, all signals used in the hardware block diagrams are defined in Appendix D, the Hardware Signal Dictionary. The timing diagrams for this design, which show that the design meets the timing requirements imposed by the RTU chip set, and thus the 1553B bus protocol requirements, are included in Appendix E. References 29, 30, 31, 32 and 33 served as references for the design effort.

General Hardware Design Notes

The purpose of the 432/670 Avionic Subsystem design effort is to demonstrate the viability of a 432/670 to 1553B bus interface. The hardware design revolves around the 1553B Bus Interface (1553BBI) since this module provides the physical connectivity and is responsible for all of the time-critical message handling functions. The 1553BBI Module design presented in this appendix is quite detailed in many respects, since these details are needed to prove the viability of the interface. This detail is particularly evident in the RTU Support Module design, since this module, unlike the the Multibus I/O Module, must perform a number of specialized functions. Where possible, to allow flexibility in the subsequent detailed design and implementation efforts, a generic design accompanied by notes concerning implementation requirements and specific component characteristics is presented. This approach allows the implementer to select specific components based upon cost, availability and the current state of technology. Design notes specific to a given module are included with its design; those which are more generally applicable are discussed below.

1. The format of the block diagrams selected clearly delineates between the internal module design and the external interfaces. Two borders are used in the diagram of each module. The inner border defines the module itself, while the interfaces are shown in the area between the two borders. Within this area, different inter-module boundaries are labeled, and the signals traversing the specific boundaries are

shown entering and leaving the module only in these assigned areas. To avoid excessive detail, related interface signals are grouped and designated as interface buses, and only signals critical to the design effort are included in the block diagrams. Signals which are not shown include available, but unused, chip outputs and internal RTU chip set signals. In addition, names are not assigned to signals with obvious functions which are generated and used internal to a given module. For example, the output signal of a NAND gate with defined inputs may not be named if it is used solely to enable a component within that module.

2. Due to speed and bus loading considerations, all discrete logic should be implemented using "ALS" or "AS" components. See Reference 33 for information concerning general characteristics of these components.

3. Tri-state devices must be used to buffer all signal lines tied to the RTU's T0 - T15 Data Highway and the Multibus I/O Module's DAT0 - DATF Data Bus. This buffering is necessary to reduce bus loading and to prevent conflicting bus access. Reference 29 states the requirement that all signals gated onto T0 - T15 be tri-stated off except during the specific periods in which transfer operations between the RTU and subsystem are being performed, since the highway is reserved for internal RTU operations at all other times. Since this buffering is so critical, tri-state buffers are shown in all required locations in the design. The buffers are shown as separate components in the design since device descriptions have been made as generic as possible. Selection of devices with built-in tri-state buffers will enhance performance, and minimize complexity and space requirements.

4. Open-collector logic must be used to buffer all signals onto the Multibus. Buffering of signals from the Multibus onto the 1553BBI is also necessary in order to reduce bus loading.

5. For the purpose of standardization, all discrete buffer, FIFO buffer, latch, counter and comparator enable, reset and load/unload signals are shown as active low. All clock signals are positive edge-triggered. The generic components used in the design show the most general set of control signal inputs expected to be encountered. After selection of the components, the design must be refined to compensate for differences in control signal availability (separate or combined load/enable, e.g.) and type (active low, active high, edge-triggered, etc.).

6. The single D-type latches used in the design are assumed to have the following characteristics: an active low clear which resets the latch to a known state, a positive edge-triggered clock and both active high and active low outputs. The truth table for this latch is shown in both Figures C.1.1.3 and C.1.2.5.

7. The latches which serve as the Command Buffer (Figure C.1.2.1), Tag and Old Tag Buffers (Figure C.1.2.3.2) and AP ERR and SS ERR Registers (Figure C.1.2.4) are assumed to be transparent D-type latches with continuously available output signals and active low enables, which, when enabled, allow data to flow from the input to the output pins and, when disabled, provide as output the values latched at the time the enable signal went high.

8. The selection of appropriate FIFO buffers is a very critical factor in successfully implementing the design. As shown in the

Required Transmit and Receive Timing diagrams (Figures E.3 and E.6), the RTU imposes strict timing constraints upon the subsystem. To meet these timing requirements, the FIFO buffers must have the very low set-up and data hold times attributable only to RAM-type buffers. Shift register-type FIFOs cannot meet the speed requirements due to the depth of the registers. Selection of components with built-in tri-state outputs is very desirable since shorter settling times are normally achieved than with external buffering. The FIFO buffers shown in the In Message Storage and Out Message Storage Module designs (Figures C.1.2.2 and C.1.2.3.1.1) possess a set of very standard active low control signals: enable (EN), reset (RS), load (LD) and unload (UNL). They are also shown with non-buffered data input and output lines and with Empty, Full and Not Empty output status signals. This generic set of signals was chosen as a basis for supporting as wide a range of implementations as possible. FIFOs possessing exactly this set of characteristics are not likely to be available, nor would they necessarily be the most desirable for use. Speed considerations, adaptability of the output status signals to provide the required interrupt signals and on-chip support to allow cascading to the attain the required buffer sizes are the critical factors in device selection. The design can easily be modified to accommodate differences such as edge-triggered load/unloads, lack of separate enables, etc., given the sample control signal definitions and timing diagrams in Appendix E.

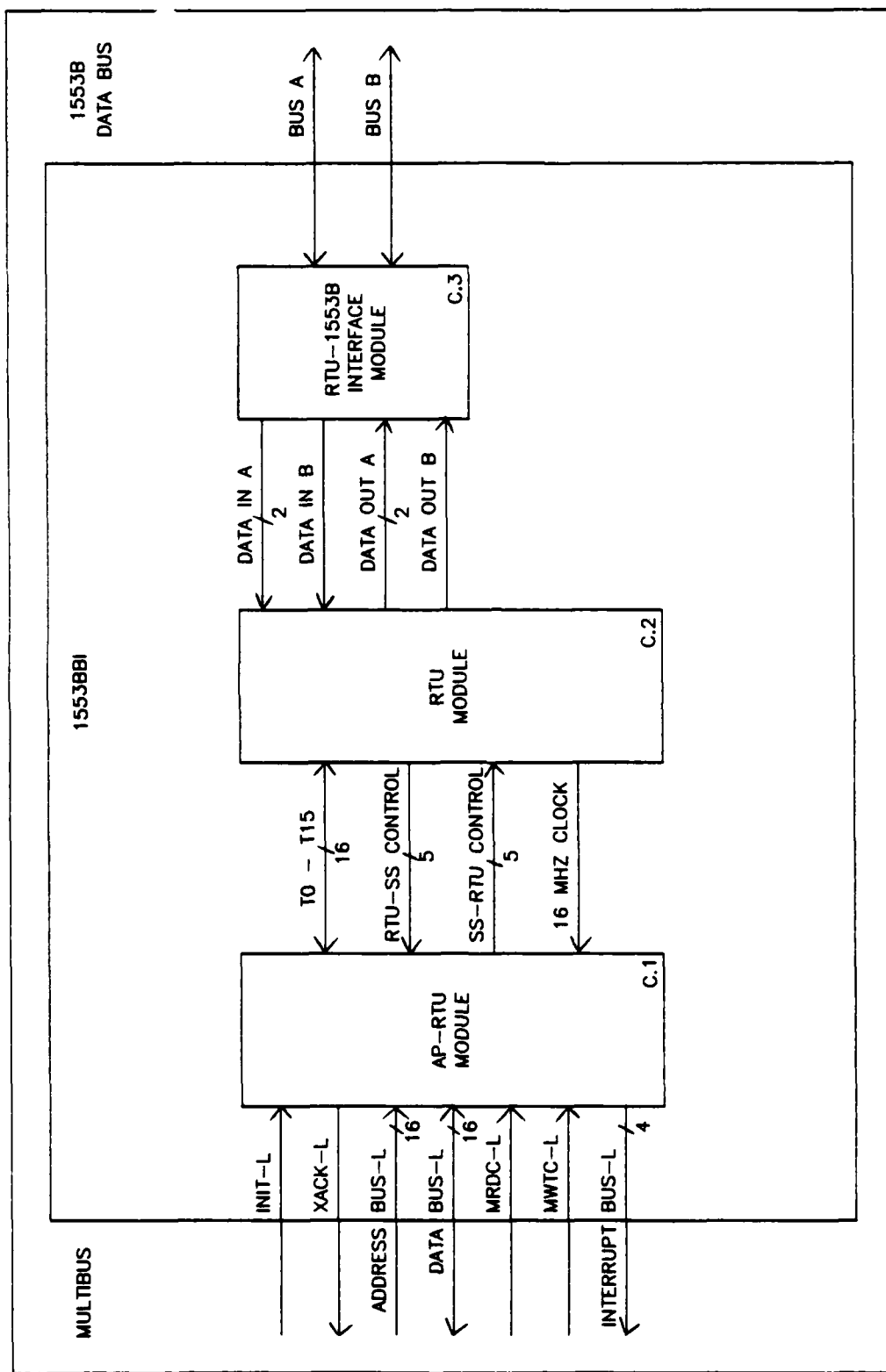


Figure C.0. 1553B Bus Interface (1553BBI) Module Block Diagram (Figure 21, Chapter 6)

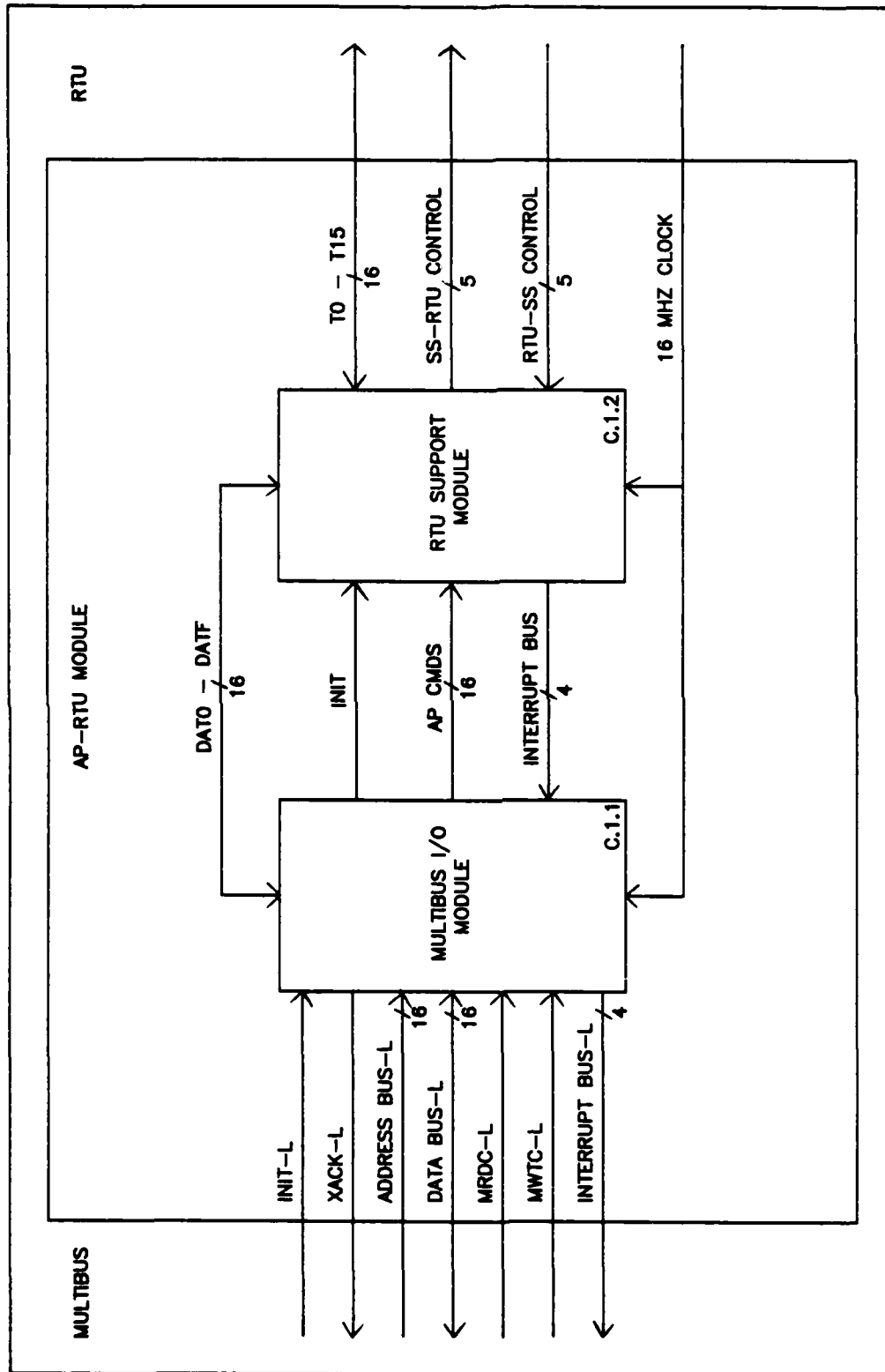


Figure C.1. AP-RTU Module Block Diagram

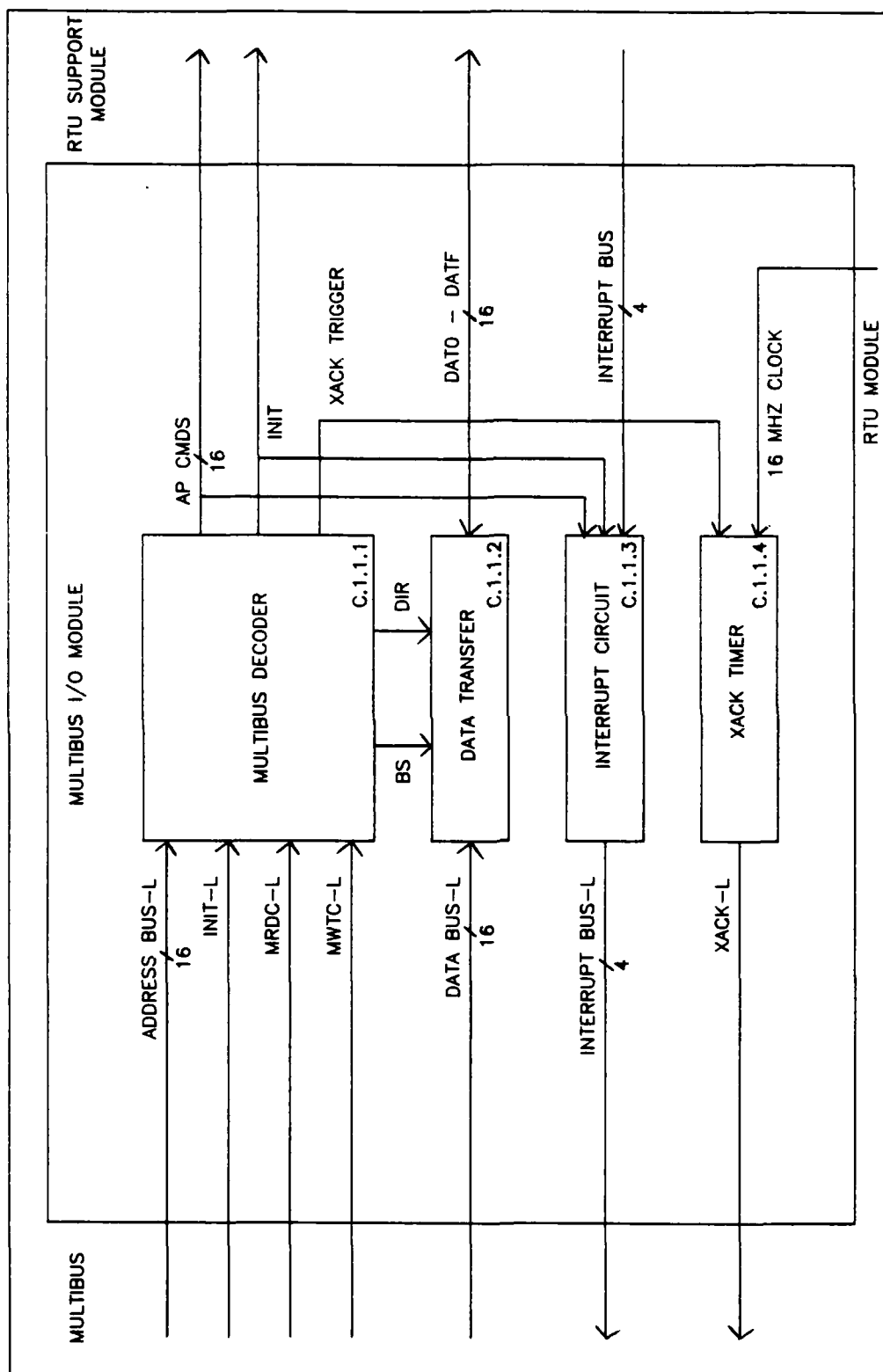


Figure C.1.1. Multibus I/O Module Block Diagram

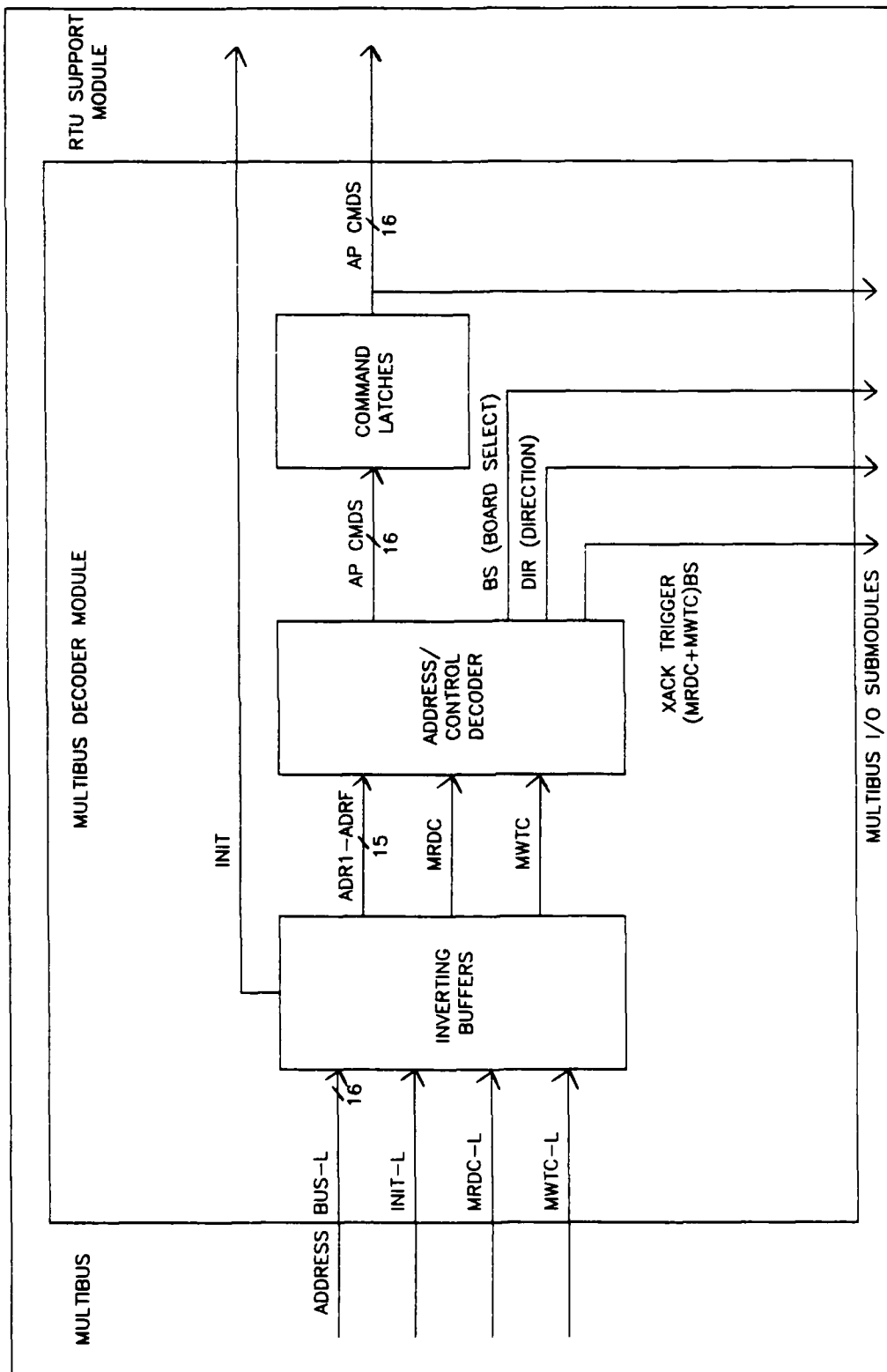


Figure C.1.1.1. Multibus Decoder Module Block Diagram

Multibus Decoder Module (C.1.1.1) Design Notes

1. Memory-mapped input/output is used to transfer 16-bit words between the AP and the 1553BBI. The Multibus signals BHEN-L (Bus High Enable) and ADRO-L are not used since the 16-bit transfers will always be on even address (byte) boundaries. Likewise, the swap buffer needed to support 8-bit transfers on either even or odd address boundaries is not required.

2. The addresses of all input/output commands are decoded to determine the function the 1553BBI is being tasked to perform. The decoding logic used to generate this set of functions, designated AP CMDS, must be kept simple since only 1 microsecond is available for command decoding before the data comes ready.

3. ROM, PROM, discrete logic or a combination thereof can be used to implement the decoder. The buffers can be incorporated directly into the decoding logic. The design requires that separate, single control lines be assigned to each function; when that function is decoded, that line and only that line should go true. The latches hold the values until the next input/output operation occurs. The actual codes associated with the individual AP CMDS are implementation dependent.

4. Tri-state bus transceivers are recommended. If non-tri-state devices are used, the default direction of the transceivers must be set for transfer from the 1553BBI onto the Multibus.

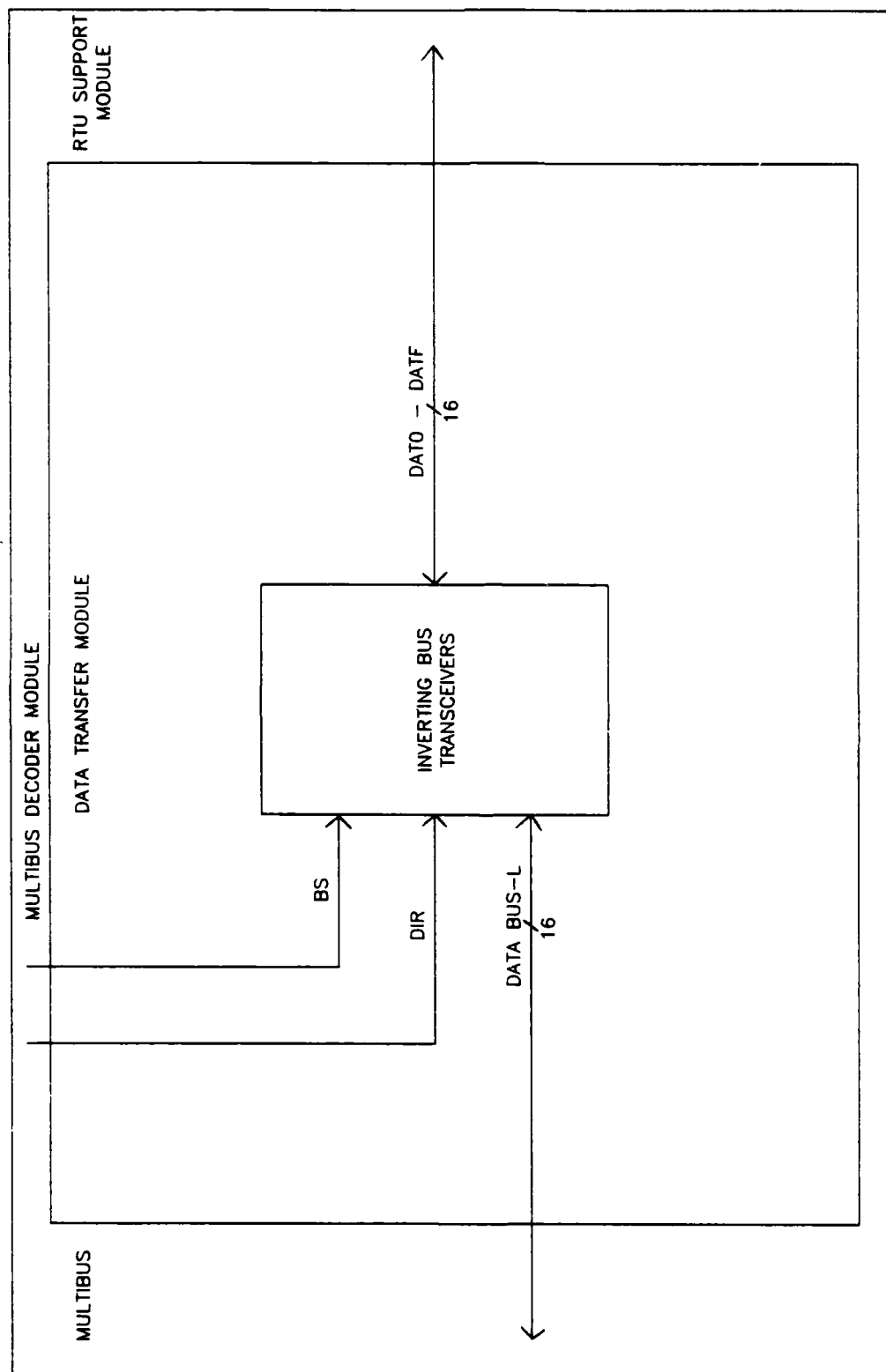


Figure C.1.1.2. Data Transfer Module Block Diagram

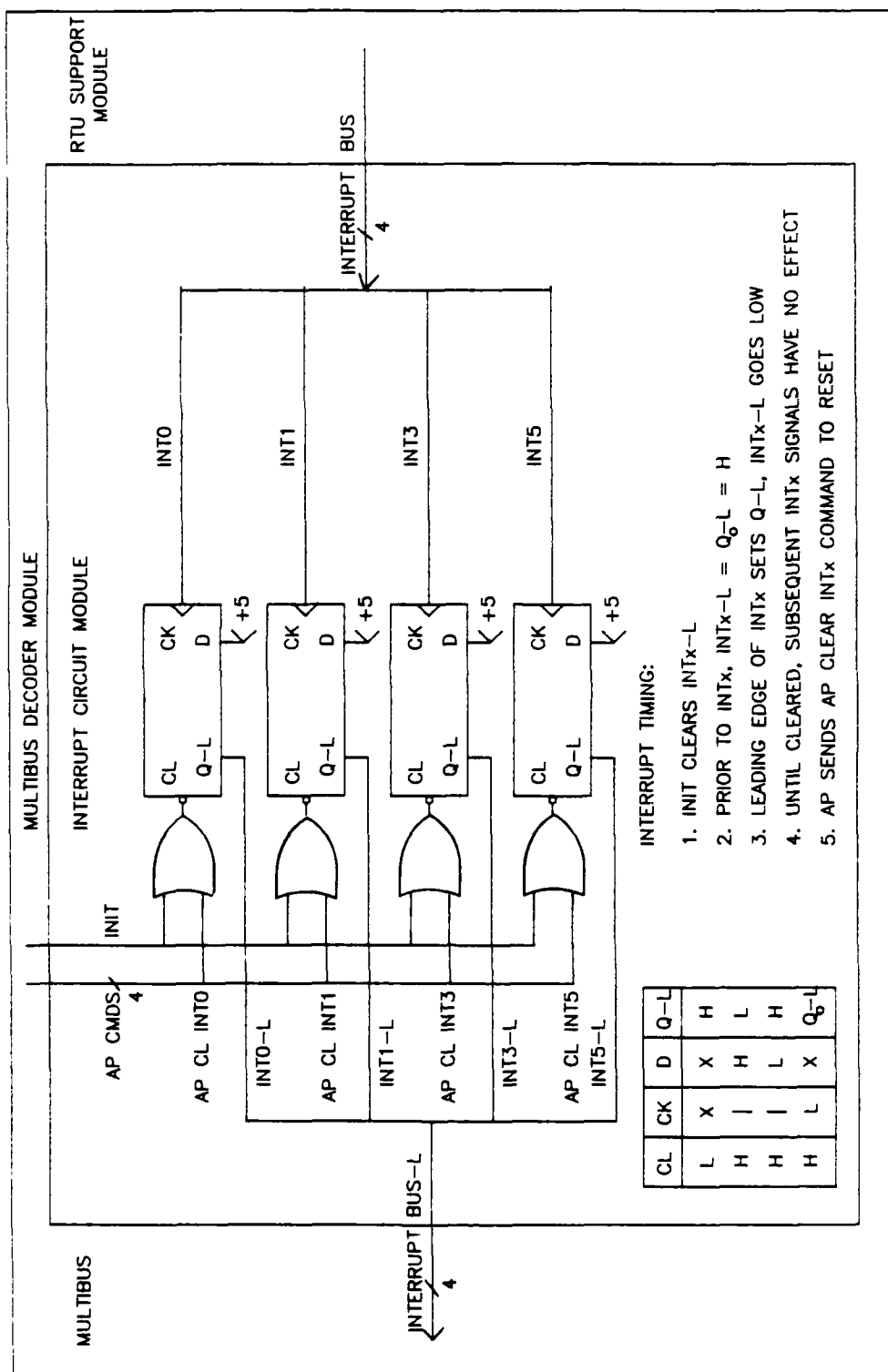


Figure C.1.1.3. Interrupt Circuit Module Block Diagram

Interrupt Circuit Module (C.1.1.3) Design Notes

1. Two interrupt handling schemes are supported by the 86/12A AP board. The bus vectored interrupt scheme requires that the slave boards gate interrupt vector addresses onto the bus upon receipt of the INTA-L signal which the CPU generates upon detecting an interrupt request. This scheme must be used whenever more than eight interrupt levels are defined. The non-bus vectored interrupt scheme, in which the AP's Programmable Interrupt Controller (PIC) generates the service routine addresses, can be used when eight or fewer interrupt levels are supported. This scheme is used in this design since it provides faster response time and is directly supported by the iMAX AP Support Package software.

With non-bus vectored interrupts, the interrupt request line for the given interrupt must be latched low on the slave (1553BBI) board until servicing is complete. The CPU does not automatically clear the interrupt; clearing the interrupt is the responsibility of the interrupt service routine. In this design, AP CMDS are used to clear the interrupts. The code for the appropriate AP CMD is written to the 1553BBI as the last service routine operation.

2. Following the initial setting of a given interrupt line, the interrupt handling circuitry automatically masks subsequent requests for that interrupt until the interrupt is cleared by the AP.

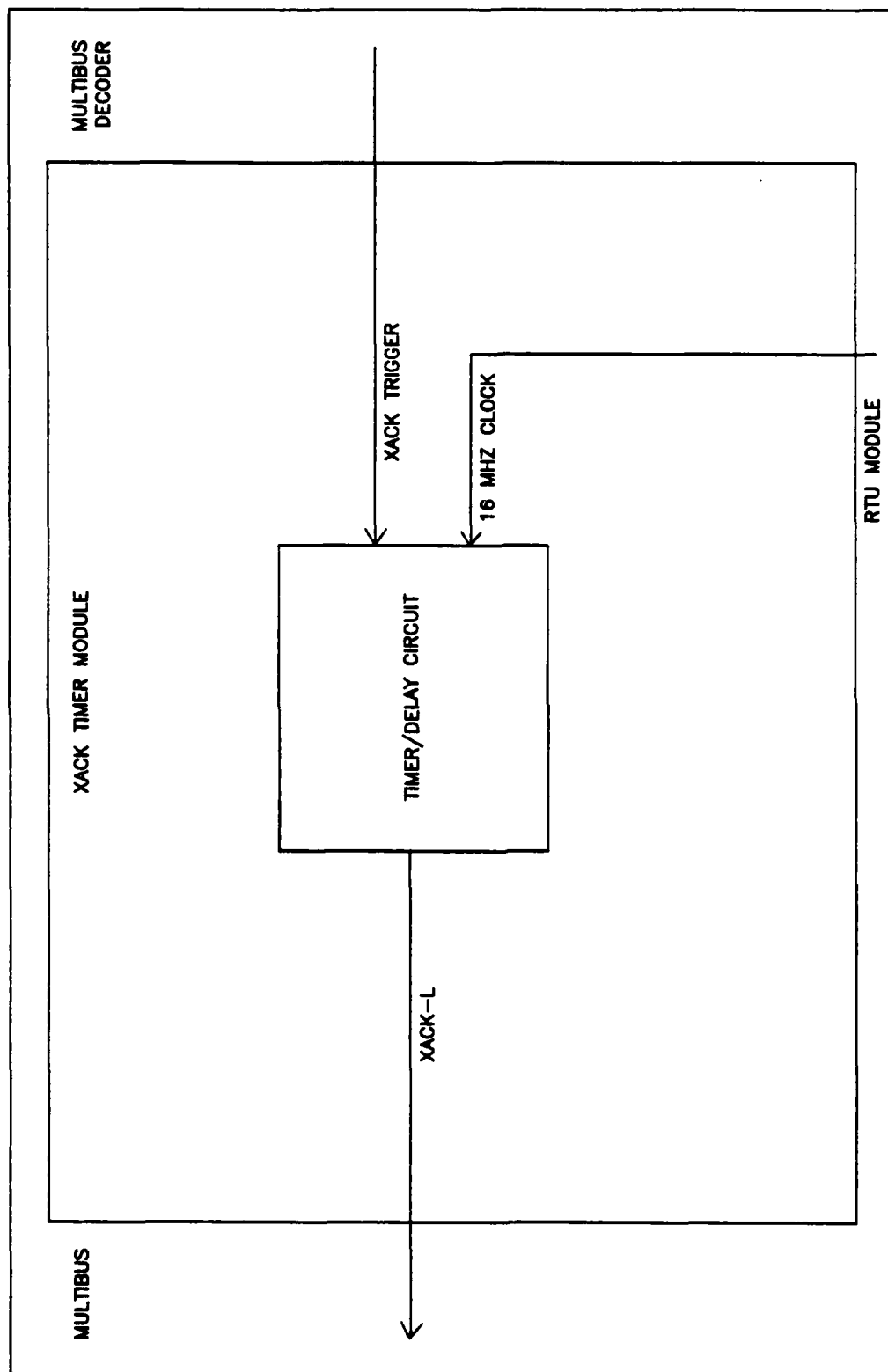


Figure C.1.1.4. XACK Timer Module Block Diagram

XACK Timer Module (C.1.1.4) Design Notes

1. The transfer acknowledge cycle is initiated for each memory read (MRDC-L) and memory write command (MWTC-L) directed to the 1553BBI board (BS true). This circuit implementation must allow for easy modification of the circuit response times since optimization will be required even if the basic timing guidelines are met.

2. Figure E.1 shows the basic required XACK Timing.

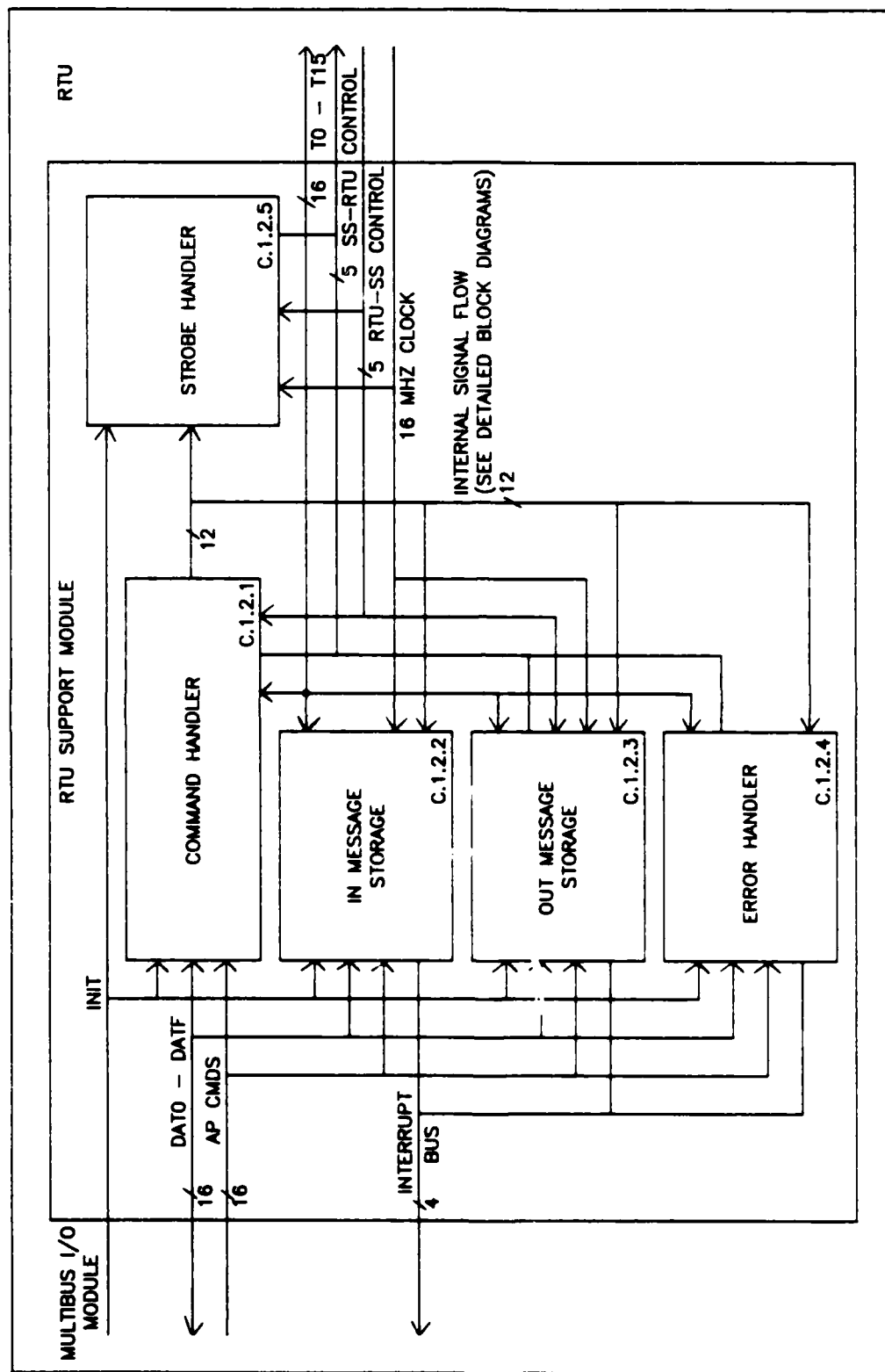


Figure C.1.2. RTU Support Module Block Diagram

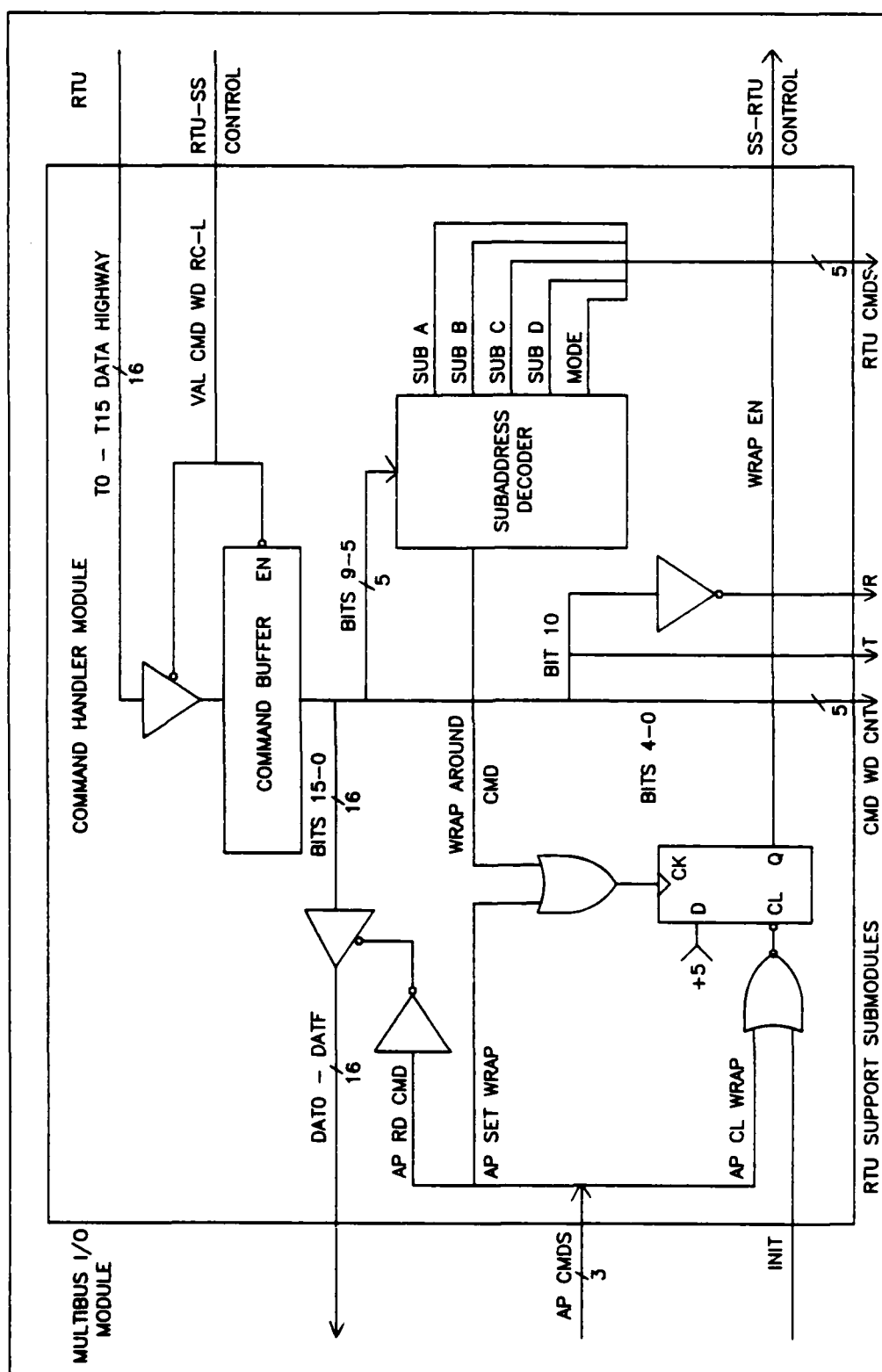


Figure C.1.2.1. Command Handler Module Block Diagram

Command Handler Module (C.1.2.1) Design Notes

1. The Subaddress Decoder circuit can be implemented in ROM, PROM, PAL or discrete logic. Rapid decoding is required to meet the subsystem timing requirements as dictated by the RTU.

2. A single, separate line must be used for each of the RTU Commands, of which only one can be true at any given time. Since these signals serve as enables for the other modules, the output lines of the decoder are latched following decoding, and must not be reset until the next valid command is received. The actual values assigned to the subaddresses are implementation dependent.

3. If desired, additional circuitry could be added to detect invalid subaddresses, with the error signal being routed to the Error Handler Module.

4. The Wrap Around handling circuitry is optional; the RTU's WRAP EN input can be hard-wired on (high) if desired. See Appendix D (WRAP EN) for further explanation.

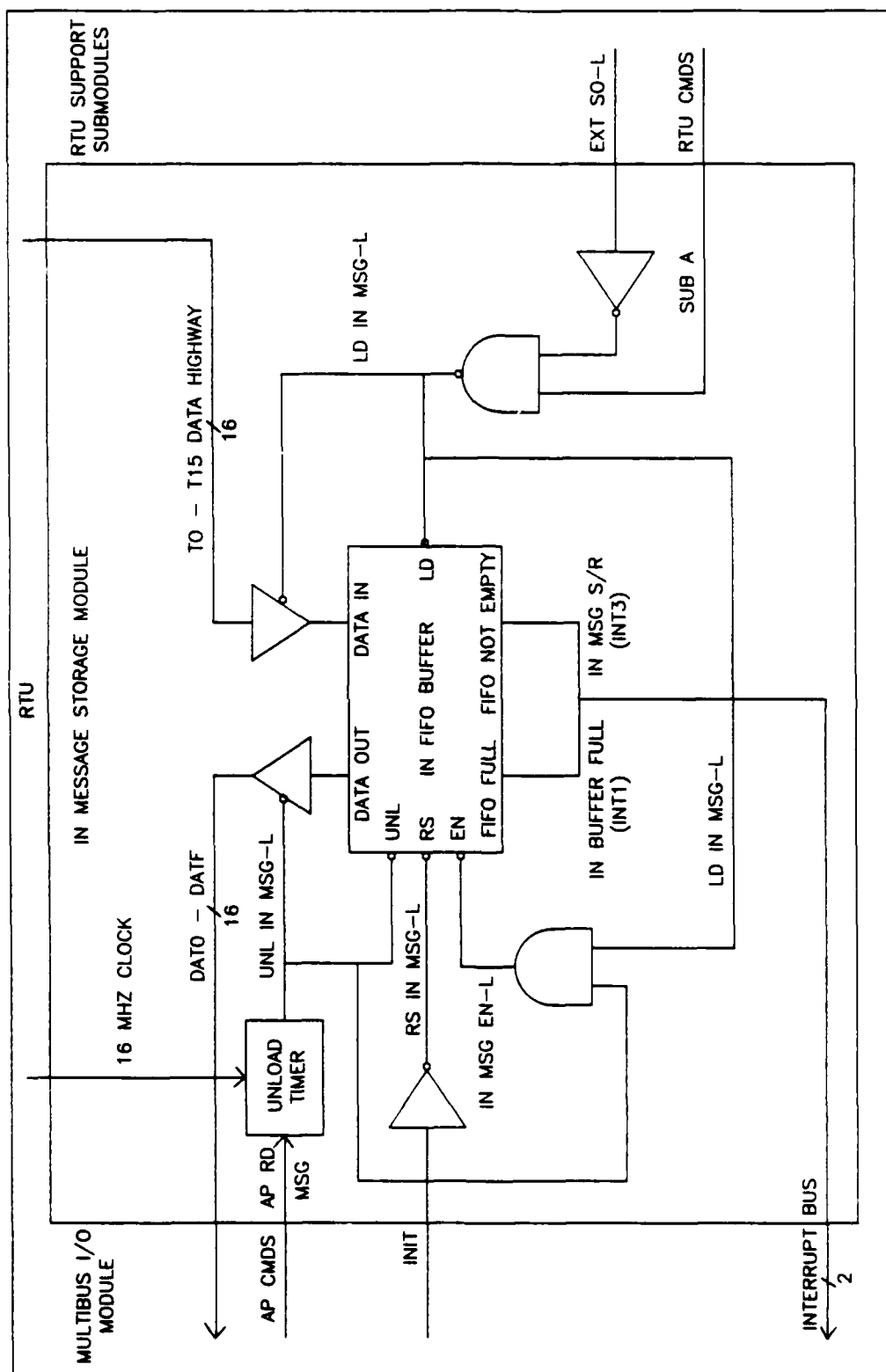


Figure C.1.2.2. In Message Storage Module Block Diagram

In Message Storage Module (C.1.2.2) Design Notes

1. Basic characteristics of the In FIFO Buffer device are discussed in the General Design Notes. The buffer must be at least 16 bits wide, but the required depth is application dependent. A minimum depth of 256K words is recommended to support the expected high-throughput processing requirements. This allows storage of 8K 32-word messages (minimum of 100% overhead), which should be adequate to compensate for the burst filling of the buffer and slower transfers between the AP and 432/670 CS, for variations in 432/670 processing time and for backlogs due to slow unloading of processed messages by the 1553B BC.

2. The Unload Timer Circuit is included to allow for conditioning of the AP RD MSG signal to match the requirements of In FIFO Buffer's UNL (unload) strobes. Signal delays and/or pulse length modifications (non-edge triggered control) may be required.

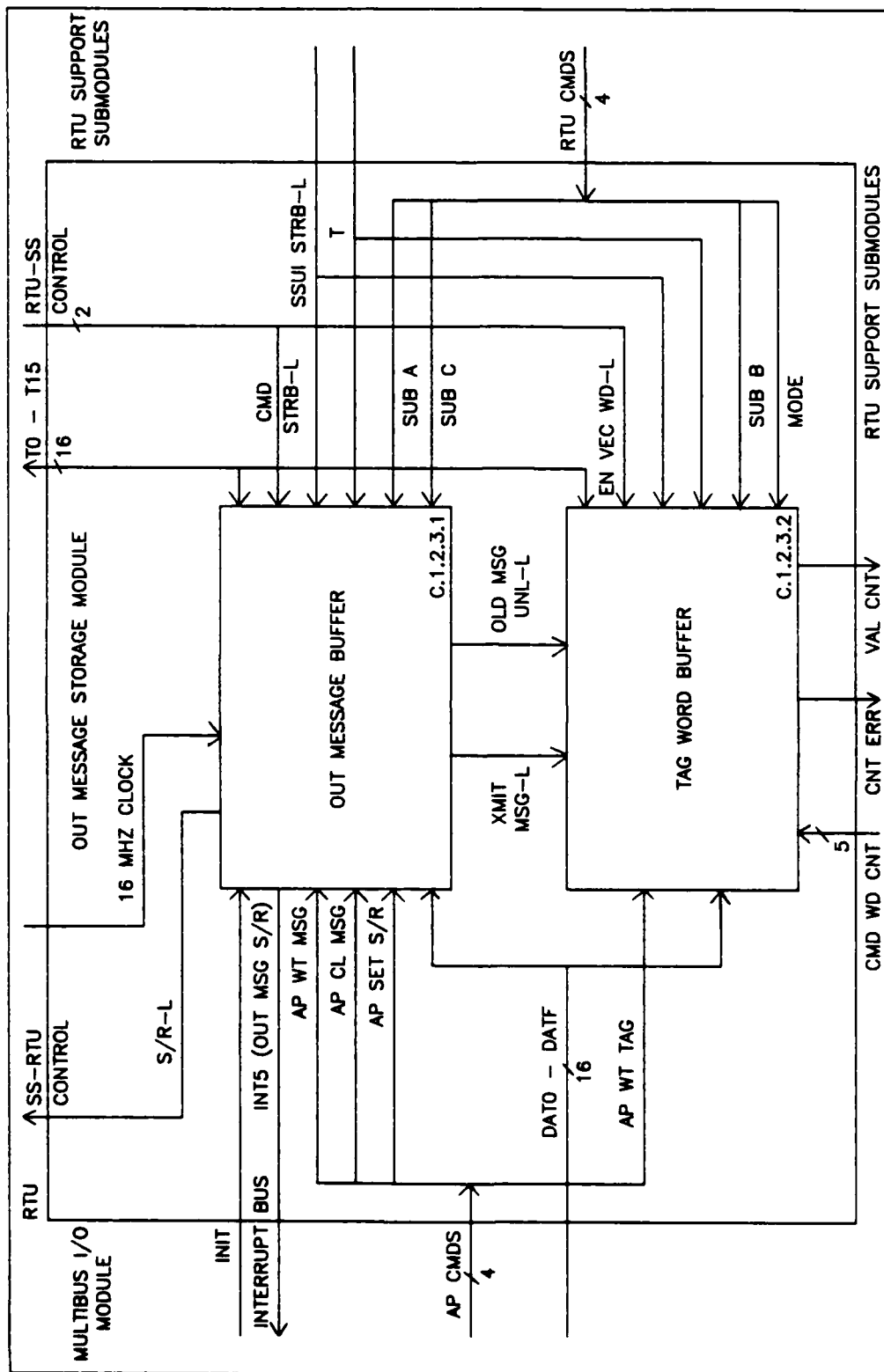


Figure C.1.2.3. Out Message Storage Module Block Diagram

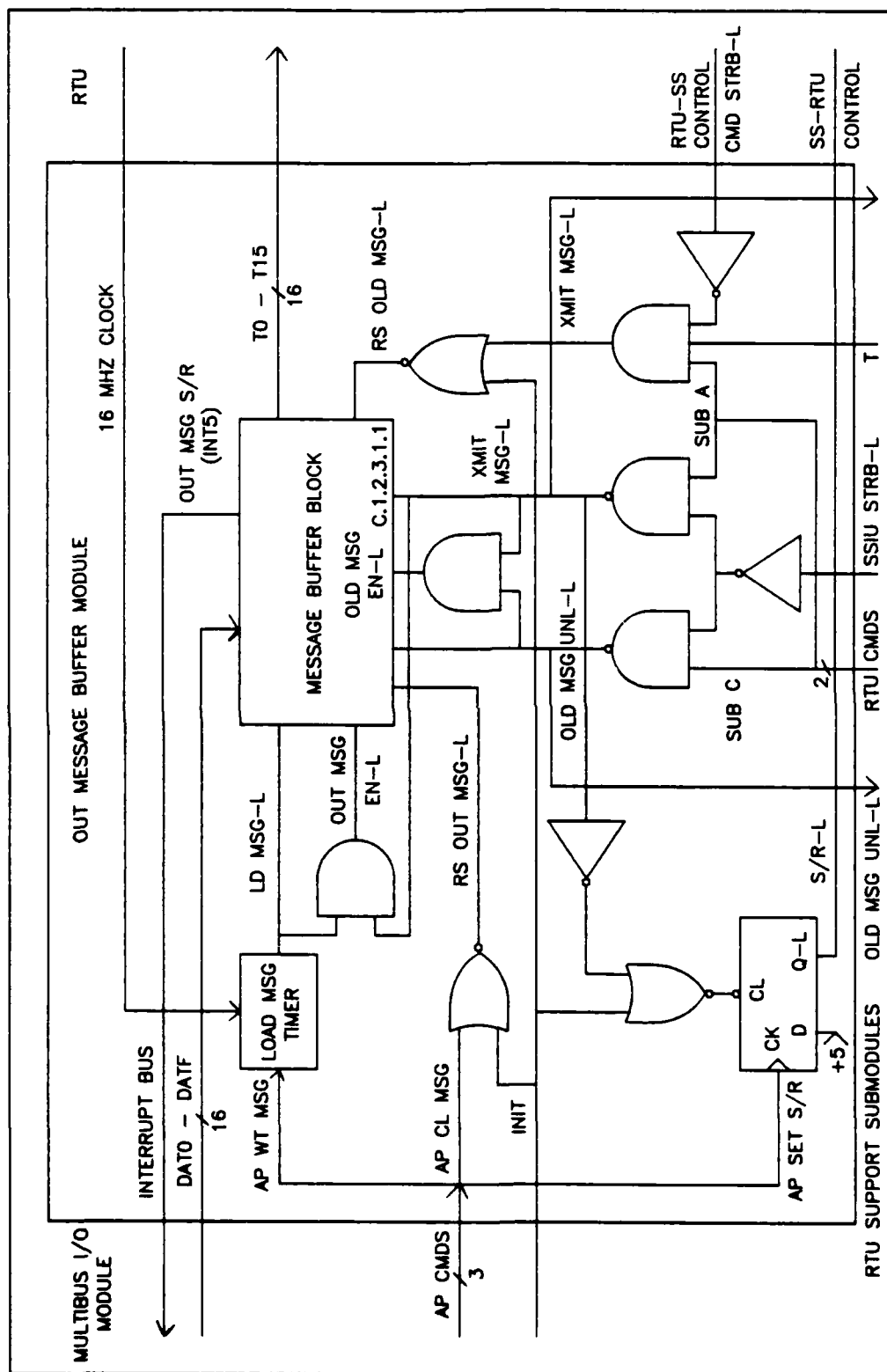


Figure C.1.2.3.1. Out Message Buffer Module Block Diagram

Out Message Buffer Module (C.1.2.3.1) Design Notes

The Load Message Timer (Load MSG Timer) Circuit is included to allow for conditioning of the AP WT MSG signal to match the requirements of the Out Message Buffer's (C.1.2.3.1.1) LD (load) strobe. Signal delays and/or pulse length modifications (non-edge triggered control) may be required.

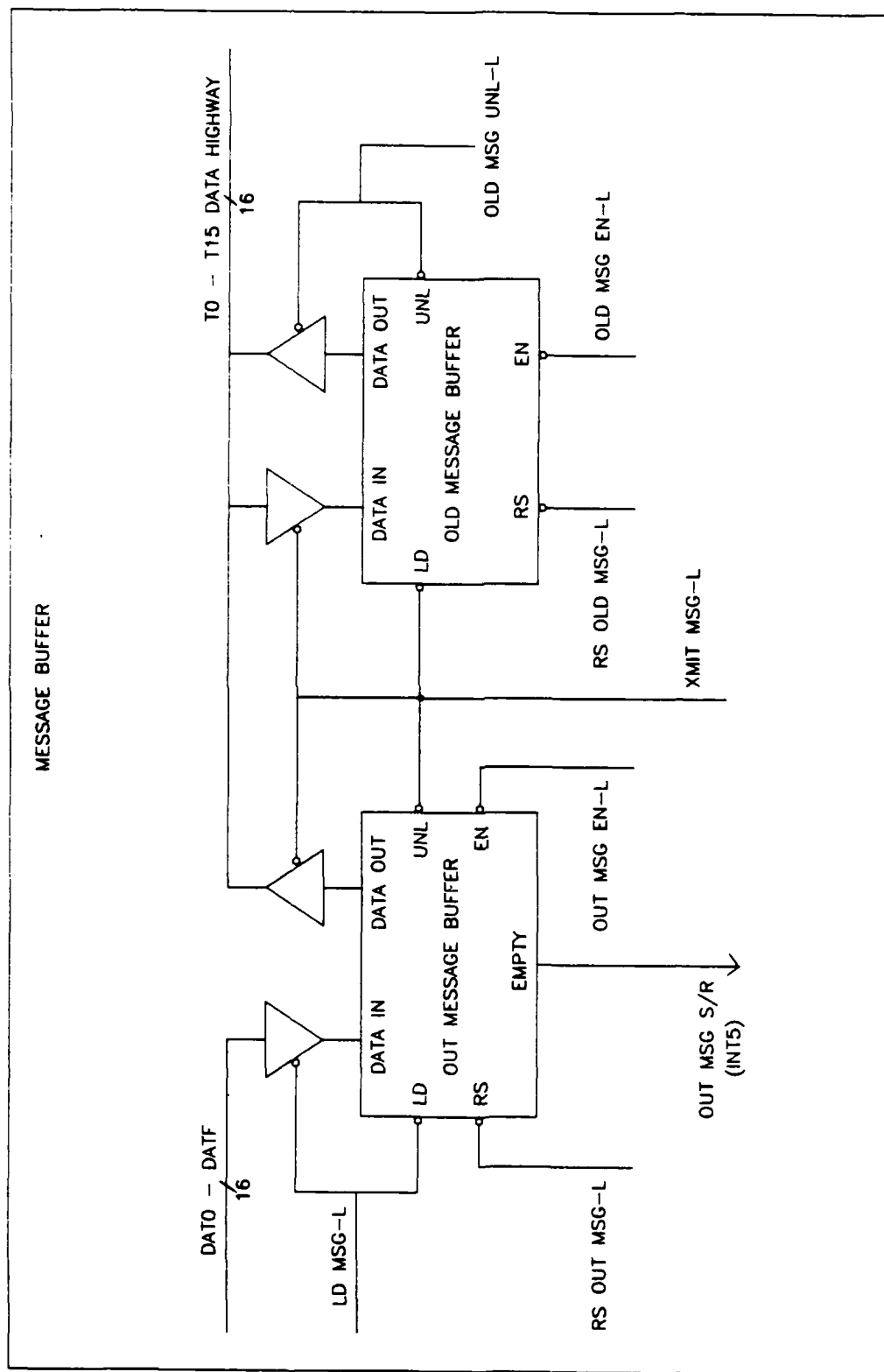


Figure C.1.2.3.1.1. Message Buffer Block Diagram

Message Buffer (C.1.2.3.1.1) Design Notes

1. The required size of both the Out Message Buffer and Old Message Buffer is 16 bits wide (minimum) and 32 words deep. The basic characteristics of the FIFO devices are discussed in the General Design Notes.

2. Additional conditioning of the XMIT MSG-L and OLD MSG UNL-L signals may be necessary to meet the FIFO LD (load) and UNL (unload) strobe requirements, depending upon the the FIFO selected.

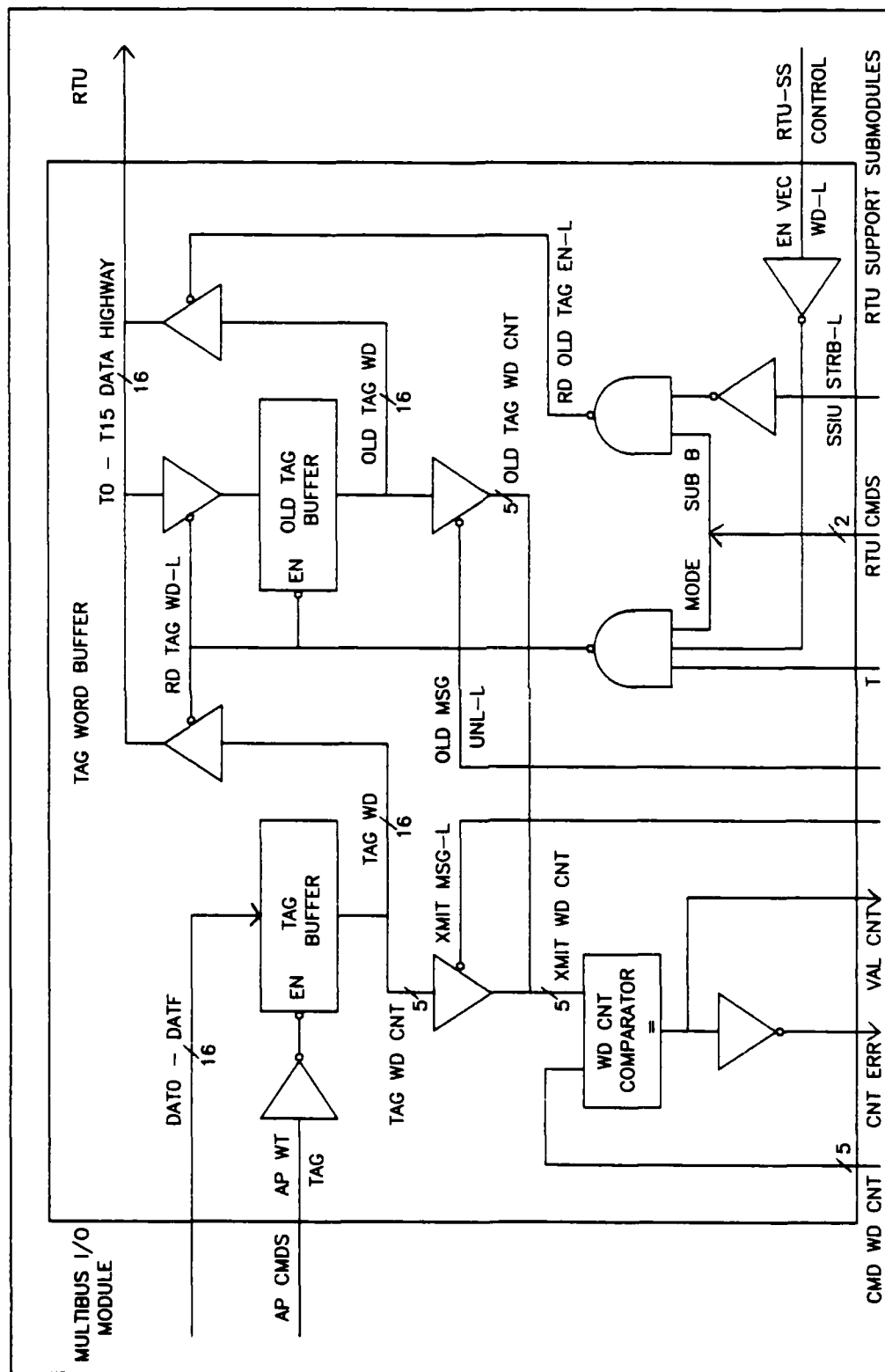


Figure C.1.2.3.2. Tag Word Buffer Module Block Diagram

Tag Word Buffer Module (C.1.2.3.2) Design Notes

A tri-state buffer may be required instead of the normal inverting buffer on the CNT ERR output of the WD CNT Comparator, depending upon the characteristics of the comparator. If an error is generated when one set of inputs is tri-stated off, a tri-state buffer, enabled when either XMIT MSG-L or OLD MSG UNL-L are low, should be substituted.

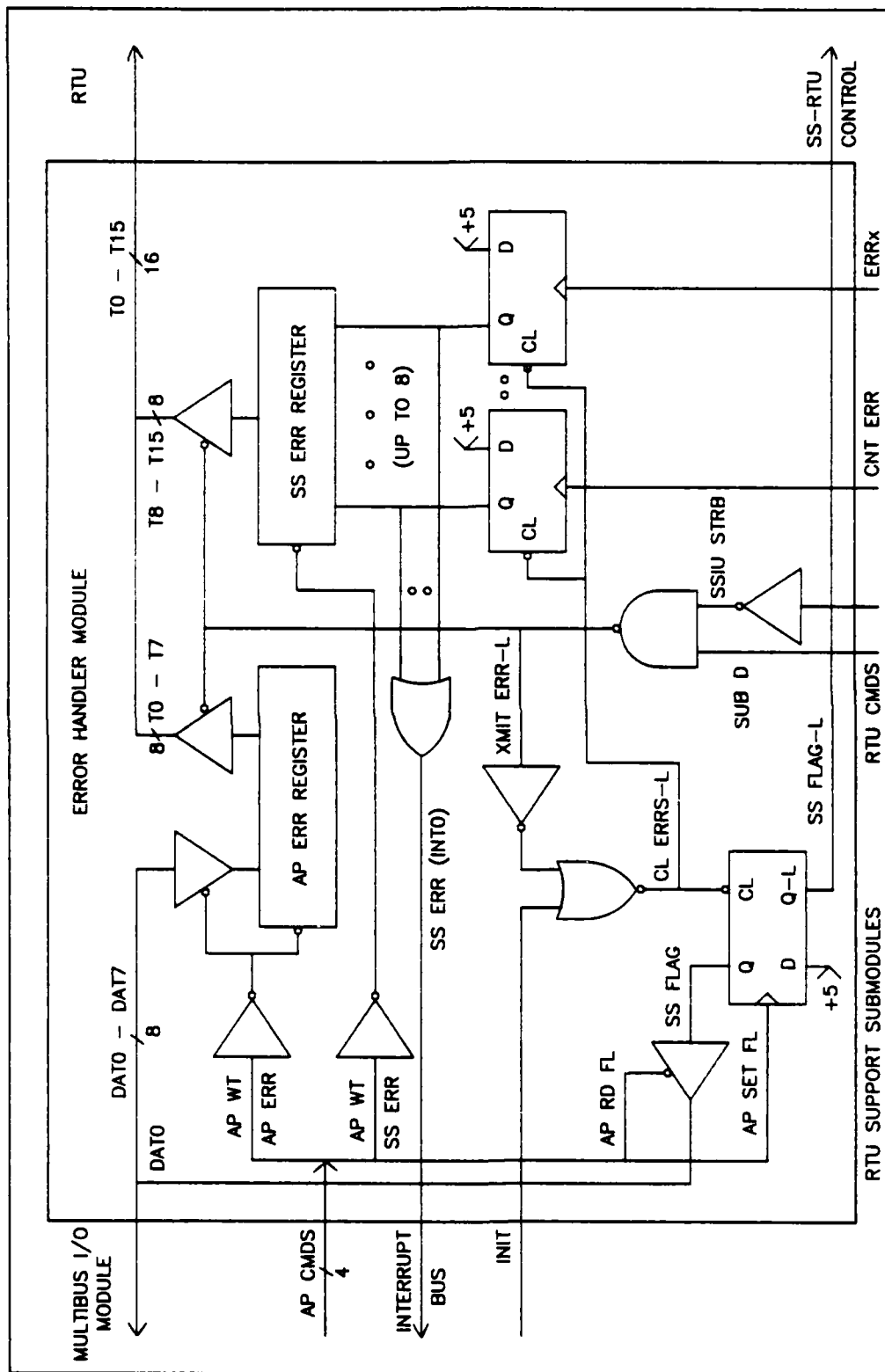


Figure C.1.2.4. Error Handler Module Block Diagram

Error Handler Module (C.1.2.4) Design Notes

The subsystem hardware error portion of this circuit can be expanded to support up to eight hardware errors directly, or up to 256 errors with the addition of encoding hardware. The 1553BBI design requires only CNT ERR for correct operation; extended error handling capability may be required for different applications.

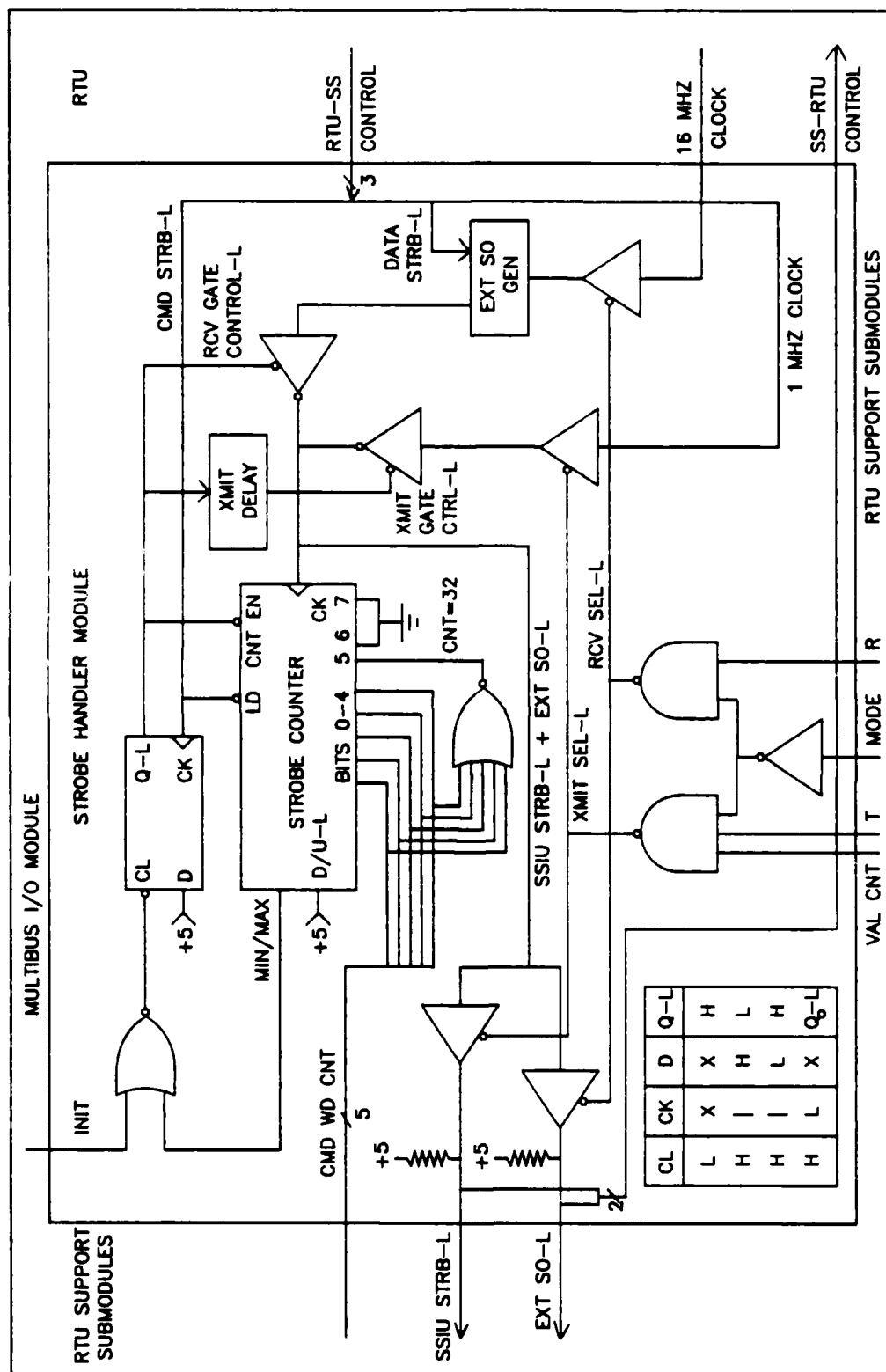


Figure C.1.2.5. Strobe Handler Module Block Diagram

Strobe Handler Module (C.1.2.5) Design Notes

1. The Strobe Counter used in this design must be a synchronous up/down counter, with a positive edge-triggered clock, active low load and enables, a minimum of six input data lines and an active high minimum/maximum count (MIN/MAX) output. The MIN/MAX output must go high on the rising edge of the input pulse that sets the counter into its minimum (COUNT=0) state, and must remain high for a full clock cycle.

2. The Strobe Counter is always used in the count down mode, after being loaded with the command word count (CMD WD CNT). Since loading of 00000 sets the MIN/MAX signal, immediately resetting the D-latch and closing the strobe gates, a sixth bit (CNT=32) must be set for a CMD WD CNT of 32. After the word counter is loaded (under control of CMD STRB-L), and the appropriate strobe gate is opened (under control of XMIT GATE CTRL-L or RCV GATE CNTR-L), the strobes themselves are used for counting. The trailing edge of the active low strobe triggers the count, so that for the last strobe the complete pulse is allowed to pass before MIN/MAX closes the strobe gates.

3. See Appendix E for additional information concerning design timing.

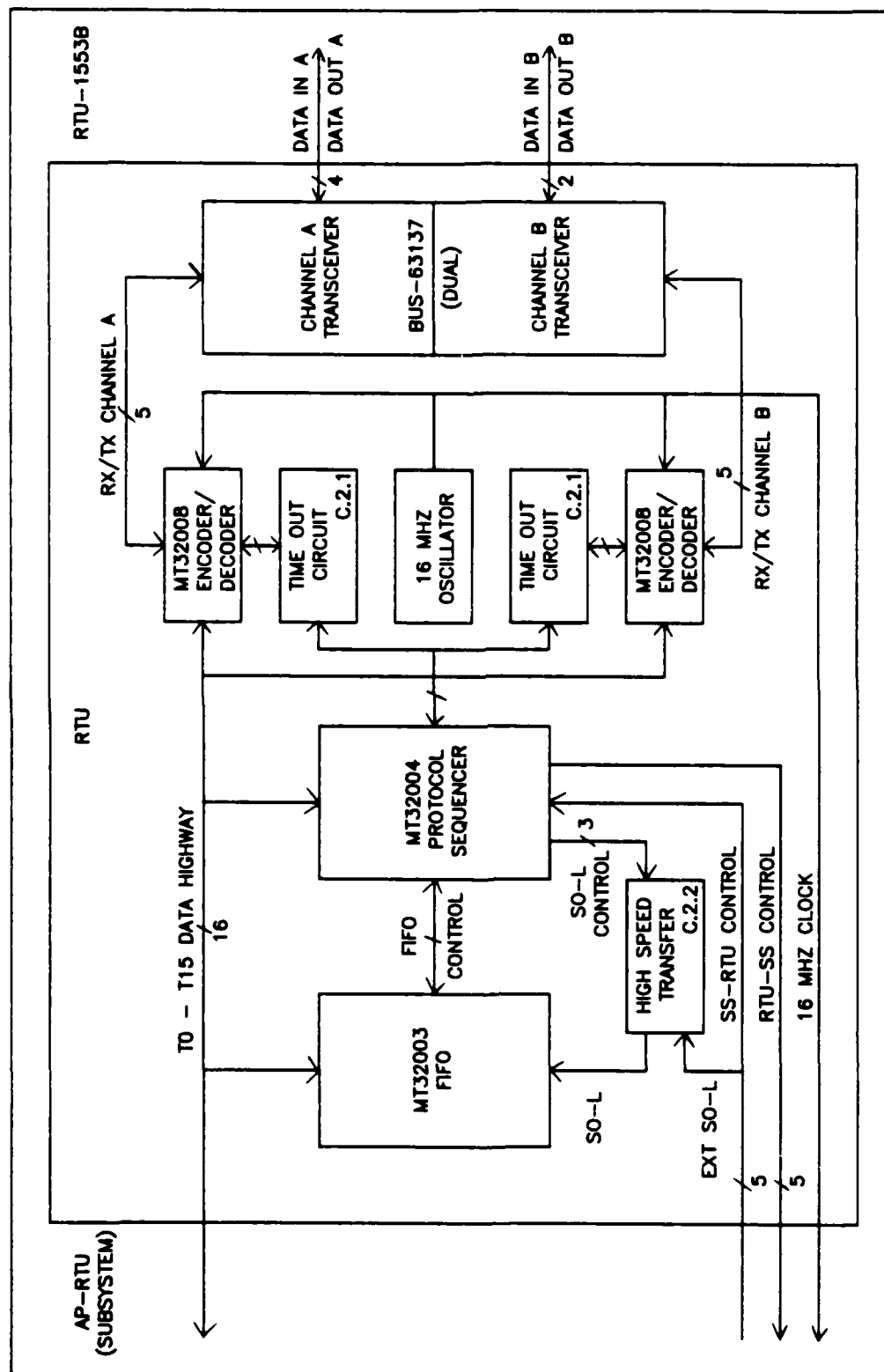


Figure C.2. RTU Module Block Diagram

RTU Module (C.2) Design Notes

1. Internal RTU connections are not shown on this diagram; see Reference 29 for chip pin-outs and interconnections. The Time Out circuitry is considered to be an extension of the internal RTU architecture, since all signals are either input or output directly from RTU components and are transparent to the subsystem and 1553B BC.

2. The RTU chip set chosen for this design consists of five LSI chips: a BUS-63137 Dual Redundant Bus Transceiver, two MT32008 Encoder/Decoders, an MT32004 Protocol Sequencer and an MT32003 FIFO. This chip set, when interfaced to the 1553B bus via the BUS-29854 and BUS-25679 isolation transformers (see Figure C.3), provides all the functionality expected of a "smart" 1553B remote terminal, meeting all MIL-STD-1553B and AF Notice 1 electrical and protocol requirements. A brief description of the features of the RTU components follows.

a. BUS-63137 Dual Redundant MIL-STD-1553B Transceivers. This chip consists of two transmitter/receiver pairs which are compatible with the MCE chip set and the $\pm 12V$ Multibus power. The receivers accept phase-modulated bipolar data at their inputs and produce bi-phase TTL output signals. The transmitters perform the opposite conversion. All receivers and transmitters can be disabled by strobe signals to allow operation to be switched to the other pair in the event of RTU or bus malfunction.

The BUS-63137 is a small (36 pin) LSI chip featuring high reliability, low power consumption, input transmitter protection time

out circuitry and short circuit protection. In accordance with AF Notice 1, the dual transceivers are totally independent, with isolated power and signal leads. This chip, which is equivalent to two 24 pin BUS-63117 chips, was chosen instead of BUS-63117s due to space and cost considerations. See Reference 31 for additional information.

b. MT32008 MIL-STD-1553 Encoder/Decoder. The MT32008 provides the word processing capability of the RTU. In its receiver mode, the decoder checks each incoming 20-bit word for a valid sync field, correct Manchester II encoding, bit count and parity and verifies the addressing. Upon decoding a valid word, the decoder generates a set of status signals and subsequently gates the resulting 16-bit word onto the T0 - T15 Data Highway. In the transmitter mode, the chip encodes the 16-bit word detected on the data highway and adds the sync field and odd parity bit. The MT32008 is a small, low cost, monolithic chip which can be directly connected to the MCE Protocol Sequencer with no additional logic being required. Together with the BUS-63137, two MT32008 chips also provide the complete redundancy required by AF Notice 1 to extend through the RTU circuitry in which command word validation is performed. See Reference 29 for additional information.

c. MT32004 Protocol Sequencer. The Protocol Sequencer provides the message processing capability of the RTU. The Protocol Sequencer is interfaced to the MT32008 encoder/decoder chips and the MT32003 FIFO through a set of control lines and the 16-bit T0 - T15 Data Highway. The chip contains a command register, last command register, BIT register and status register, all of which are accessed from the data highway. The Protocol Sequencer contains the logic to support

message receive and transmit functions, all mode code processing, word count and address error checking. The Protocol Sequencer generates all outgoing subsystem and FIFO control signals, monitors and responds to incoming signals from both, and continuously tracks the FIFO status.

The Protocol Sequencer provides a considerable degree of functionality and reliability for a relatively low cost. The chip is available in a 64-pin package compatible with Multibus power. See Reference 29 for detailed information.

d. MT32003 FIFO. The MT32003 is a 16-bit wide by 32-bit deep FIFO buffer which is required for use with the MT32004 Protocol Sequencer. The FIFO provides bi-directional storage capability, latching and gating message data words off of and onto the T0 - T15 Data Highway under the control of the Protocol Sequencer. The MT32003 is a 40-pin LSI chip.

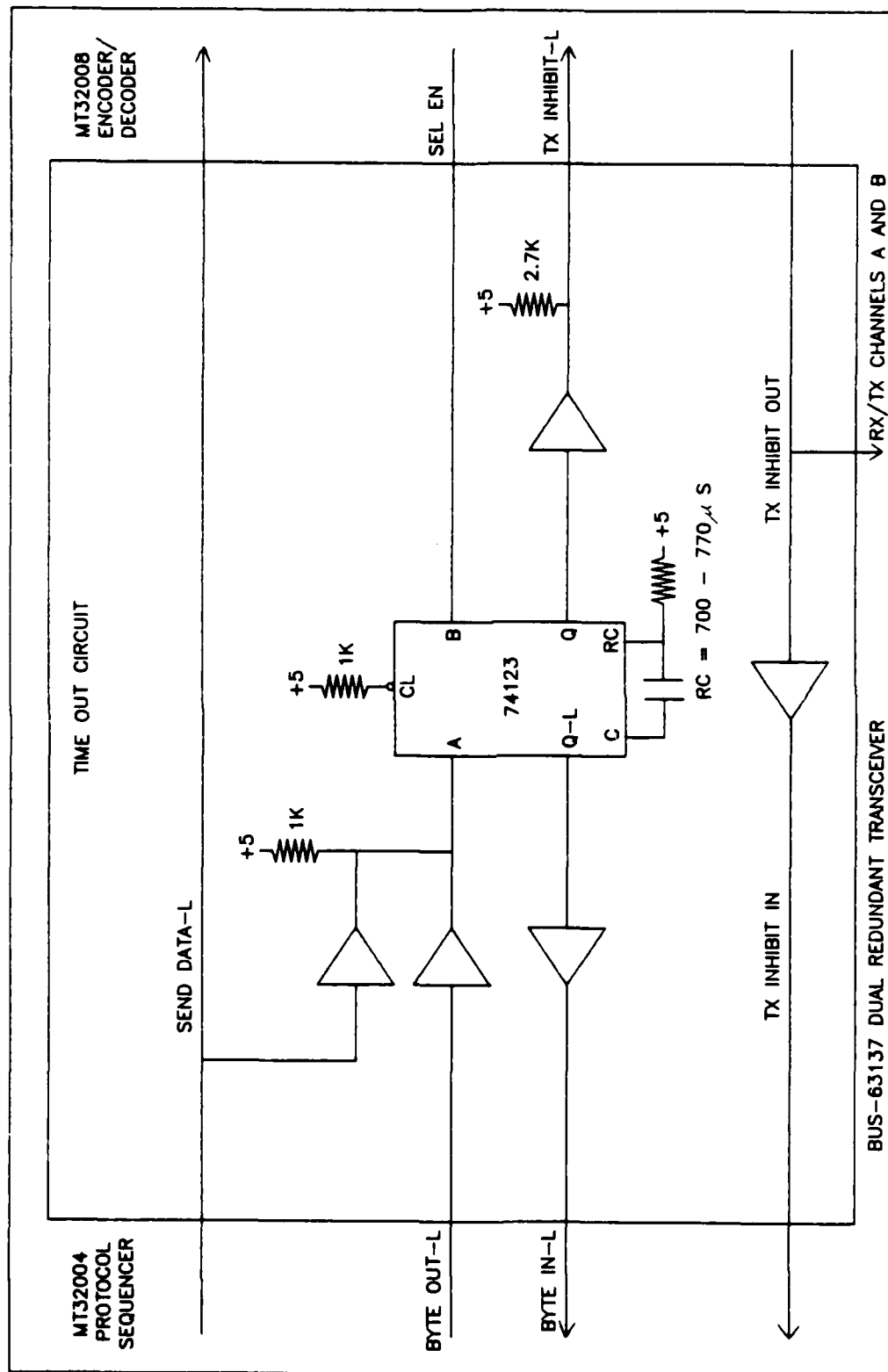


Figure C.2.1. Time Out Circuit Block Diagram

Time Out Circuit (C.2.1) Design Notes

This circuit design is that recommended by the RTU manufacturer (Ref 29:3) for inhibiting transmission of data onto the 1553B bus in the event of a BUS-63137 transceiver failure which results in continuous transmission by that transceiver. Signals for this circuit are not defined, since they are internal signals used solely by the RTU. This circuit should be implemented without modification.

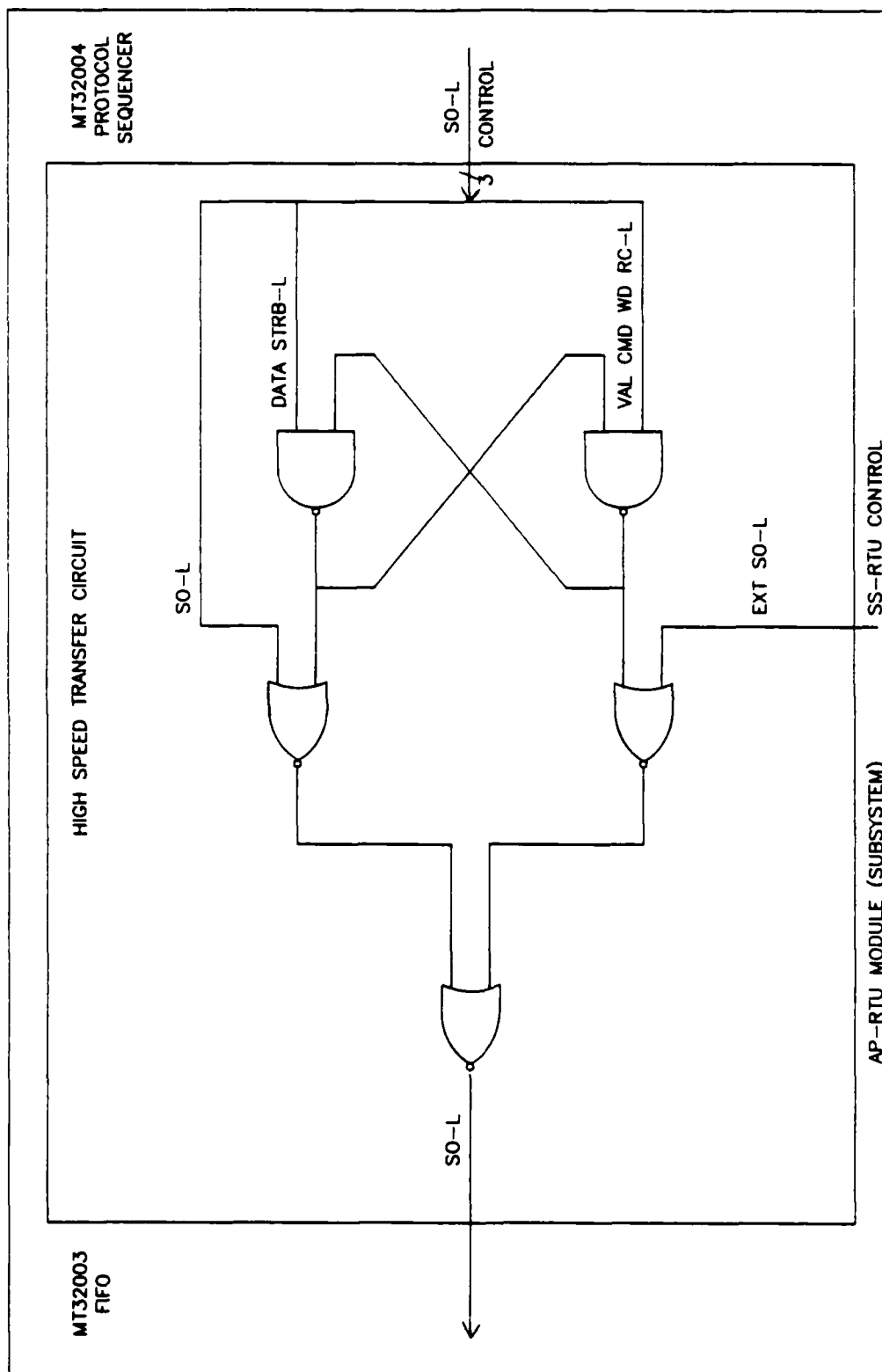


Figure C.2.2. High Speed Transfer Circuit Block Diagram

High Speed Transfer Circuit (C.2.2) Design Notes

This circuit design is that recommended by the RTU manufacturer (Ref 29:130) for supporting high speed data transfers from the MT32003 FIFO to the subsystem when continuous blocks of data are received. This circuit should be implemented without modification.

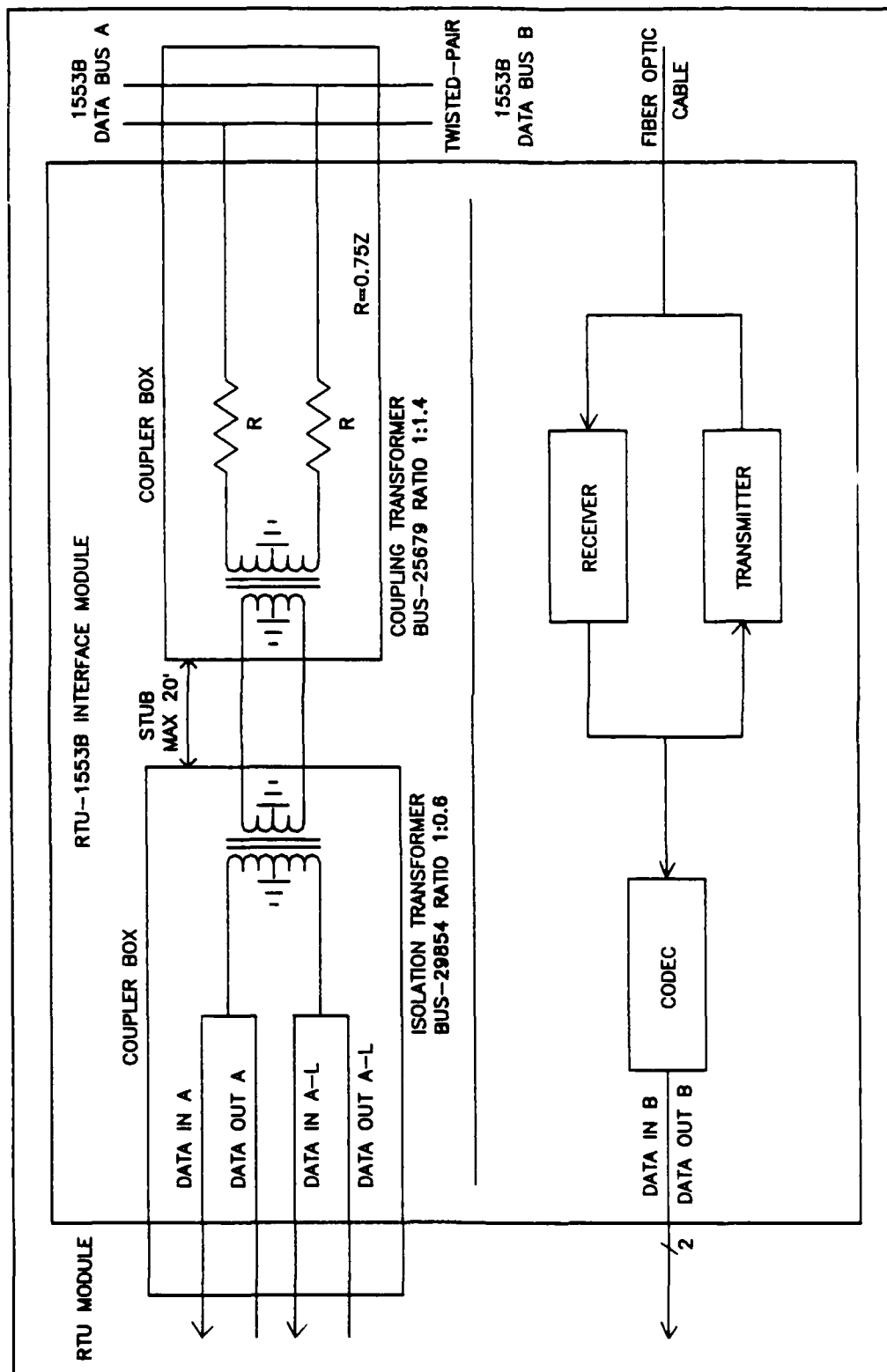


Figure C.3. RTU-1553B Interface Module Block Diagram

RTU-1553B Interface Module (C.3) Design Notes

1. The BUS-29854 (1:0.6 turns ratio) and BUS-25679 (1:1.4 ratio) isolation transformers interface the RTU's BUS-63137 transceiver to the Channel A, twisted-pair 1553B data bus. The transformer-coupled circuit provides the component isolation and common mode rejection ratio characteristics required by MIL-STD-1553B and AF Notice 1. Reference 30 contains a more detailed description of the transformers.

2. The fiber optic interface circuit provides a standard interface between the Channel B fiber optic cable and the the remaining channel of the BUS-63137 transceiver. The fiber optic interface implementation must meet all Channel B transmission requirements as defined at the time of the implementation.

APPENDIX D

HARDWARE SIGNAL DICTIONARY

SIGNAL: 1 MHZ CLOCK

FULL SIGNAL NAME: 1 MHz Protocol Sequencer Clock

SIGNAL TYPE: RTU-SS CONTROL

DESCRIPTION: The 1 MHz RTU Protocol Sequencer clock output is used to synchronize the subsystem and RTU operation during data message transmission. The Command Handler Module (C.1.2.1) derives SSIU STRB-L from this clock in order to meet the synchronization requirement.

SIGNAL: 16 MHZ CLOCK

FULL SIGNAL NAME: RTU 16 MHz System Clock

SIGNAL TYPE: Clock

DESCRIPTION: The 16 MHz RTU clock oscillator (C.2) from which all RTU and AP-RTU Module timing is derived.

SIGNAL: ADRI - ADRF

FULL SIGNAL NAME: AP-RTU Address Bus

SIGNAL TYPE: 15-Bit address bus

DESCRIPTION: The active high 15-bit wide address bus derived from ADDRESS BUS-L and used by the Multibus Decoder Module (C.1.1.1) Address/Control Decoder in generating AP CMDS, BS, DIR and XACK TRIGGER. The lowest ADDRESS BUS-L bit (ADRO-L), is not needed since only 16-bit data transfers are supported.

SIGNAL: ADDRESS BUS-L

FULL SIGNAL NAME: Multibus Address Bus, ADRO-L - ADRF-L

SIGNAL TYPE: 16-Bit address bus (active low)

DESCRIPTION: The 16-bit wide active low Multibus address bus.

SIGNAL: AP CL INTO

FULL SIGNAL NAME: AP Clear Interrupt 0

SIGNAL TYPE: AP CMD

DESCRIPTION: AP CMD to clear Interrupt Circuit Module INTO latch
(C.1.1.3) as prescribed by non-bus vectored interrupt servicing.

SIGNAL: AP CL INT1

FULL SIGNAL NAME: AP Clear Interrupt 1

SIGNAL TYPE: AP CMD

DESCRIPTION: AP CMD to clear Interrupt Circuit Module INT1 latch
(C.1.1.3) as prescribed by non-bus vectored interrupt servicing.

SIGNAL: AP CL INT3

FULL SIGNAL NAME: AP Clear Interrupt 3

SIGNAL TYPE: AP CMD

DESCRIPTION: AP CMD to clear Interrupt Circuit Module INT3 latch
(C.1.1.3) as prescribed by non-bus vectored interrupt servicing.

SIGNAL: AP CL INT5

FULL SIGNAL NAME: AP Clear Interrupt 5

SIGNAL TYPE: AP CMD

DESCRIPTION: AP CMD to clear Interrupt Circuit Module INT5 latch
(C.1.1.3) as prescribed by non-bus vectored interrupt servicing.

SIGNAL: AP CL MSG

FULL SIGNAL NAME: AP Clear Message Buffer

SIGNAL TYPE: AP CMD

DESCRIPTION: AP CMD which clears the Out Message Buffer Module's Out Message Buffer (C.1.2.3.1) prior to loading the block of data words which composes the next outgoing message.

SIGNAL: AP CL WRAP

FULL SIGNAL NAME: AP Clear Wrap Around Enable

SIGNAL TYPE: AP CMD

DESCRIPTION: AP CMD which resets the Command Handler Module latch (C.1.2.1) which generates WRAP EN to disable the RTU Protocol Sequencer Wrap Around function.

SIGNAL: AP CMDS

FULL SIGNAL NAME: AP Commands

SIGNAL TYPE: Control signal bus

DESCRIPTION: The collection of control signal lines driven by the Multibus Decoder Module's Address/Control Decoder (C.1.1.1). For each decoding operation, a single AP CMD signal is generated from the combination of the ADDRESS BUS-L, MRDC-L AND MWTC-L Multibus control lines. Each signal is individually responsible for initiating a separate 1553BBI Module (C.0) function.

COMPONENTS:	AP CL INTO	AP CL INT1
	AP CL INT3	AP CL INT5
	AP CL MSG	AP CL WRAP
	AP RD CMD	AP RD FL
	AP RD MSG	AP SET FL
	AP SET S/R	AP SET WRAP
	AP WT AP ERR	AP WT MSG
	AP WT SS ERR	AP WT TAG

SIGNAL: AP RD CMD

FULL SIGNAL NAME: AP Read 1553B Command Word

SIGNAL TYPE: AP CMD

DESCRIPTION: AP CMD which enables the 1553B command word from the Command Handler Module's Command Buffer (C.1.2.1) onto DAT0 - DATF for reading by the AP.

SIGNAL: AP RD FL

FULL SIGNAL NAME: AP Read Subsystem Flag Status

SIGNAL TYPE: AP CMD

DESCRIPTION: AP CMD enables SS FLAG onto DAT0 for reading by the AP prior to updating the Error Handler Module's AP ERR Register (C.1.2.4). If SS FLAG is set, it indicates that the BC has not yet responded to SS FLAG-L. The AP may then decide to wait until servicing is complete before writing the new error condition, or may rewrite the AP ERR REGISTER contents using a code that reflects the combination of errors.

SIGNAL: AP RD MSG

FULL SIGNAL NAME: AP Read Message Word

SIGNAL TYPE: AP CMD

DESCRIPTION: AP CMD which enables the word in the first location of the In Message Storage Module's In FIFO Buffer (C.1.2.2) onto DAT0 - DATF for reading by the AP.

SIGNAL: AP SET FL

FULL SIGNAL NAME: AP Set Subsystem Flag

SIGNAL TYPE: AP CMD

DESCRIPTION: AP CMD which sets the latch in the Error Handler Module (C.1.2.4) to generate SS FLAG-L. The latch is reset by CL ERRS-L.

SIGNAL: AP SET S/R

FULL SIGNAL NAME: AP Set Service Request

SIGNAL TYPE: AP CMD

DESCRIPTION: AP CMD which sets the latch in the Out Message Buffer Module (C.1.2.3.1) to generate S/R-L. The latch is reset by INIT or by XMIT MSG-L, the signal which controls the servicing of the request.

SIGNAL: AP SET WRAP

FULL SIGNAL NAME: AP Set Wrap Around Enable

SIGNAL TYPE: AP CMD

DESCRIPTION: AP CMD which sets Command Handler Module latch (C.1.2.1) which generates WRAP EN. Together with AP CL WRAP, this command provides an alternative to normal hardwiring of the WRAP EN input. See WRAP AROUND CMD for another implementation alternative.

SIGNAL: AP WT AP ERR

FULL SIGNAL NAME: AP Write AP Error Register

SIGNAL TYPE: AP CMD

DESCRIPTION: AP CMD which initiates the loading of a byte of data from DAT0 - DAT7 into the Error Handler Module's 8-bit AP ERR Register (C.1.2.4). Register contents are gated onto data lines T0 - T7 when a XMIT ERR command is received.

SIGNAL: AP WT MSG

FULL SIGNAL NAME: AP Write Message Word

SIGNAL TYPE: AP CMD

DESCRIPTION: AP CMD which initiates the loading of a data word from DAT0 - DATF into the Out Message Buffer Module's Out Message Buffer (C.1.2.3.1).

SIGNAL: AP WT SS ERR

FULL SIGNAL NAME: AP Write Subsystem Error Register

SIGNAL TYPE: AP CMD

DESCRIPTION: AP CMD which initiates the loading of the Error Handler Module's 8-bit SS ERR Register (C.1.2.4) from the set of eight hardware error latches. Register contents are gated onto data lines T8 - T15 when a XMIT ERR command is received.

SIGNAL: AP WT TAG

FULL SIGNAL NAME: AP Write Tag Word

SIGNAL TYPE: AP CMD

DESCRIPTION: AP CMD which initiates the loading of a data word from DAT0 - DATF into the Tag Word Buffer Module's Tag Buffer (C.1.2.3.2).

SIGNAL: BS

FULL SIGNAL NAME: Board Select

SIGNAL TYPE: Control signal

DESCRIPTION: Signal generated by the Multibus Decoder Module Address/Control Decoder (C.1.1.1) which is set when a valid 1553BBI Module (C.0) address is detected. BS enables the Data Transfer Module (C.1.1.2).

SIGNAL: CL ERRS-L

FULL SIGNAL NAME: Clear Subsystem Error Latches

SIGNAL TYPE: Control signal (active low)

DESCRIPTION: CL ERRS-L is used to reset the latch which generates SS FLAG-L and all of the subsystem hardware error latches in the Error Handler Module (C.1.2.4). The signal is the NOR of INIT and XMIT ERR, the signal generated when the BC responds to the service requested by SS FLAG-L.

SIGNAL: CMD STRB-L

FULL SIGNAL NAME: Command Strobe

SIGNAL TYPE: RTU-SS CONTROL (active low)

DESCRIPTION: CMD STRB-L is an 8.5 microsecond signal generated by the RTU following completion of the internal RTU receive or transmit command processing. CMD STRB-L signals the subsystem that the command word is available for reading from the T0 - T15 Data Highway. The AP-RTU Module uses CMD STRB-L to trigger the opening of the strobe gates in the Strobe Handler Module (C.1.2.5) and to trigger the resetting of the Out Message Buffer Module's Old Message Buffer (C.1.2.3.1).

SIGNAL: CMD WD CNT

FULL SIGNAL NAME: Command Word Count

SIGNAL TYPE: Word count

DESCRIPTION: The 5-bit word count (bits 4 - 0) derived from the Command Handler Module's Command Buffer (C.1.2.1). A comparison of CMD WD CNT and XMIT WD CNT is performed by the Tag Word Buffer Module (C.1.2.3.2) as a rough verification of the BC's process management operation prior to message transmission.

SIGNAL: CNT=32

FULL SIGNAL NAME: Word Count 32

SIGNAL TYPE: Counter input

DESCRIPTION: CNT=32 is used to set Bit 5 of the Strobe Handler Module's Strobe Counter input count when the message word count equals 32 (CMD WD CNT=00000). Without introducing this additional counter input, whenever a 32 word count is detected, the counter will generate MIN/MAX, resetting the strobe gate control latch and aborting the strobe generation sequence.

SIGNAL: CNT ERR

FULL SIGNAL NAME: Transmit Command Count Error

SIGNAL TYPE: Status signal

DESCRIPTION: CNT ERR is one of the eight hardware error status signals latched by the Error Handler Module (C.1.2.4) and enabled into the SS ERR Register by the AP WT SS ERR command. CNT ERR is generated by the Tag Word Buffer Module (C.1.2.3.2) when the transmit command word count (CMD WD CNT) does not equal the transmit word count (XMIT WD CNT).

SIGNAL: DATO - DATF

FULL SIGNAL NAME: AP-RTU Module Data Bus

SIGNAL TYPE: 16-Bit data bus

DESCRIPTION: The inverted version of DATA BUS-L. The Data Transfer Module (C.1.1) performs the bidirectional inversion of the data bus lines.

SIGNAL: DATA BUS-L

FULL SIGNAL NAME: Multibus Data Bus, DATO-L - DATF-L

SIGNAL TYPE: 16-Bit data bus (active low)

DESCRIPTION: The 16-bit wide Multibus data bus.

SIGNAL: DATA IN A

FULL SIGNAL NAME: Data In Channel A

SIGNAL TYPE: Serial data bus

DESCRIPTION: The Manchester encoded serial input data bus which links the Channel A BUS-63137 Transceiver (C.2) to the RTU-1553 Interface Module. In Channel A of Figure C.3 this signal is shown as its two component lines since the active high and active low inputs have separate physical connections to the transformer.

COMPONENTS: DATA IN A
DATA IN A-L

SIGNAL: DATA IN B

FULL SIGNAL NAME: Data In Channel B

SIGNAL TYPE: Serial data bus

DESCRIPTION: The Manchester encoded serial input data bus which links the Channel B BUS-63137 Transceiver (C.2) to the RTU-1553 Interface Module (C.3).

SIGNAL: DATA OUT A

FULL SIGNAL NAME: Data Out Channel A

SIGNAL TYPE: Serial data bus

DESCRIPTION: The Manchester encoded serial output data bus which links the Channel A BUS-63137 Transceiver (C.2) to the RTU-1553 Interface Module. In Channel A of Figure C.3 this signal is shown as its two component lines since the active high and active low inputs have separate physical connections to the transformer.

COMPONENTS: DATA OUT A
DATA OUT A-L

SIGNAL: DATA OUT B

FULL SIGNAL NAME: Data Out Channel B

SIGNAL TYPE: Serial data bus

DESCRIPTION: The Manchester encoded serial output data bus which links the Channel B BUS-63137 Transceiver (C.2) to the RTU-1553 Interface Module (C.3).

SIGNAL: DATA STRB-L

FULL SIGNAL NAME: Data Strobe

SIGNAL TYPE: RTU-SS CONTROL (active low)

DESCRIPTION: DATA STRB-L is signal normally responsible for controlling the transfer of data words from the RTU; to the subsystem. While SO-L gates the data onto the bus, the subsystem uses DATA STRB-L to enable its buffers and latch the data. When the High Speed Transfer Circuit is used, DATA STRB-L serves only as the trigger which initiates EXT SO-L generation by the Strobe Handler Module (C.1.2.5).

SIGNAL: DIR

FULL SIGNAL NAME: Direction

SIGNAL TYPE: Control signal

DESCRIPTION: Signal generated by the Multibus Decoder Module's Address/Control Decoder (C.1.1.1) which provides the transfer direction to the Data Transfer Module (C.1.1.2).

SIGNAL: EN VEC WD-L

FULL SIGNAL NAME: Enable Vector Word

SIGNAL TYPE: RTU-SS CONTROL (active low)

DESCRIPTION: EN VEC WD-L is generated by the RTU Protocol Sequencer in response to a transmit vector word mode command. The signal is used to trigger the gating of the Tag Word Buffer Module's Tag Buffer (C.1.2.3.2) onto T0 - T15 for transfer to the RTU and for loading into the Old Tag Buffer.

SIGNAL: ERRx

FULL SIGNAL NAME: Hardware Error X

SIGNAL TYPE: Status signal

DESCRIPTION: One of seven as yet undefined hardware error status signals available to the system implementer. ERRx is latched by the Error Handler Module (C.1.2.4) and enabled into the SS ERR Register by the AP WT SS ERR command.

SIGNAL: EXT SO-L

FULL SIGNAL NAME: External Strobe Out

SIGNAL TYPE: SS-RTU CONTROL (active low)

DESCRIPTION: The 2.0 MHz receive strobe signal generated by the Strobe Handler Module (C.1.2.5) which replaces DATA STRB-L in controlling the subsystem's gating of the individual data words (one per strobe) onto T0 - T15 and from which the RTU derives a new SO-L signal to control the subsequent higher-speed latching of the words from the data highway.

SIGNAL: FIFO CONTROL

FULL SIGNAL NAME: RTU FIFO Control Bus

SIGNAL TYPE: Internal RTU control bus

DESCRIPTION: The RTU Protocol Sequencer (C.2) uses this set of signals to control the FIFO's interactions with the T0 - T15 Data Highway.

COMPONENTS: See Reference 29.

SIGNAL: INIT

FULL SIGNAL NAME: 1553BBI Initialize

SIGNAL TYPE: Control signal

DESCRIPTION: Inverted version of INIT-L signal used to reset the 1553BBI Module (C.0).

SIGNAL: INIT-L

FULL SIGNAL NAME: Multibus Initialize

SIGNAL TYPE: Control signal (active low)

DESCRIPTION: Multibus initialize/system reset signal.

SIGNAL: IN MSG EN-L

FULL SIGNAL NAME: In Message Buffer Enable

SIGNAL TYPE: Control signal (active low)

DESCRIPTION: This signal enables the In Message Storage Module's In FIFO Buffer (C.1.2.2) for loading and unloading operations if such a signal is required by the selected FIFO hardware.

SIGNAL: INTO, SS ERR

FULL SIGNAL NAME: AP-RTU Interrupt 0, Subsystem Error

SIGNAL TYPE: Interrupt signal

DESCRIPTION: Inverted version of INTO-L. SS ERR is generated by the Error Handler Module (C.1.2.4) to notify the AP of detected non-RTU 1553BBI hardware errors.

SIGNAL: INT1, IN BUFFER FULL

FULL SIGNAL NAME: AP-RTU Interrupt 1, In Message FIFO Buffer Full

SIGNAL TYPE: Interrupt signal

DESCRIPTION: Inverted version of INT1-L. IN BUFFER FULL is generated by the In Message Storage Module (C.1.2.2) to notify the AP that the incoming message storage is exhausted. The AP requests the BC to hold incoming messages by writing the appropriate error condition code to the Error Handler Module's AP ERR Register and setting SS FLAG.

SIGNAL: INT3, IN MSG S/R

FULL SIGNAL NAME: AP-RTU Interrupt 3, In Message Service Request

SIGNAL TYPE: Interrupt signal

DESCRIPTION: Inverted version of INT3-L. IN MSG S/R is generated by the In Message Storage Module (C.1.2.2) to notify the AP that incoming messages are stored in the In FIFO Buffer and awaiting transfer to the AP.

SIGNAL: INT5, OUT MSG S/R

FULL SIGNAL NAME: AP-RTU Interrupt 5, Out Message Service Request

SIGNAL TYPE: Interrupt signal

DESCRIPTION: Inverted version of INT5-L. OUT MSG S/R is generated by the Out Message Buffer Module (C.1.2.3.1.1) to notify the AP that the next outgoing message can be written to the Out Message Buffer for subsequent transmission.

SIGNAL: INTO-L

FULL SIGNAL NAME: Multibus Interrupt 0

SIGNAL TYPE: Interrupt signal (active low)

DESCRIPTION: The highest priority Multibus interrupt.

SIGNAL: INT1-L

FULL SIGNAL NAME: Multibus Interrupt 1

SIGNAL TYPE: Interrupt signal (active low)

DESCRIPTION: The second highest priority Multibus interrupt.

SIGNAL: INT3-L

FULL SIGNAL NAME: Multibus Interrupt 3

SIGNAL TYPE: Interrupt signal (active low)

DESCRIPTION: The fourth priority level Multibus interrupt.

SIGNAL: INT5-L

FULL SIGNAL NAME: Multibus Interrupt 5

SIGNAL TYPE: Interrupt signal (active low)

DESCRIPTION: The sixth priority level Multibus interrupt.

SIGNAL: INTERRUPT BUS

FULL SIGNAL NAME: AP-RTU Module Interrupt Bus

SIGNAL TYPE: Interrupt bus

DESCRIPTION: The inverted version of INTERRUPT BUS-L.

COMPONENTS: INTO, SS ERR
INT1, IN BUFFER FULL
INT3, IN MSG S/R
INT5, OUT MSG S/R

SIGNAL: INTERRUPT BUS-L

FULL SIGNAL NAME: Multibus Interrupt Bus

SIGNAL TYPE: Interrupt bus (active low)

DESCRIPTION: The Multibus interrupt bus. In this non-bus vectored implementation, eight interrupts are supported by the 86/12A AP's 8259A Programmable Interrupt Controller. Only four of these interrupts are available to the 1553BBI.

COMPONENTS: INTO-L
INT1-L
INT3-L
INT5-L

SIGNAL: LD IN MSG-L

FULL SIGNAL NAME: Load In Message Buffer

SIGNAL TYPE: Control signal (active low)

DESCRIPTION: LD IN MSG-L provides In Message Storage Module (C.1.2.2) buffer enables and In FIFO Buffer loading controls as required by the FIFO selected for the implementation.

SIGNAL: LD MSG-L

FULL SIGNAL NAME: Load Out Message Buffer

SIGNAL TYPE: Control signal (active low)

DESCRIPTION: LD MSG-L, generated by the Out Message Buffer's Load Message Timer (LOAD MSG TIMER) from AP WT MSG and 16 MHZ CLOCK, enables the output buffers and controls the loading of the Old Message Buffer as required by the FIFO selected for the implementation.

SIGNAL: MIN/MAX

FULL SIGNAL NAME: Minimum/Maximum Count

SIGNAL TYPE: Control signal

DESCRIPTION: MIN/MAX is the signal generated by the Strobe Handler Module's Strobe Counter (C.1.2.5) whenever its minimum or maximum count value is detected in its count registers. For this application, the counter is loaded with CMD WD CNT or 10000 (see CNT=32), which is always a value less than the maximum count. As the strobes are generated, the count is decremented until zero is reached, indicating that the required number of strobes has been output. The MIN/MAX signal, which goes high at this point, resets the strobe gate control latch, shutting down the strobe output.

SIGNAL: MODE

FULL SIGNAL NAME: Mode Command

SIGNAL TYPE: RTU CMD

DESCRIPTION: MODE is set when the Command Handler Module's Subaddress Decoder (C.1.2.1) detects subaddress field values 00000 or 11111, which indicate mode commands. MODE is used with T to enable transmission of the tag word (TAG WD) from the Tag Word Buffer Module's Tag Buffer (C.1.2.3.2) when triggered by EN VEC WD-L. NOT MODE is used to enable the Strobe Handler Module's (C.1.2.5) generation of SSIU STRB-L and EXT SO-L.

SIGNAL: MRDC

FULL SIGNAL NAME: Memory Read Control

SIGNAL TYPE: Control signal

DESCRIPTION: The inverted version of MRDC-L. MRDC is used by the Multibus Decoder Module's Address/Control Decoder (C.1.1.1) in generating DIR, XACK TRIGGER and AP CMDS.

SIGNAL: MRDC-L

FULL SIGNAL NAME: Multibus Memory Read Control

SIGNAL TYPE: Control signal (active low)

DESCRIPTION: The Multibus memory-mapped input/output read signal which controls the CPU's reading of a data word from the 1553BBI.

SIGNAL: MWTC

FULL SIGNAL NAME: Memory Write Control

SIGNAL TYPE: Control signal

DESCRIPTION: The inverted version of MWTC-L. MWTC is used by the Multibus Decoder Module's Address/Control Decoder (C.1.1.1) in generating DIR, XACK TRIGGER and AP CMDS.

SIGNAL: MWTC-L

FULL SIGNAL NAME: Multibus Memory Write Control

SIGNAL TYPE: Control signal (active low)

DESCRIPTION: The Multibus memory-mapped input/output write signal which controls the CPU's writing of a data word to the 1553BBI.

SIGNAL: OLD MSG EN-L

FULL SIGNAL NAME: Old Message Buffer Enable

SIGNAL TYPE: Control signal (active low)

DESCRIPTION: This signal enables the Out Message Storage Module's Old Message Buffer FIFO (C.1.2.3.1.1) for loading and unloading operations if such a signal is required by the selected FIFO hardware.

SIGNAL: OLD MSG UNL-L

FULL SIGNAL NAME: Unload Old Message Buffer

SIGNAL TYPE: Control signal (active low)

DESCRIPTION: OLD MSG UNL-L, generated from SUB C and SSIU STRB-L, enables the output buffers and controls the unloading of the OLD MESSAGE BUFFER as required by the FIFO selected for the implementation.

SIGNAL: OLD TAG WD

FULL SIGNAL NAME: Old Tag Word

SIGNAL TYPE: 16-Bit word

DESCRIPTION: OLD TAG WD is the tag word associated with the previously transmitted data message which is stored in the Out Message Buffer Module's Old Message Buffer (C.1.2.3.1.1), in order to provide a transmission error recovery mechanism.

SIGNAL: OLD TAG WD CNT

FULL SIGNAL NAME: Old Tag Word Count

SIGNAL TYPE: Word count

DESCRIPTION: See XMIT WD CNT.

SIGNAL: OUT MSG EN-L

FULL SIGNAL NAME: Out Message Buffer Enable

SIGNAL TYPE: Control signal (active low)

DESCRIPTION: This signal enables the Out Message Storage Module's Out Message Buffer FIFO (C.1.2.3.1.1) for loading and unloading operations if such a signal is required by the selected FIFO hardware.

SIGNAL: R

FULL SIGNAL NAME: Receive

SIGNAL TYPE: Control signal

DESCRIPTION: R is derived from Bit 10 (T/R) of the Command Handler Module's Command Buffer (C.1.2.1) and is the inverse of T. When high, R enables generation of the receive strobes (EXT SO-Ls).

SIGNAL: RCV GATE CONTROL-L

FULL SIGNAL NAME: Receive Strobe Gate Control

SIGNAL TYPE: Control signal (active low)

DESCRIPTION: RCV GATE CONTROL-L controls the enabling of the tri-state buffer which allows passage of the EXT SO strobes from the Strobe Handler Module's EXT SO Generator (EXT SO GEN) circuit (C.1.2.5). This strobe also passes through the Transmit Delay (XMIT DELAY) circuit prior to opening the window to allow passage of the 1 MHZ CLOCK which is used in producing the transmit strobes (SSIU STRB-L). See the receive and transmit timing diagrams in Appendix E for the rationale behind the use of the strobe gates.

SIGNAL: RCV SEL-L

FULL SIGNAL NAME: Receive Select

SIGNAL TYPE: Control signal (active low)

DESCRIPTION: RCV SEL-L is the receive command enable signal generated from R and NOT MODE. This signal controls the gating of the 16 MHZ CLOCK into the Strobe Handler Module (C.1.2.5) as well as the gating of the EXT SO-L from the module to the RTU and the In Message Storage Module.

SIGNAL: RD OLD TAG EN-L

FULL SIGNAL NAME: Read Old Tag Word Enable

SIGNAL TYPE: Control signal (active low)

DESCRIPTION: RD OLD TAG EN-L enables the Tag Word Buffer Module's OLD TAG WD (C.1.2.3.2) onto T0 - T15 for transfer to the RTU. RD OLD TAG EN-L is generated from SUB B and SSIU STRB-L.

SIGNAL: RD TAG WD-L

FULL SIGNAL NAME: Read Tag Word

SIGNAL TYPE: Control signal (active low)

DESCRIPTION: RD TAG WD-L enables the tag word associated with the current message from the Tag Word Buffer Module's Tag Buffer (C.1.2.3.2) onto T0 - T15 for transfer to the RTU, while concurrently enabling the buffers and latching the word into the Old Tag Buffer. RD TAG WD-L is generated in response to a valid transmit vector word command.

SIGNAL: RS IN MSG-L

FULL SIGNAL NAME: Reset In Message Buffer

SIGNAL TYPE: Control signal (active low)

DESCRIPTION: Control signal which resets the In Storage Module's In FIFO Buffer (C.1.2.2). LD IN MSG-L is the inverse of INIT.

SIGNAL: RS OLD MSG-L

FULL SIGNAL NAME: Reset Old Message Buffer

SIGNAL TYPE: Control signal (active low)

DESCRIPTION: RS OLD MSG-L resets the Out Message Buffer Module's Old Message Buffer (C.1.2.3.1.1) in response to INIT or when the combination of SUB A, T and NOT CMD STRB-L is detected in anticipation of Out Message Buffer unloading and the associated Old Message Buffer loading.

SIGNAL: RS OUT MSG-L

FULL SIGNAL NAME: Reset Out Message Buffer

SIGNAL TYPE: Control signal (active low)

DESCRIPTION: RS OUT MSG-L resets the Out Message Buffer Module's Out Message Buffer (C.1.2.3.1.1) in response to INIT or the AP CL MSG command sent by the AP prior to loading a new message.

SIGNAL: RTU CMDS

FULL SIGNAL NAME: RTU Commands

SIGNAL TYPE: Control signal bus

DESCRIPTION: The collection of control signal lines driven by the Command Handler Module (C.1.2.1). The command word subaddress field (bits 9 - 5) is continuously decoded to generate the RTU CMD signal. The appropriate signal line is set when its unique code is detected. The signal acts as an enable for RTU Support Module functions since it is not reset until the next valid command word is received.

COMPONENTS: MODE
 SUB A
 SUB B
 SUB C
 SUB D

SIGNAL: RTU-SS CONTROL

FULL SIGNAL NAME: RTU to Subsystem Control

SIGNAL TYPE: Control signal bus

DESCRIPTION: RTU-SS CONTROL is the set of external subsystem control signals generated by the RTU'S Protocol Sequencer.

COMPONENTS: 1 MHZ CLOCK
 CMD STRB-L
 DATA STRB-L
 EN VEC WD-L
 VAL CMD WD RC-L

SIGNAL: RX/TX CHANNEL A

FULL SIGNAL NAME: Receive/Transmit Channel A

SIGNAL TYPE: Serial data bus

DESCRIPTION: RX/TX CHANNEL A is the set of serial receive and transmit lines which transfer serial data between the RTU's Channel A MT32008 Encoder/Decoder and BUS-63137 Transceiver (C.2). See Reference 29 for a description of the bus components.

COMPONENTS: RX DATA OUT (Channel A)
RX DATA OUT-L (Channel A)
TX DATA IN (Channel A)
TX DATA IN-L (Channel A)
TX INHIBIT (Channel A)

SIGNAL: RX/TX CHANNEL B

FULL SIGNAL NAME: Receive/Transmit Channel B

SIGNAL TYPE: Serial data bus

DESCRIPTION: RX/TX CHANNEL B is the set of serial receive and transmit lines which transfer serial data between the RTU's Channel B MT32008 Encoder/Decoder and BUS-63137 Transceiver (C.2). See Reference 29 for a description of the bus components.

COMPONENTS: RX DATA OUT (Channel B)
RX DATA OUT-L (Channel B)
TX DATA IN (Channel B)
TX DATA IN-L (Channel B)
TX INHIBIT (Channel B)

SIGNAL: SO-L CONTROL

FULL SIGNAL NAME: Strobe Out Control

SIGNAL TYPE: Control signal bus

DESCRIPTION: SO-L CONTROL is the set of control signals generated by the RTU's Protocol Sequencer (C.2) which, together with EXT SO-L, are used by the High Speed Transfer Circuit Module (C.2.2) to generate the SO-L signal.

COMPONENTS: DATA STRB-L
SO-L (Protocol Sequencer Output)
VAL CMD WD RC-L

SIGNAL: SO-L

FULL SIGNAL NAME: Strobe Out

SIGNAL TYPE: SO-L CONTROL, Control signal (active low)

DESCRIPTION: SO-L controls the gating of data words from the RTU's FIFO (C.2) onto T0 - T15. When the High Speed Transfer Circuit (C.2.2) to allow continuous reception of 32-word data messages, SO-L is modified, with the resultant SO-L signal being 2.0 MHz. To conform to the nomenclature in the RTU chip set's documentation, SO-L is used to designate both the input and output signal.

SIGNAL: S/R-L

FULL SIGNAL NAME: RTU Service Request

SIGNAL TYPE: SS-RTU CONTROL (active low)

DESCRIPTION: S/R-L is a direct input into the RTU's Protocol Sequencer (C.2) which allows external setting of the RTU status word service request bit.

SIGNAL: SS FLAG

FULL SIGNAL NAME: Subsystem Flag

SIGNAL TYPE: Status signal

DESCRIPTION: The inverted form of SS FLAG-L which is used by the AP to determine SS FLAG-L service status.

SIGNAL: SS FLAG-L

FULL SIGNAL NAME: RTU Subsystem Flag

SIGNAL TYPE: SS-RTU CONTROL (active low)

DESCRIPTION: SS FLAG-L is a direct input into the RTU's Protocol Sequencer (C.2) which allows external setting of the RTU status word subsystem flag bit.

SIGNAL: SSIU STRB-L

FULL SIGNAL NAME: Subsystem Interface Unit Strobe

SIGNAL TYPE: SS-RTU CONTROL (active low)

DESCRIPTION: The 1.0 MHz transmit strobe signal generated by the Strobe Handler Module (C.1.2.5) which controls the subsystem's gating of the individual data words (one per strobe) onto T0 - T15 and the RTU's subsequent latching of the words from the data highway.

SIGNAL: SSIU STRB-L + EXT SO-L

FULL SIGNAL NAME: SSIU STRB-L .OR. EXT SO-L

SIGNAL TYPE: Strobe (active low)

DESCRIPTION: This signal is either SSIU STRB-L or EXT SO-L, depending upon the type of sequence initiated (transmit or receive). Tri-state buffers control the gating of the appropriate signal onto and off of this signal line.

SIGNAL: SS-RTU CONTROL

FULL SIGNAL NAME: Subsystem to RTU Control

SIGNAL TYPE: Control signal bus

DESCRIPTION: SS-RTU CONTROL is the set of external RTU control signals generated by the RTU Support Module.

COMPONENTS:	EXT SO-L	S/R-L
	SS FLAG-L	SSIU STRB-L
	WRAP EN	

SIGNAL: SUB A

FULL SIGNAL NAME: Subaddress A

SIGNAL TYPE: RTU CMD

DESCRIPTION: RTU CMD which enables normal message word transmission (from the Out Message Buffer Module's Out Message Buffer C.1.2.3.1) or reception (by the In Message Storage Module's In FIFO Buffer, C.1.2.2) when triggered by SSIU STRB-L or EXT SO-L, respectively. The actual subaddress field value assigned to SUB A is application dependent.

SIGNAL: SUB B

FULL SIGNAL NAME: Subaddress B

SIGNAL TYPE: RTU CMD

DESCRIPTION: RTU CMD which enables transmission of the old tag word (OLD TAG WD) from the Tag Word Buffer Module's Old Tag Buffer (C.1.2.3.2) when triggered by SSIU STRB-L. The actual subaddress field value assigned to SUB B is application dependent.

SIGNAL: SUB C

FULL SIGNAL NAME: Subaddress C

SIGNAL TYPE: RTU CMD

DESCRIPTION: RTU CMD which enables transmission of a data word from the Out Message Buffer Module's Old Message Buffer (C.1.2.3.1) when triggered by SSIU STRB-L. The actual subaddress field value assigned to SUB C is application dependent.

SIGNAL: SUB D

FULL SIGNAL NAME: Subaddress D

SIGNAL TYPE: RTU CMD

DESCRIPTION: RTU CMD which enables the contents of the Error Handler Module's AP ERR Register and SS ERR Register (C.1.2.4) onto T0 - T7 and T8 - T15, respectively, when triggered by SSIU STRB-L. The actual subaddress field value assigned to SUB D is application dependent.

SIGNAL: T

FULL SIGNAL NAME: Transmit

SIGNAL TYPE: Control signal

DESCRIPTION: T is derived from Bit 10 (T/R) of the Command Handler Module's Command Buffer (C.1.2.1). When high, T enables generation of the transmit strobes (SSIU STRB-Ls), as well as a number of other RTU Support Module Functions.

SIGNAL: T0 - T15

FULL SIGNAL NAME: T0 - T15 Data Highway

SIGNAL TYPE: 16-Bit data bus

DESCRIPTION: The RTU's internal 16-bit data bus which links the dual redundant MT32008 Encoder/Decoders, MT32004 Protocol Sequencer and MT32003 FIFO and interfaces the RTU Module (C.2) to the RTU Support Module (C.1.2).

SIGNAL: TAG WD

FULL SIGNAL NAME: Tag Word

SIGNAL TYPE: 16-Bit word

DESCRIPTION: Generically, a tag word is a message identifier assigned to an incoming message by the BC and updated by the AP prior to message transmission. TAG WD is the tag word associated with the current message to be transmitted which is stored in the Out Message Buffer Module's Out Message Buffer (C.1.2.3.1.1).

SIGNAL: TAG WD CNT

FULL SIGNAL NAME: Tag Word Count

SIGNAL TYPE: Word count

DESCRIPTION: See XMIT WD CNT.

SIGNAL: UNL IN MSG-L

FULL SIGNAL NAME: Unload In Message Buffer

SIGNAL TYPE: Control signal (active low)

DESCRIPTION: Conditioned signal generated by In Message Storage Module's Unload Timer (C.1.2.2) from AP RD MSG and 16 MHZ CLOCK which provides the buffer enables and In FIFO Buffer unloading controls required by the FIFO selected for the implementation.

SIGNAL: VAL CMD WD RC-L

FULL SIGNAL NAME: Valid Command Word Received

SIGNAL TYPE: SO-L CONTROL, RTU-SS CONTROL (active low)

DESCRIPTION: VAL CMD WD RC-L is an internal RTU signal which controls the latching of the command word by the Protocol Sequencer (C.2) during the first 500 nanosecond period in which it is available on T0 - T15. The subsystem uses this signal to latch the command word into the Command Handler Module's Command Buffer. This signal is used to allow the subsystem access to the command word during mode commands, in which CMD STRB-L is not generated.

SIGNAL: VAL CNT

FULL SIGNAL NAME: Valid Transmit Command Count

SIGNAL TYPE: Control signal

DESCRIPTION: The inverted form of CNT ERR, which is used by the Strobe Handler Module (C.1.2.5) to enable the generation of the transmit strobes (SSIU STRB-Ls).

SIGNAL: WRAP AROUND CMD

FULL SIGNAL NAME: Wrap Around Command

SIGNAL TYPE: Control signal

DESCRIPTION: Control signal generated by the Command Handler Module's Subaddress Decoder (C.1.2.1) upon detection of the Wrap Around Command subaddress (11110). This is an alternative to the AP SET WRAP/AP CL WRAP implementation which enables execution of the wrap around function upon receipt of the first wrap around command by immediately setting WRAP EN.

SIGNAL: WRAP EN

FULL SIGNAL NAME: RTU Wrap Enable

SIGNAL TYPE: SS-RTU CONTROL

DESCRIPTION: WRAP EN is the input to the RTU Protocol Sequencer (C.2) which allows execution of the special RTU wrap around test function in which the BC sends consecutive receive and transmit commands to the wrap around subaddress. The data words are stored, then retransmitted directly from the RTU's FIFO without subsystem intervention.

SIGNAL: XACK-L

FULL SIGNAL NAME: Multibus Memory Transfer Acknowledge

SIGNAL TYPE: Control signal (active low)

DESCRIPTION: XACK-L is used to terminate a memory-mapped input/output operation. Following a read command (MRDC-L), XACK-L informs the AP that stable data is available on the data bus (DATA BUS-L); following a write command (MWTC-L) it indicates 1553BBI acceptance of the data word.

SIGNAL: XACK TRIGGER

FULL SIGNAL NAME: Transfer Acknowledge Trigger

SIGNAL TYPE: Control signal

DESCRIPTION: Signal generated by the Multibus Decoder Module's Address/Control Decoder (C.1.1.1) which initiates an XACK Timer Module (C.1.1.4) transfer acknowledge cycle. Signal is equivalent to ((MRDC .OR. MWTC) .AND. BS).

SIGNAL: XMIT ERR-L

FULL SIGNAL NAME: Transmit Error Registers

SIGNAL TYPE: Control signal (active low)

DESCRIPTION: XMIT ERR-L, generated by the Error Handler Module (C.1.2.4) from SUB D and SSIU STRB-L, which enables the contents of the Error Handler Module's AP ERR Register and SS ERR Register (C.1.2.4) onto T0 - T7 and T8 - T15, respectively, for transmission to the BC.

SIGNAL: XMIT GATE CTRL-L

FULL SIGNAL NAME: Transmit Gate Control

SIGNAL TYPE: Control signal (active low)

DESCRIPTION: See RCV GATE CONTROL-L.

SIGNAL: XMIT MSG-L

FULL SIGNAL NAME: Transmit Message

SIGNAL TYPE: Control signal (active low)

DESCRIPTION: XMIT MSG-L, generated by the Out Message Buffer from SUB A and SSIU STRB-L, controls both the unloading of the Out Message Buffer for transmission and the concurrent loading of the data into the Old Message Buffer. XMIT MSG-L enables the FIFO buffers and controls their loading and unloading as required by the selected FIFOs. XMIT MSG-L also resets the latch which controls S/R-L, since even if a transmission error subsequently occurs, the transfer of the Out Message Buffer contents to the Old Message Buffer will be driven to completion by the Strobe Handler Module.

SIGNAL: XMIT SEL-L

FULL SIGNAL NAME: Transmit Select

SIGNAL TYPE: Control signal (active low)

DESCRIPTION: XMIT SEL-L is the transmit command enable signal generated from T, NOT MODE and VAL CNT. This signal controls the gating of the 1 MHZ CLOCK into the Strobe Handler Module (C.1.2.5) as well as the gating of the resulting SSIU STRB-L from the module to the RTU and the Out Message Storage Module.

SIGNAL: XMIT WD CNT

FULL SIGNAL NAME: Transmit Word Count

SIGNAL TYPE: Word count

DESCRIPTION: The 5-bit word count (bits 4 - 0) derived from the Tag Word Buffer Module's Tag Buffer (TAG WD CNT) or Old Tag Buffer (OLD TAG WD CNT) (C.1.2.3.2) for normal message transmissions (SUB A) or old message transmissions (SUB C), respectively. XMIT WD CNT is compared to CMD WD CNT as a rough verification of the BC's process management operation, generating the VAL CNT and CNT ERR signals.

APPENDIX E

HARDWARE TIMING DIAGRAMS

The timing diagrams associated with the hardware design are included in this appendix. The first diagram, Figure E.1, shows the timing constraints which must be met when implementing the design of the XACK Timer Module's (Figure C.1.1.4) Timer/Delay Circuit. The remaining eight figures, which are concerned with message reception, message transmission and vector word transmission timing, are included to demonstrate that the RTU Support Module design satisfies the RTU timing constraints. For each of these operations, the general timing considerations and constraints (required timing) imposed upon the subsystem by the selected RTU chip set are shown. The design timing diagrams show the estimated timing for each operation. A set of general notes concerning the design timing precedes the set of diagrams. These notes also include a cross-reference to indicate the modules from which the timing information is derived.

General Design Timing Notes

1. The majority of the 1553B protocol timing requirements are handled internally by the RTU chip set. These include time out checking, status word transmission, BIT word management and all mode command responses except for vector word transmission. To prevent conflicts in usage of its internal T0 - T15 Data Highway, the RTU imposes very strict timing constraints upon the subsystem for the operations performed across the subsystem interface: message reception, message transmission and vector word transmission. As discussed in Chapter 6, Section 6.2.3.2, the time critical subsystem message handling functions are handled by the RTU Support Module hardware to allow these constraints to be met.

A considerable effort was made during the course of the RTU Support Module design to minimize the effects of the timing constraints upon the subsystem. A large input storage buffer removes the necessity for extremely rapid transfers of data from the 1553BBI and the AP. By pre-loading the tag (vector) word register and outgoing message buffer, and initiating data transmission via the service request mechanism, the transfers from the AP to the 1553BBI become less time-critical. Thus, the only really critical timing problems are limited to those at the subsystem interface (i.e., the RTU Support Module). The problem of decoding the command word in time to generate the function enable signals (RTU CMDS, T, R), which is driven by the vector word transmission timing, was solved by latching the command word during a

period intended for internal RTU use (VAL CMD WD RC-L). Then, after circumventing the timing problems associated with setting the status bit values (S/R-L and SS FLAG-L) by including a set of jointly controlled latches (via AP and RTU CMDS), the remaining timing problems could be summarized as follows:

- a. Generating transmit strobes (SSIU STRB-L) and receive strobes (EXT SO-L) which meet the timing and pulse characteristic requirements shown in Figures E.3 and E.6, respectively.

- b. Determining the maximum delay characteristics which could be tolerated in the FIFO buffers and support circuitry used to latch message words from or gate the message words and tag word onto the Data Highway and ensuring that components with such characteristics are available, based on current technology.

The Strobe Handler Module design (C.1.2.5) produces strobes which meet the required timing constraints. A survey of available technology showed that components which meet the timing requirements for latching/enabling the data words, as shown in Figures E.3, E.6 and E.9, are available. As discussed in Appendix C, the design requires that the discrete logic components must use "ALS" or "AS" technology, and the FIFO buffers must be RAM-based instead of shift register type. The required characteristics of the FIFOs are discussed in detail in Appendix C, General Design Note 8. The design timing calculations stated in the design notes for Figures E.4, E.7 and E.9 which are used to demonstrate that the design meets the timing requirements are based upon typical chip delays for the discrete logic (gates and latches) and

required maximum latency for the FIFOs. These delays, obtained from Reference 33, are as follows:

- a. Inverting buffer: 3.5 nanoseconds (ns), 74ALS04
- b. Non-inverting buffer: 8.0 ns, 74ALS34
- c. Two-input positive NAND gate: 3.5 ns, 74ALS00
- d. Two-input positive AND gate: 6.5 ns, 74ALS08
- e. Three-input positive NAND gate: 7 ns, 74ALS10
- f. Non-inverted tri-state buffer: 8 ns, 74LS125A
- h. Inverting tri-state buffer: 5.5 ns, 74ALS368
- i. D-type latch: 10 ns set-up, 4 ns hold, 74ALS577
- j. FIFO buffer: Maximum latency approximately 45 - 50 ns,

depending upon type of logic used in Out Message Buffer Module (C.1.2.3.1). The total of the gate delays and the FIFO latency cannot exceed 75 ns; this is the tightest constraint imposed on the FIFOs.

2. The module designs from which the design timing diagrams were derived are as follows:

- a. Design Transmit Timing: Transmit timing calculations (Figure E.4) are derived from SSIU STRB-L generation timing (Figure C.1.2.5, Strobe Handler Module) and from the delays in enabling the message words onto the T0 - T15 Data Highway following the receipt of each SSIU STRB-L. Data words are transmitted from three sources, as determined by the transmit command subaddress. Current and old data message transmissions are handled by the Out Message Buffer Module (Figures C.1.2.3.1 and C.1.2.3.1.1), while transmission of the old tag word and error register contents are handled by the Tag Word Buffer Module (Figure C.1.2.3.2) and the Error Handler Module (Figure C.1.2.4),

respectively. The SSIU STRB-L generation timing is the same in all three cases. The data enable delay timing is based upon the worst case transmit timing, which because of the higher FIFO buffer access times, occurs during data message transmissions.

b. Design Receive Timing: Receive timing calculations (Figure E.7) are derived from EXT SO-L generation timing (Figure C.1.2.5, Strobe Handler Module) and from the delays in latching the message words by the In Message Storage Module (Figure C.1.2.2).

c. Design Transmit Vector Word Timing: Transmit vector word timing calculations (Figure E.9) are derived from the Tag Word Buffer Module (Figure C.1.2.3.2) delays in enabling the vector word (tag word) onto the T0 - T15 Data Highway.

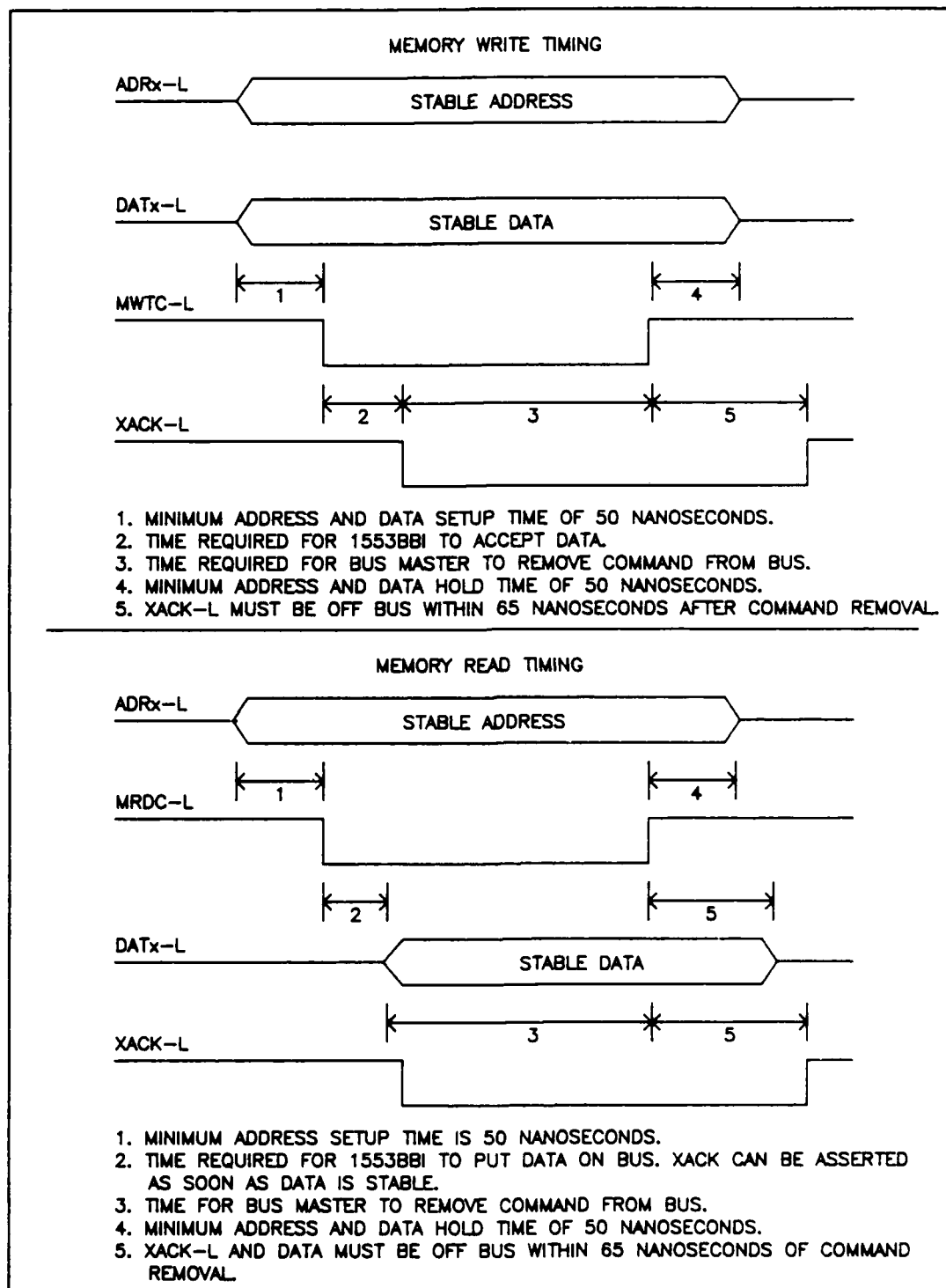


Figure E.1. XACK Timing Diagram

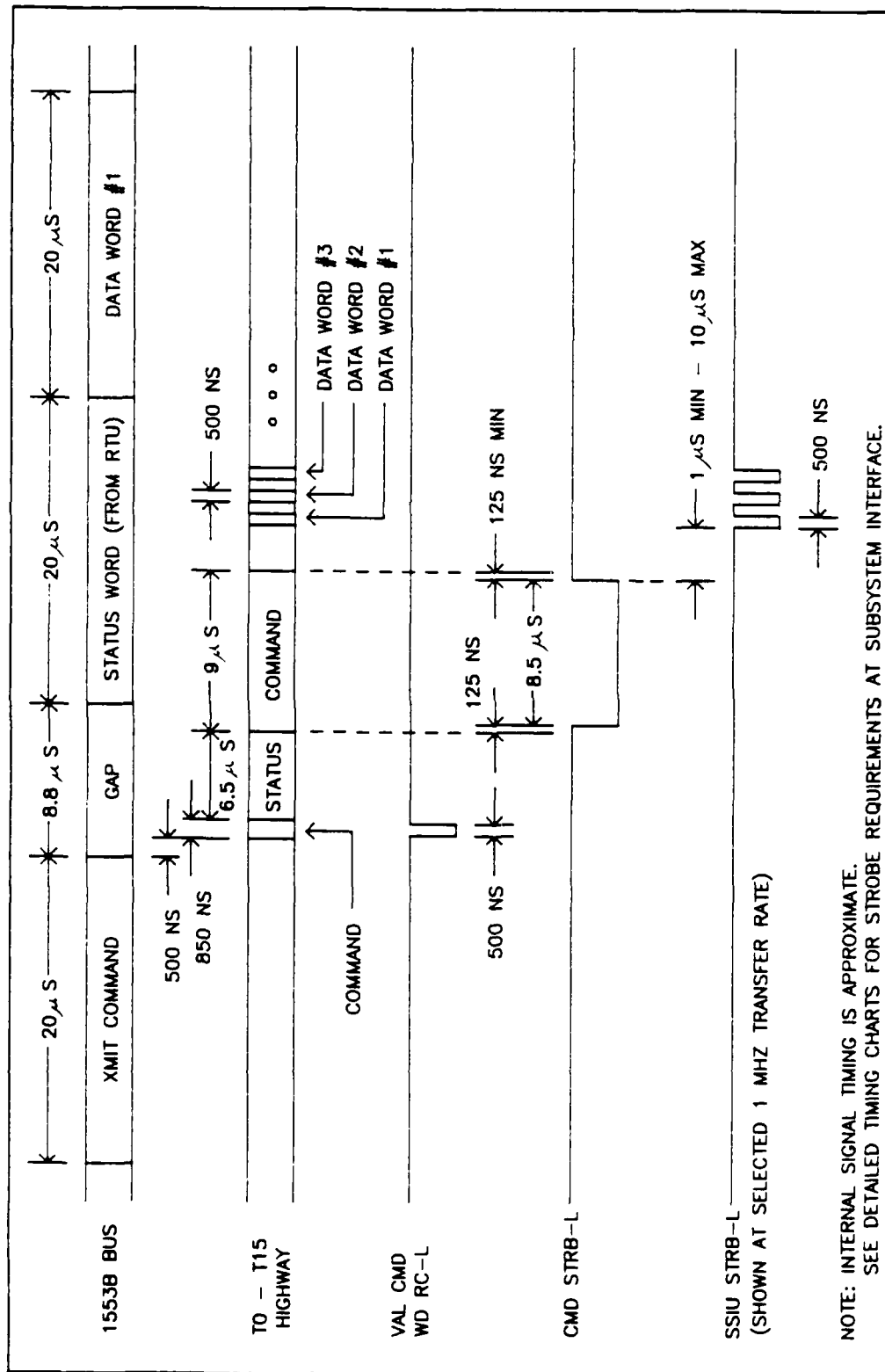


Figure E.2. General Transmit Command Timing Considerations (Figure 22, Chapter 6)

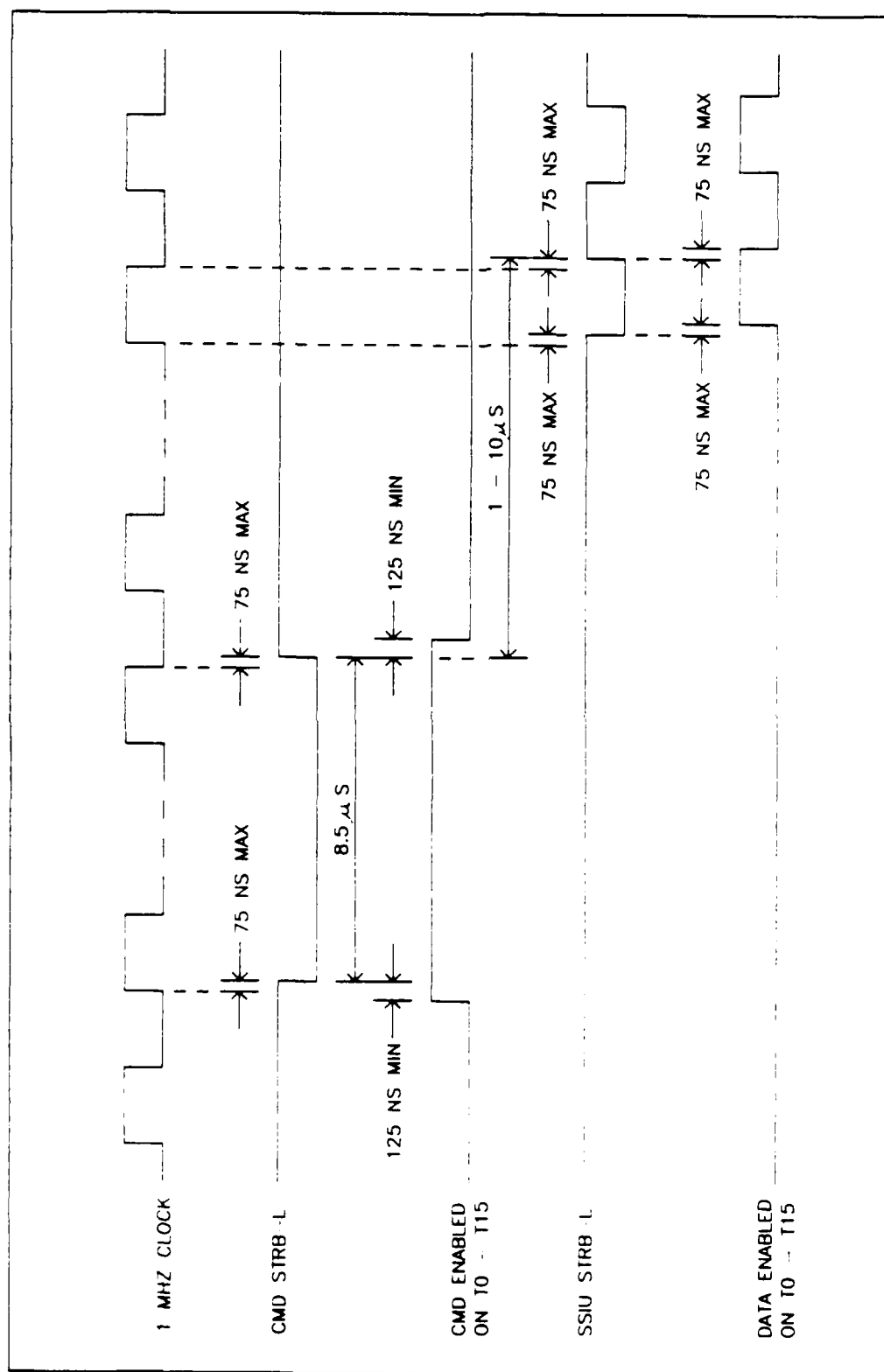


Figure E.3. Required Transmit Timing

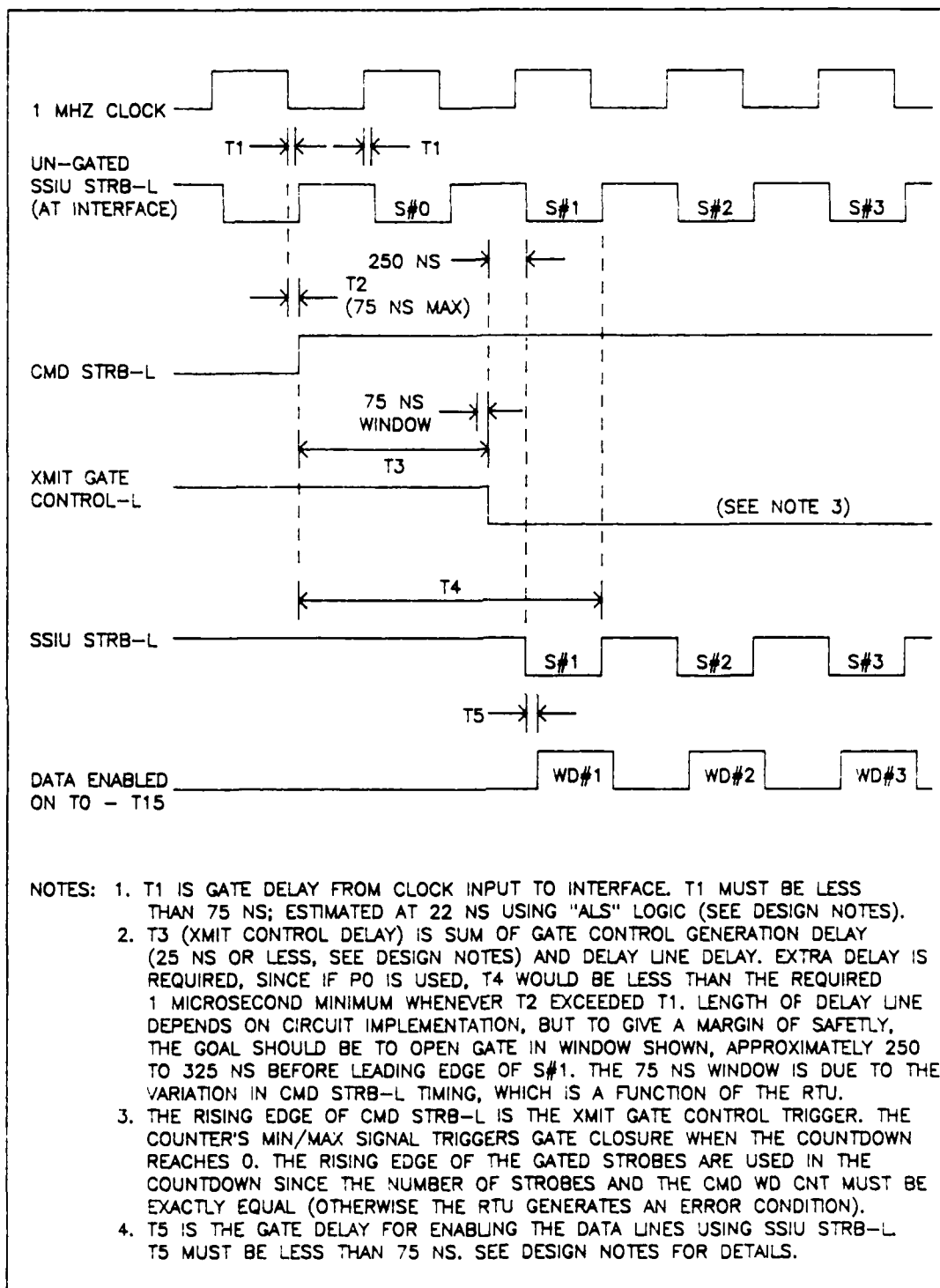


Figure E.4. Design Transmit Timing

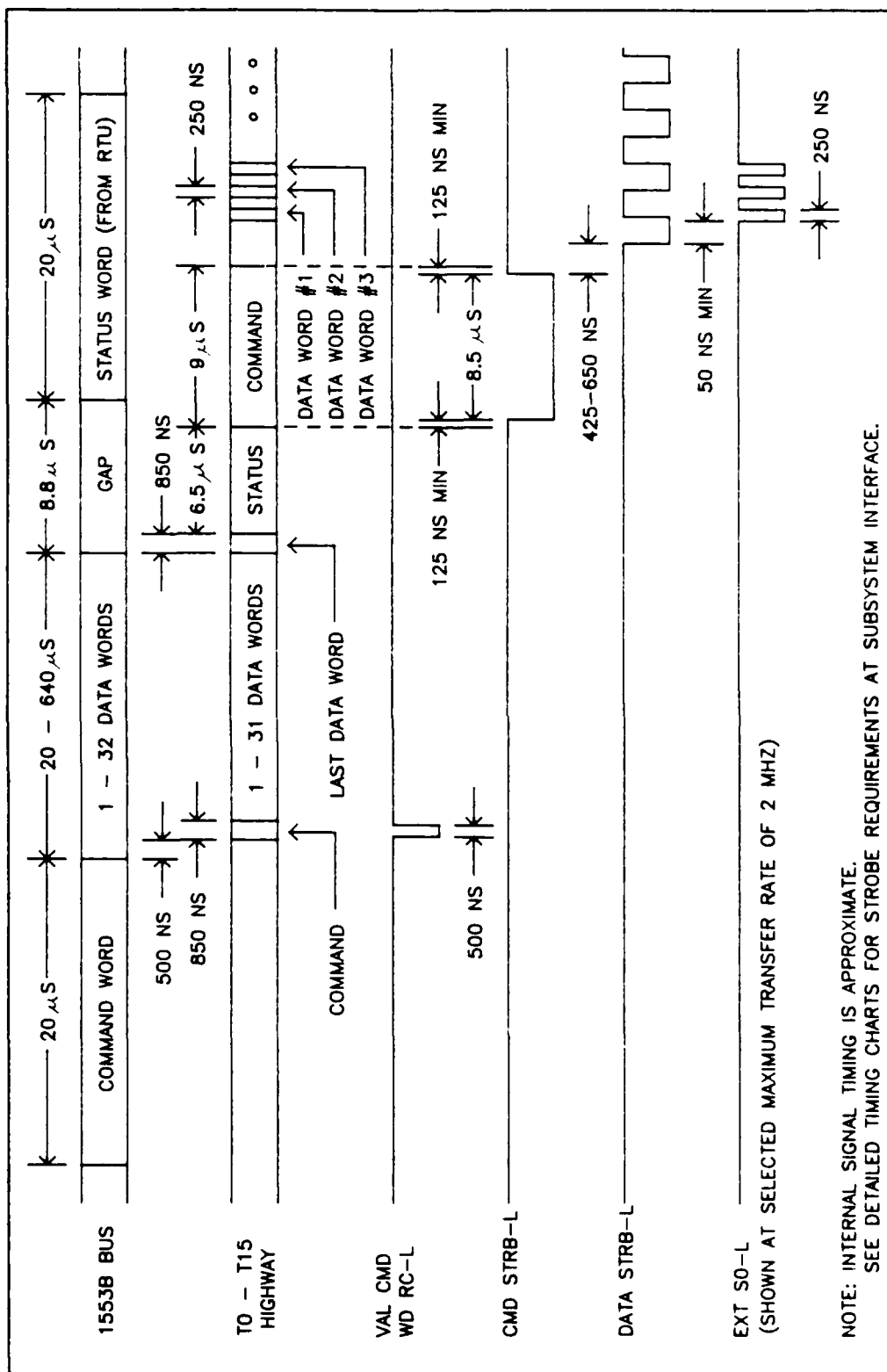


Figure E.5. General Receive Command Timing Considerations (Figure 23, Chapter 6)

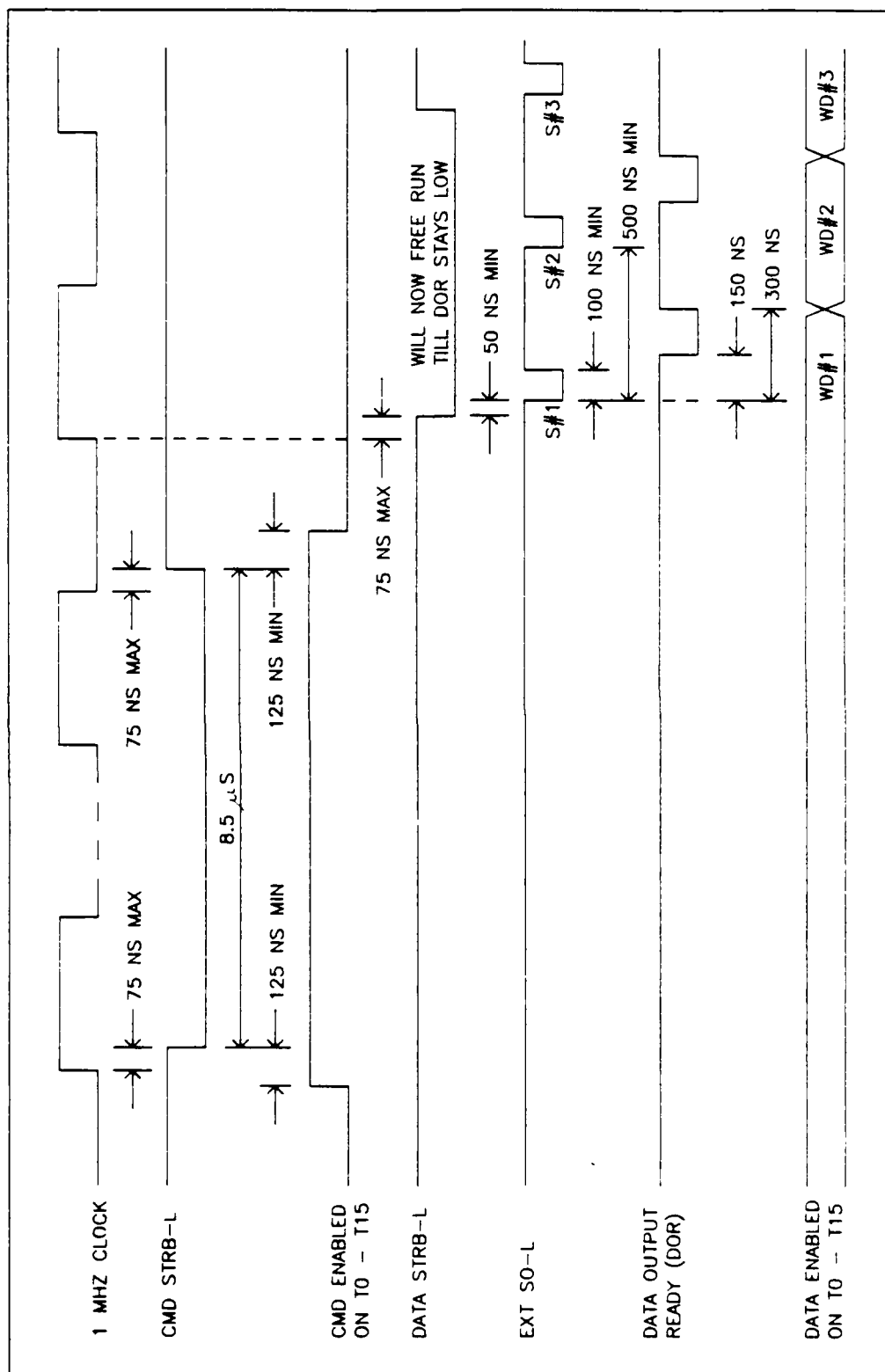


Figure E.6. Required Receive Timing

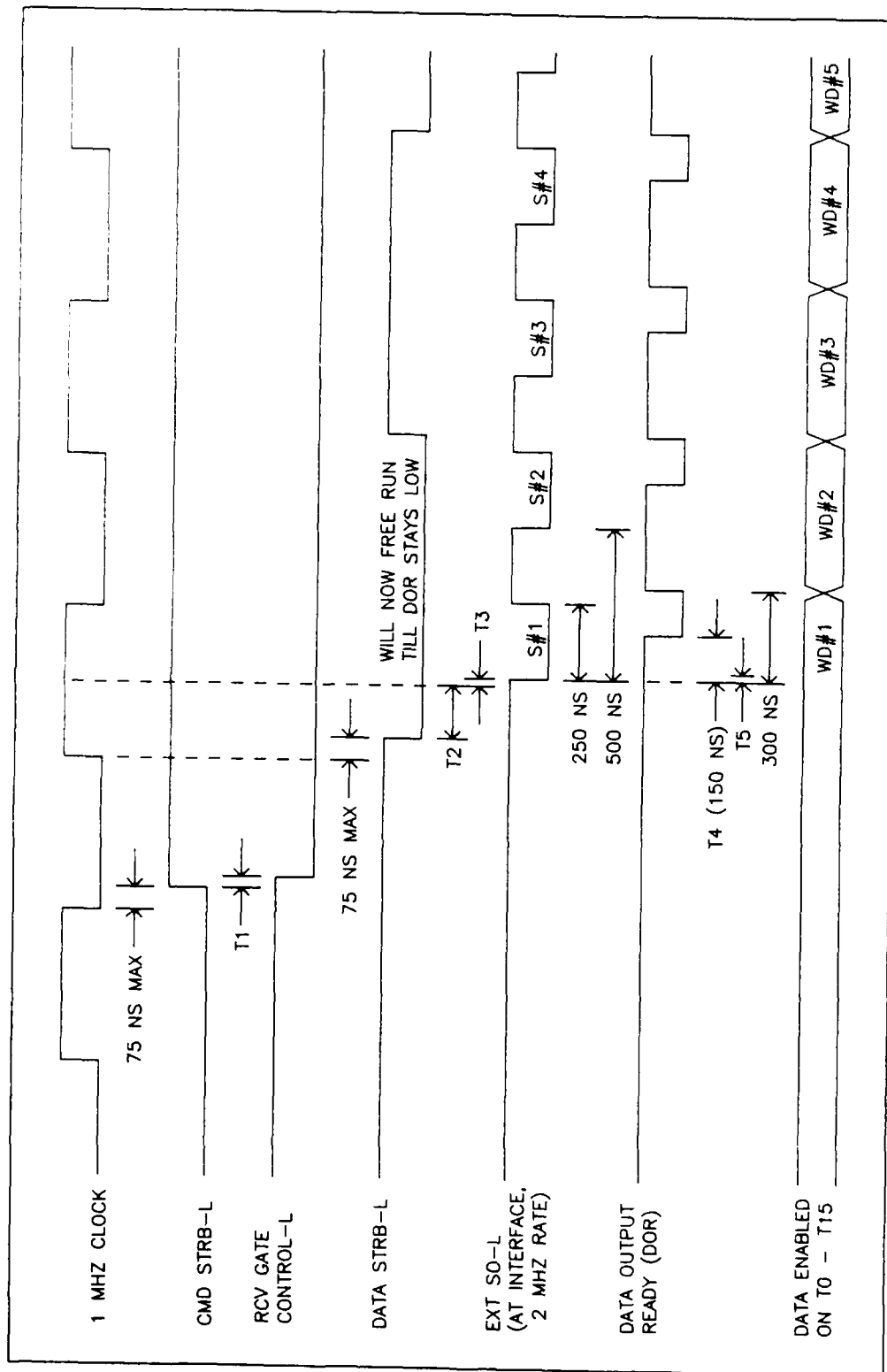


Figure E.7. Design Receive Timing

DESIGN RECEIVE TIMING NOTES

- NOTES:
1. T1 (RCV CONTROL DELAY) IS EQUAL TO THE GATE CONTROL GENERATION DELAY (25 NS OR LESS, SEE DESIGN NOTES). THE TRAILING EDGE OF CMD STRB-L DIRECTLY CONTROLS THE GATE; SINCE EXT SO IS TRIGGERED BY DATA STRB-L, NO PREMATURE PULSES ARE CREATED.
 2. THE COUNTER'S MIN/MAX SIGNAL TRIGGERS GATE CLOSURE WHEN THE COUNTDOWN REACHES 0. THE RISING EDGES OF THE GATED STROBES ARE USED FOR COUNTING SINCE THE NUMBER OF STROBES AND CMD WD CNT MUST BE EXACTLY EQUAL (TO PREVENT AN RTU ERROR).
 3. TO HANDLE SUCCESSIVE BLOCKS OF INCOMING DATA, TRANSFER TIME BETWEEN THE MT32003 AND THE SUBSYSTEM MUST BE MINIMIZED. ALTHOUGH ONLY THE 50 NS MINIMUM FOR T2 IS STATED, MINIMIZING THE DELAY IN UNLOADING THE MT32003 IS ESSENTIAL. IF THE LEADING EDGE OF THE FIRST EXT SO (DERIVED FROM 16 MHZ CLOCK) OCCURS EXACTLY 250 NS INTO THE 1 MHZ CLOCK PULSE (SHOWN AT RIGHT EDGE OF T2), T2 WILL VARY BETWEEN 175 AND 250 NS, DEPENDING ON THE DATA STRB-L DELAY. T3 IS THE GATE DELAY EXPERIENCED BY EXT SO BETWEEN ITS GENERATION AND ITS ARRIVAL AT THE INTERFACE AS EXT SO-L. T3 IS ESTIMATED AT LESS THAN 14 NS WHEN USING ALS LOGIC.
 4. THE DATA OUTPUT READY (DOR) SIGNAL IS HIGH WHEN DATA ON THE TO - T15 HIGHWAY IS STABLE. ALTHOUGH EXT SO-L IS 250 NS, THE SUBSYSTEM IS ONLY ALLOWED 150 NS FROM THE TIME THE LEADING EDGE OF EXT SO-L REACHES THE RTU (T4) TO LATCH THE DATA. THE DATA MUST BE LATCHED BY THE IN FIFO BUFFER (IN MESSAGE STORAGE MODULE) WITHIN THIS TIME PERIOD.
 5. T5 IS THE SUM OF THE EXT SO-L PROPAGATION DELAY (FROM THE STROBE HANDLER INTERFACE TO THE IN FIFO BUFFER) AND THE SETTLING TIME OF THE INPUT BUFFER GATE (FOLLOWING THE ENABLE). T5 IS ESTIMATED AT 15 NS IF ALS LOGIC IS USED. THE DIFFERENCE BETWEEN T4 AND T5 IS THEN THE AMOUNT OF TIME AVAILABLE FOR INITIATING (VIA LD) AND LOADING THE DATA (INCLUDING ANY HOLD TIME FOLLOWING THE LATCHING).

Figure E.7. Design Receive Timing (Cont.)

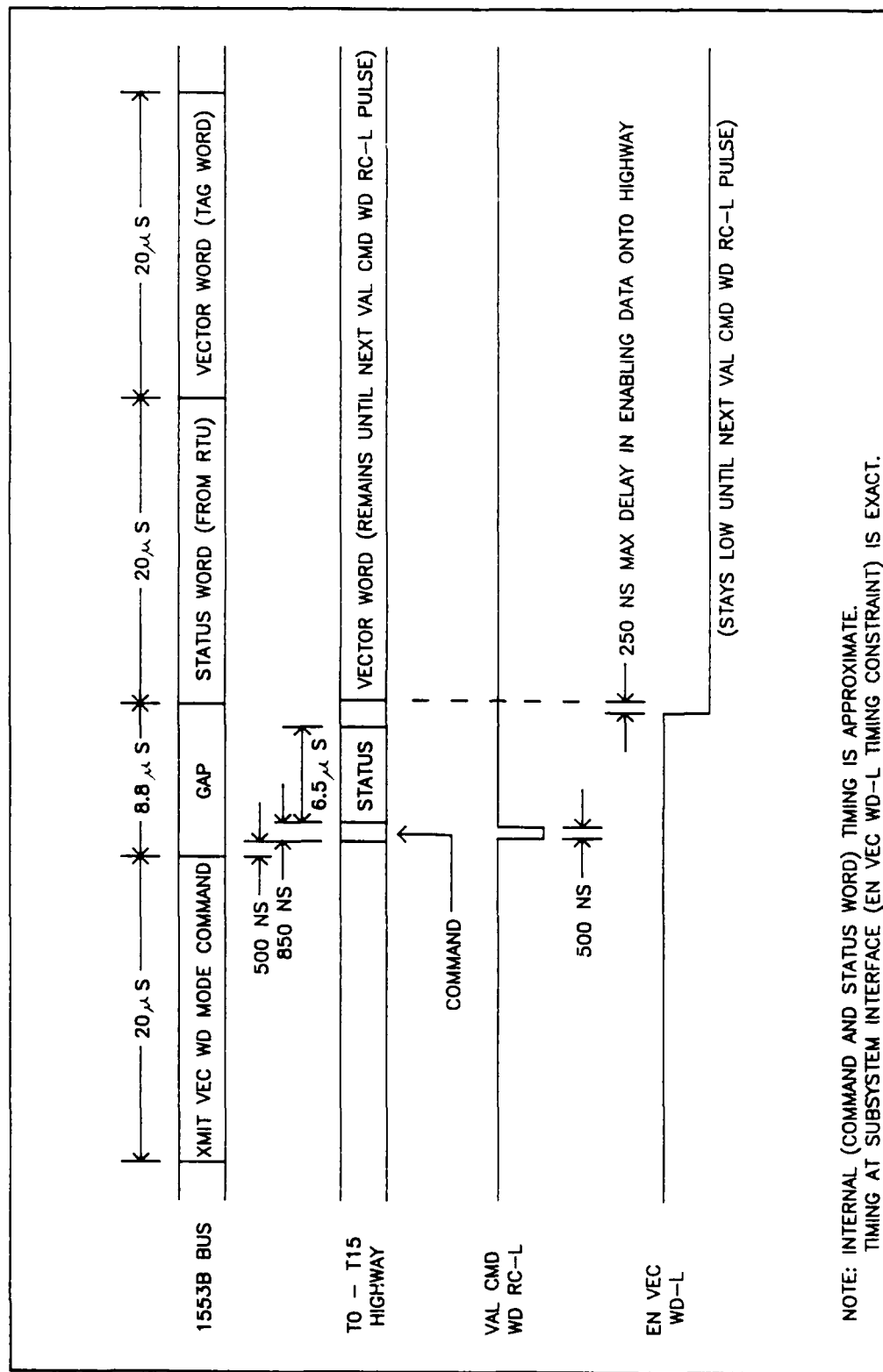


Figure E.8. Transmit Vector Word Timing (Figure 24, Chapter 6)

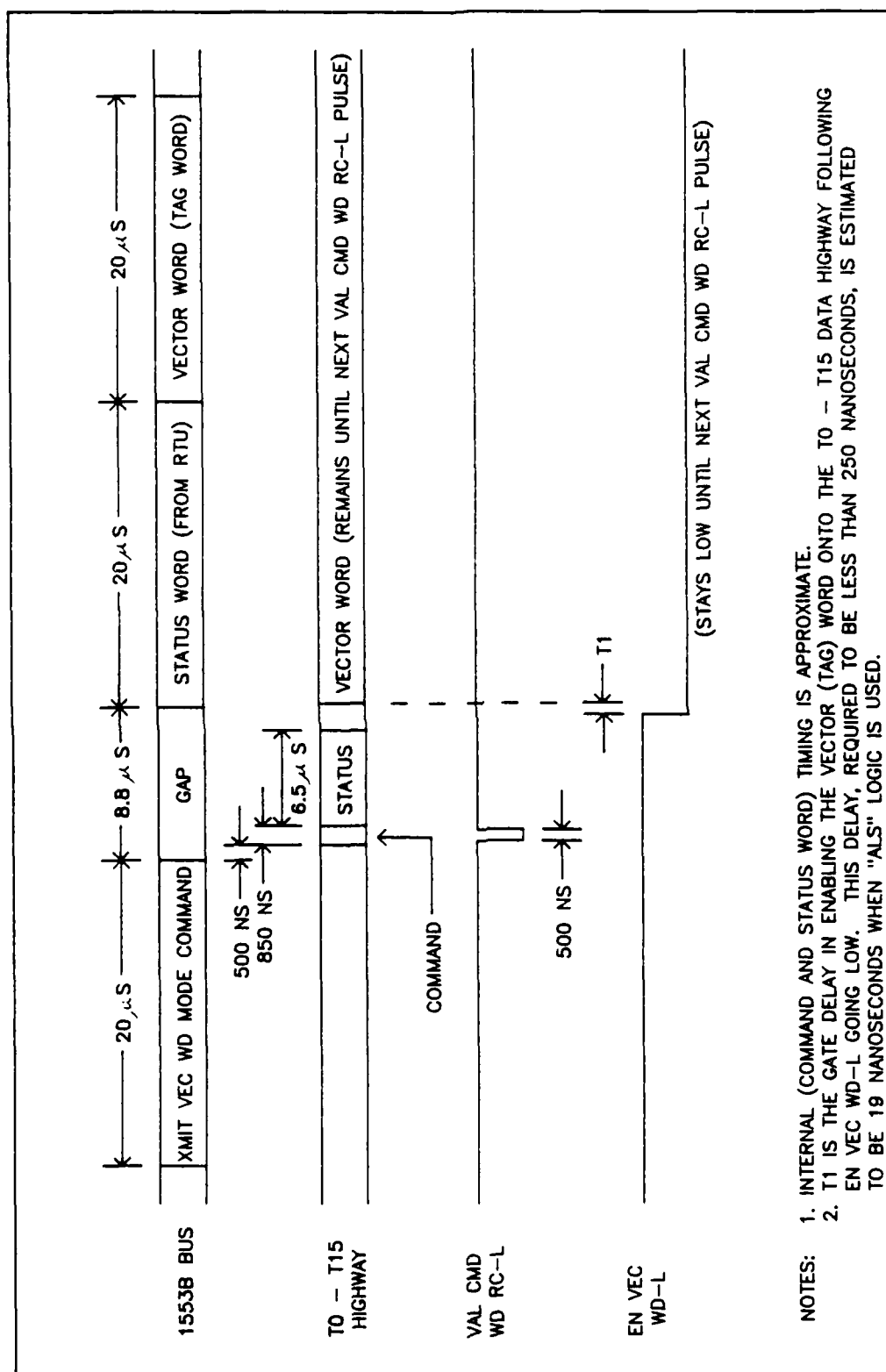


Figure E.9. Design Transmit Vector Word Timing

APPENDIX F

SYSTEM SOFTWARE DATA FLOW DIAGRAMS

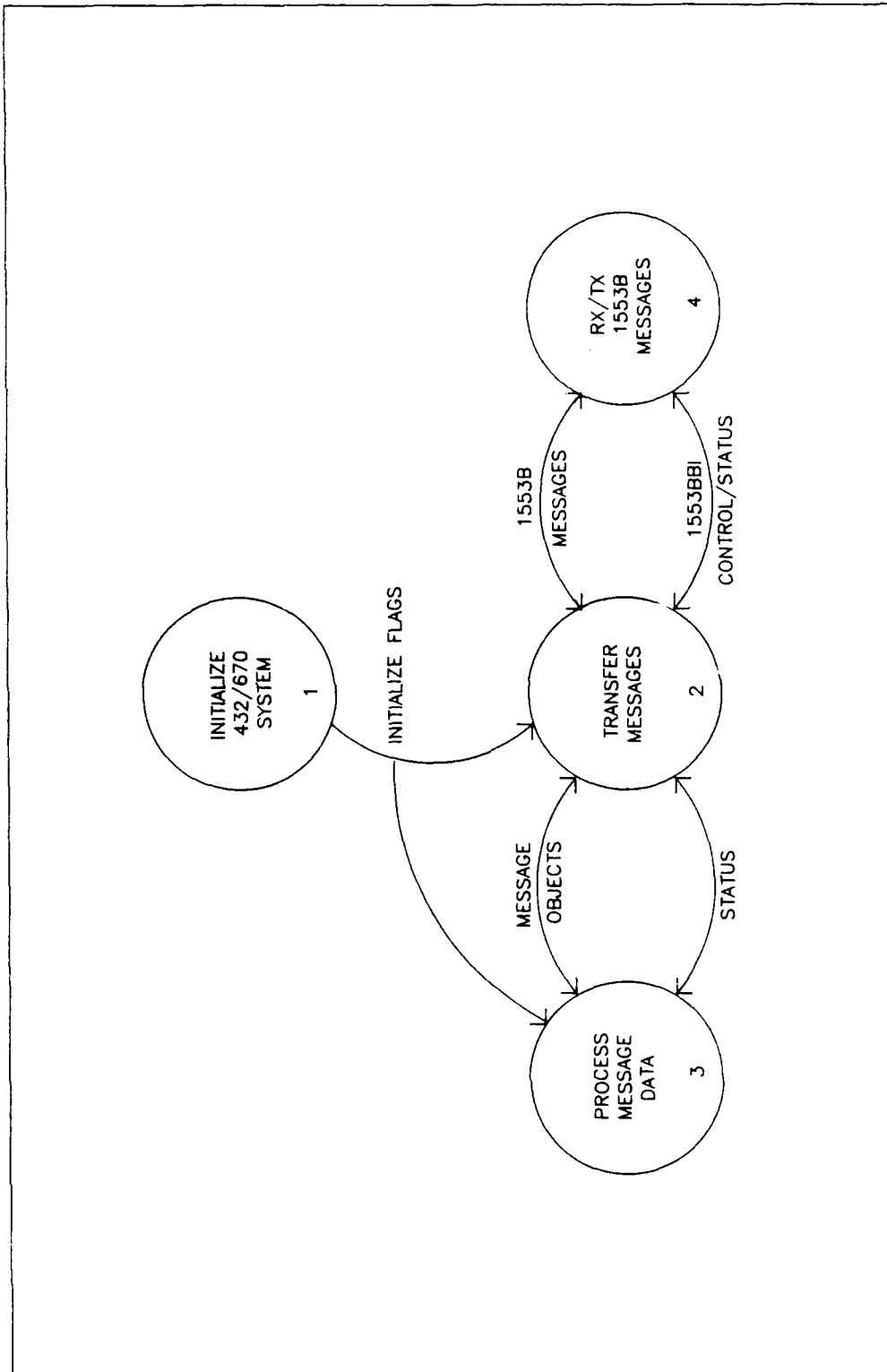


Figure F.0. Top Level Software Data Flow Diagram

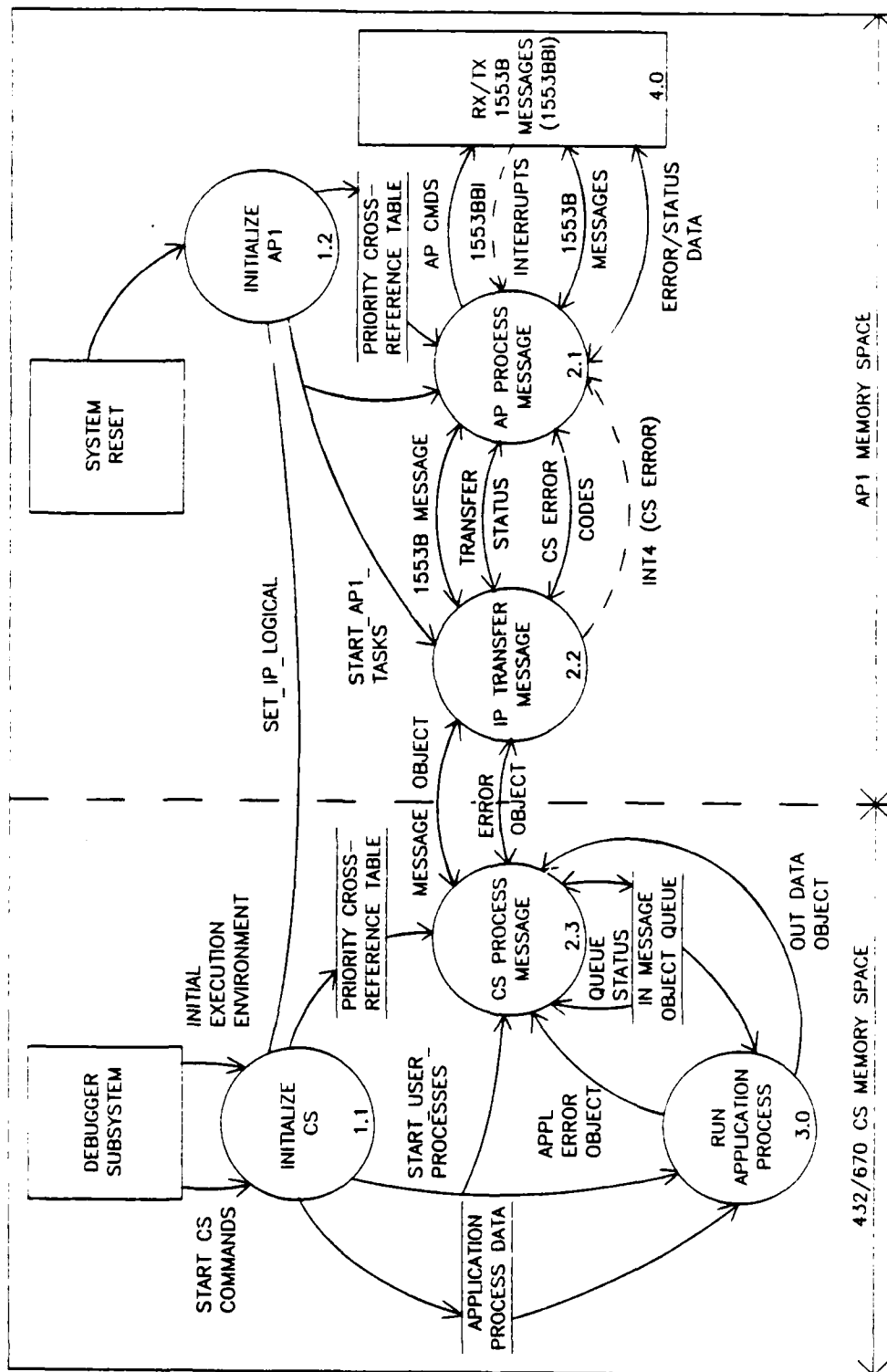


Figure F.1. System Level Software Data Flow Diagram

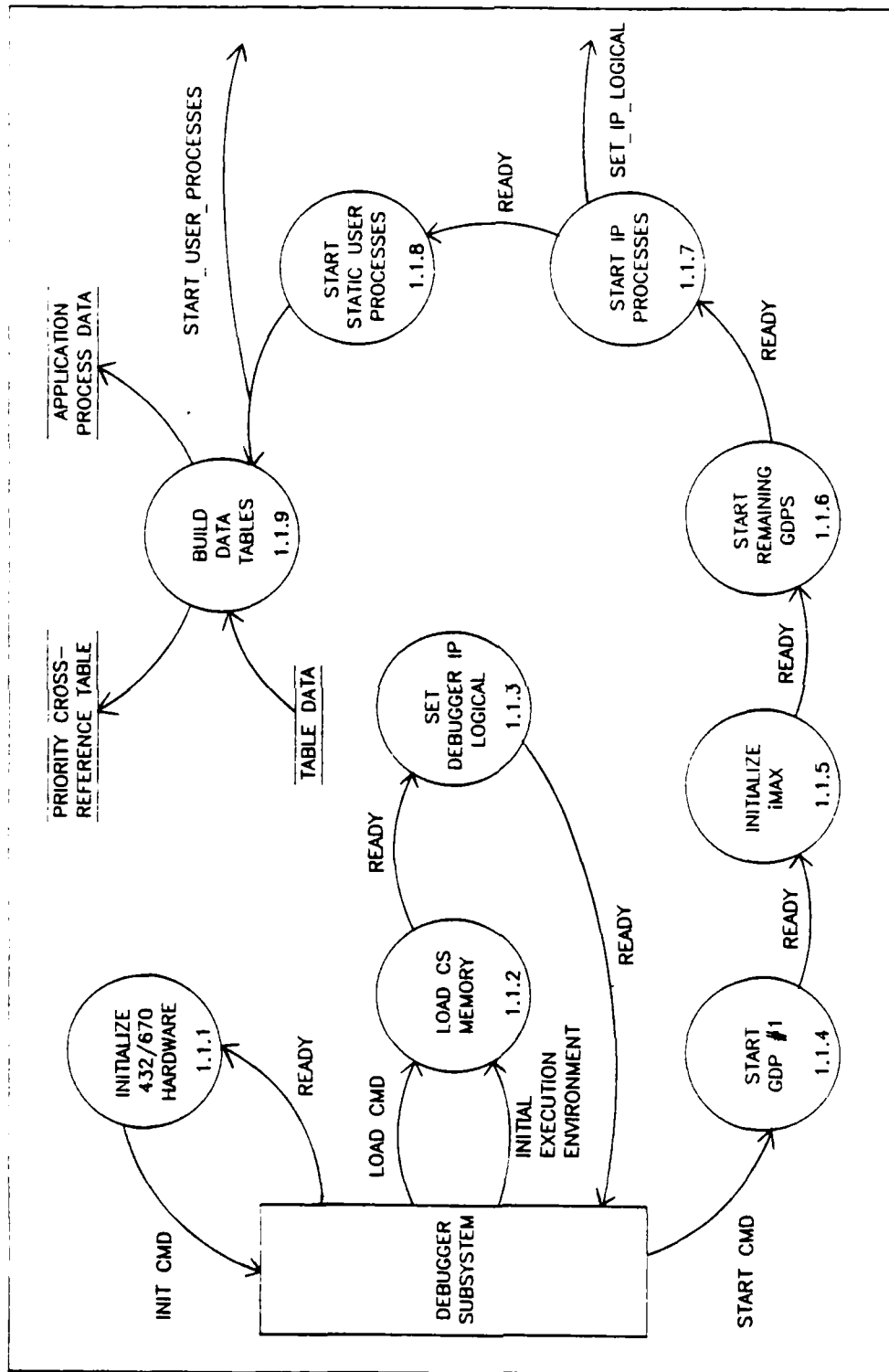


Figure F.1.1. Initialize CS Data Flow Diagram

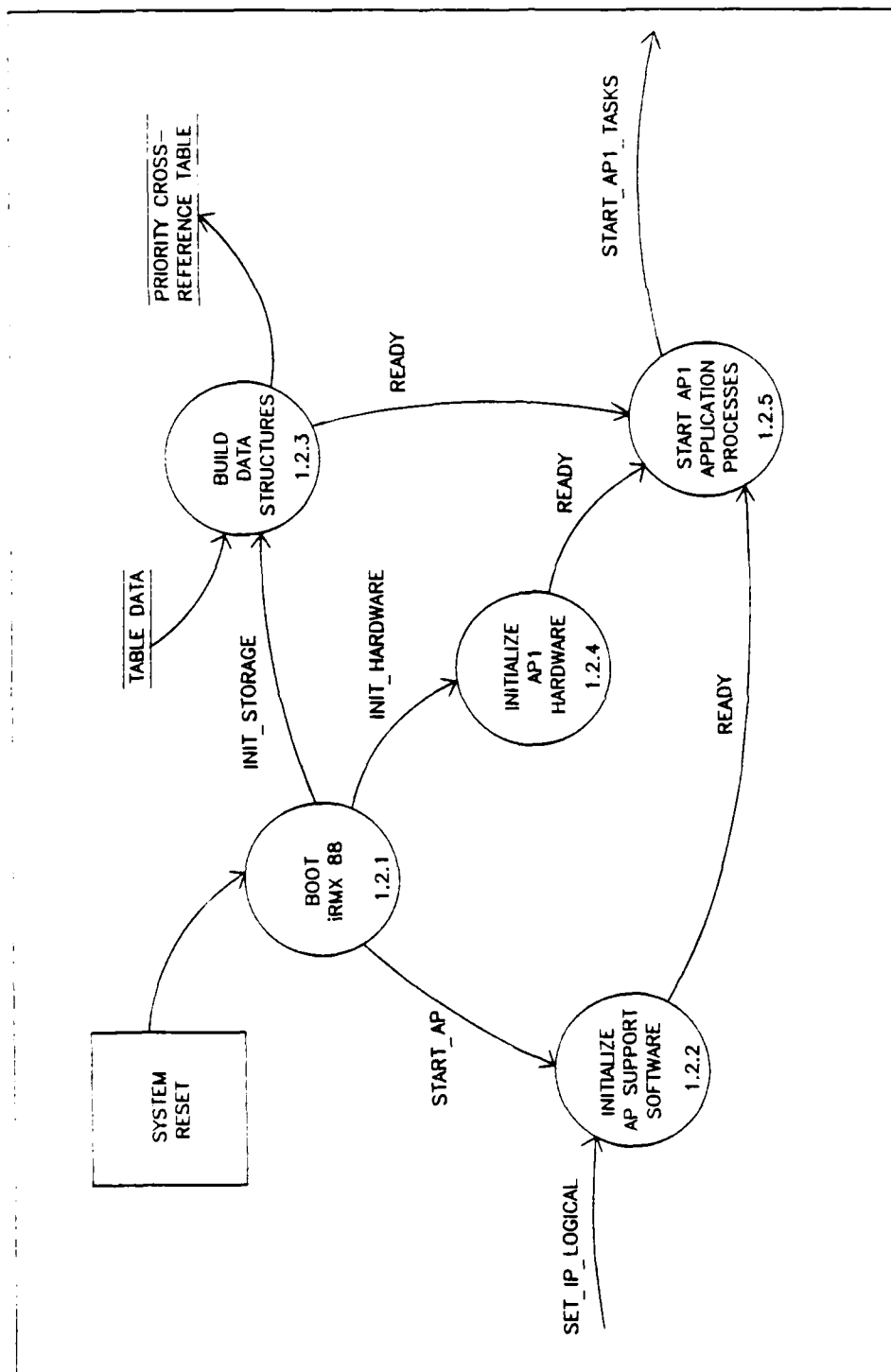


Figure F.1.1.2. Initialize AP1 Data Flow Diagram

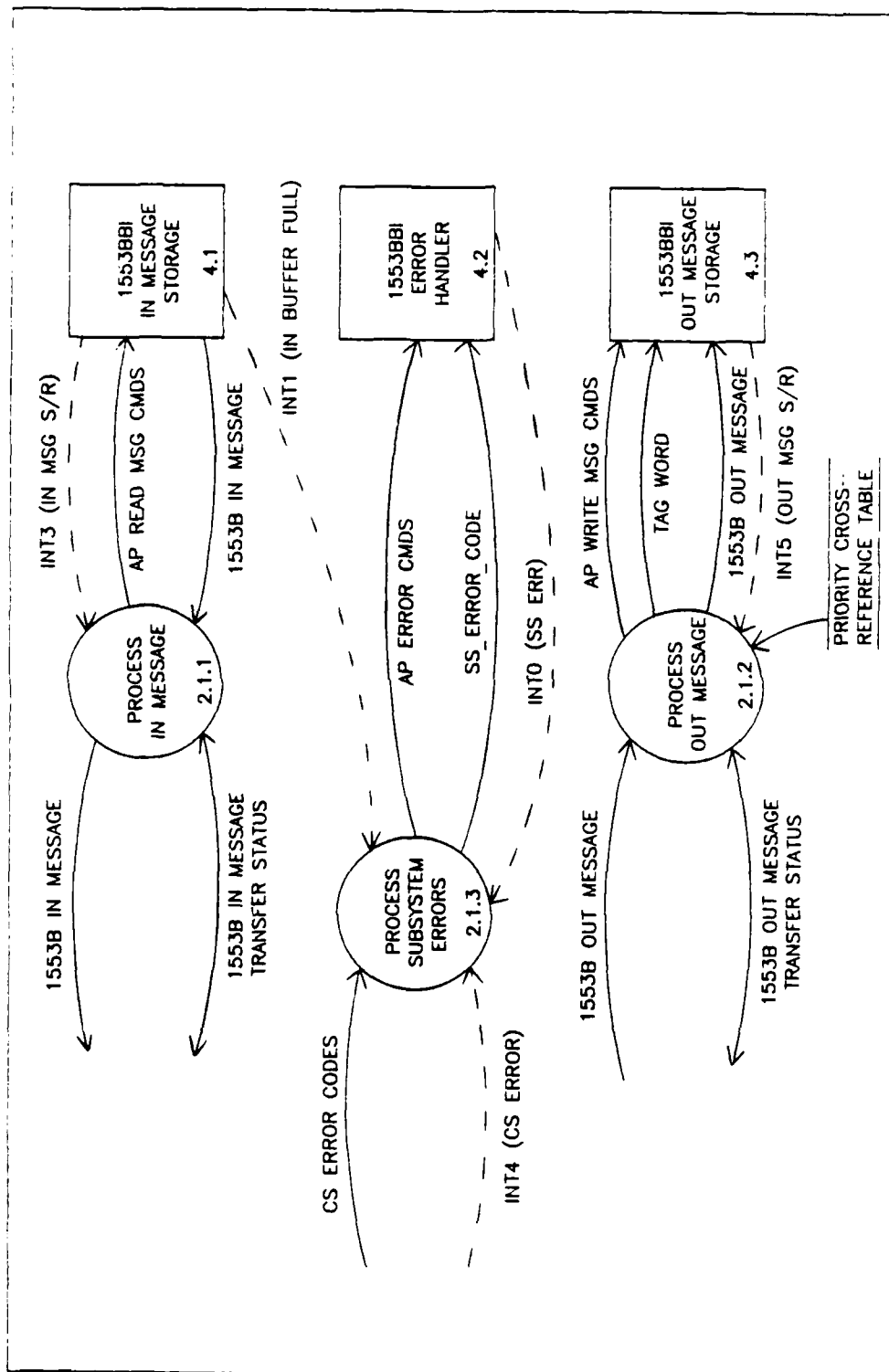


Figure F.2.1. AP Process Message Data Flow Diagram

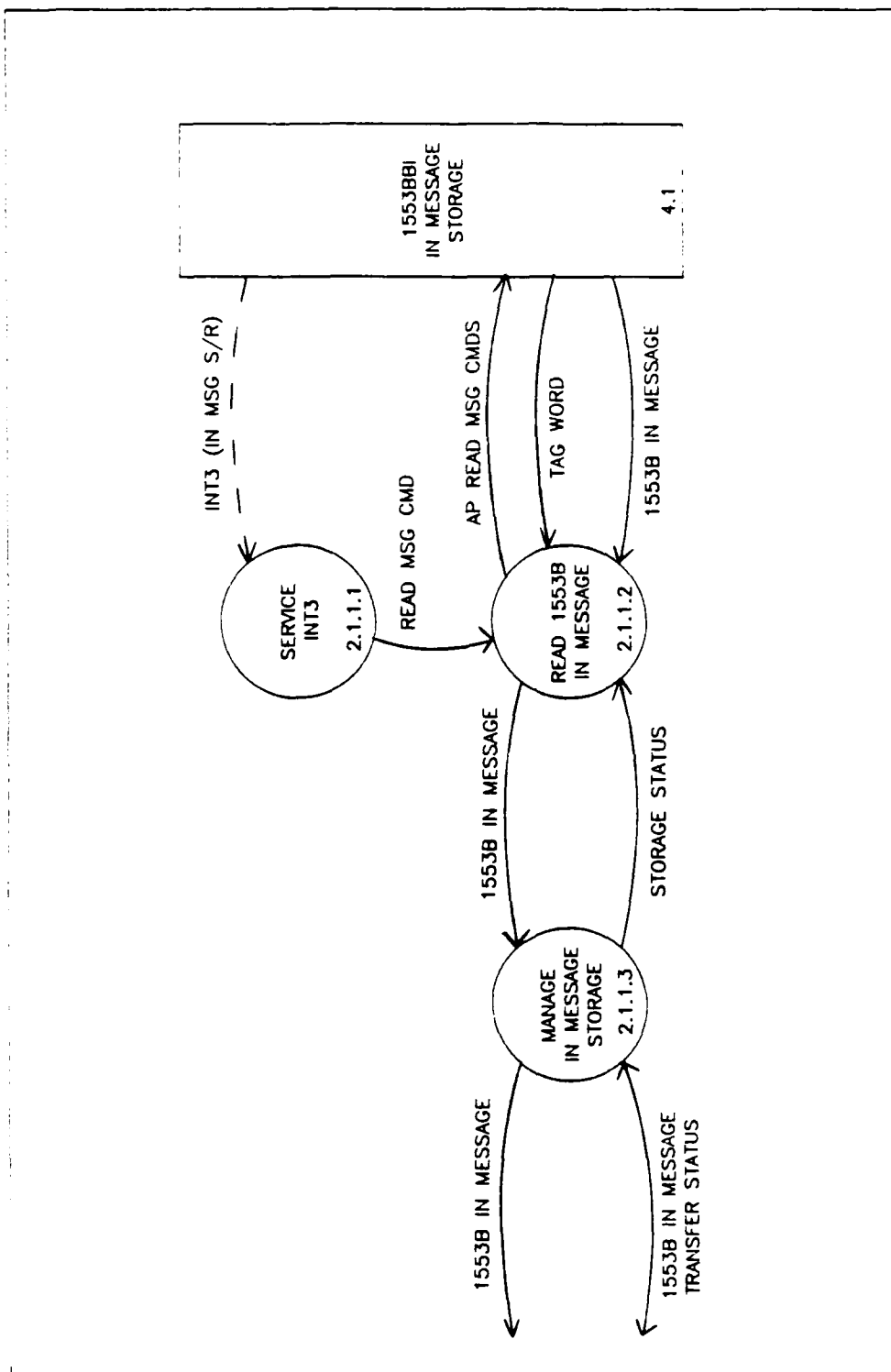


Figure F.2.1.1. Process In Message Data Flow Diagram

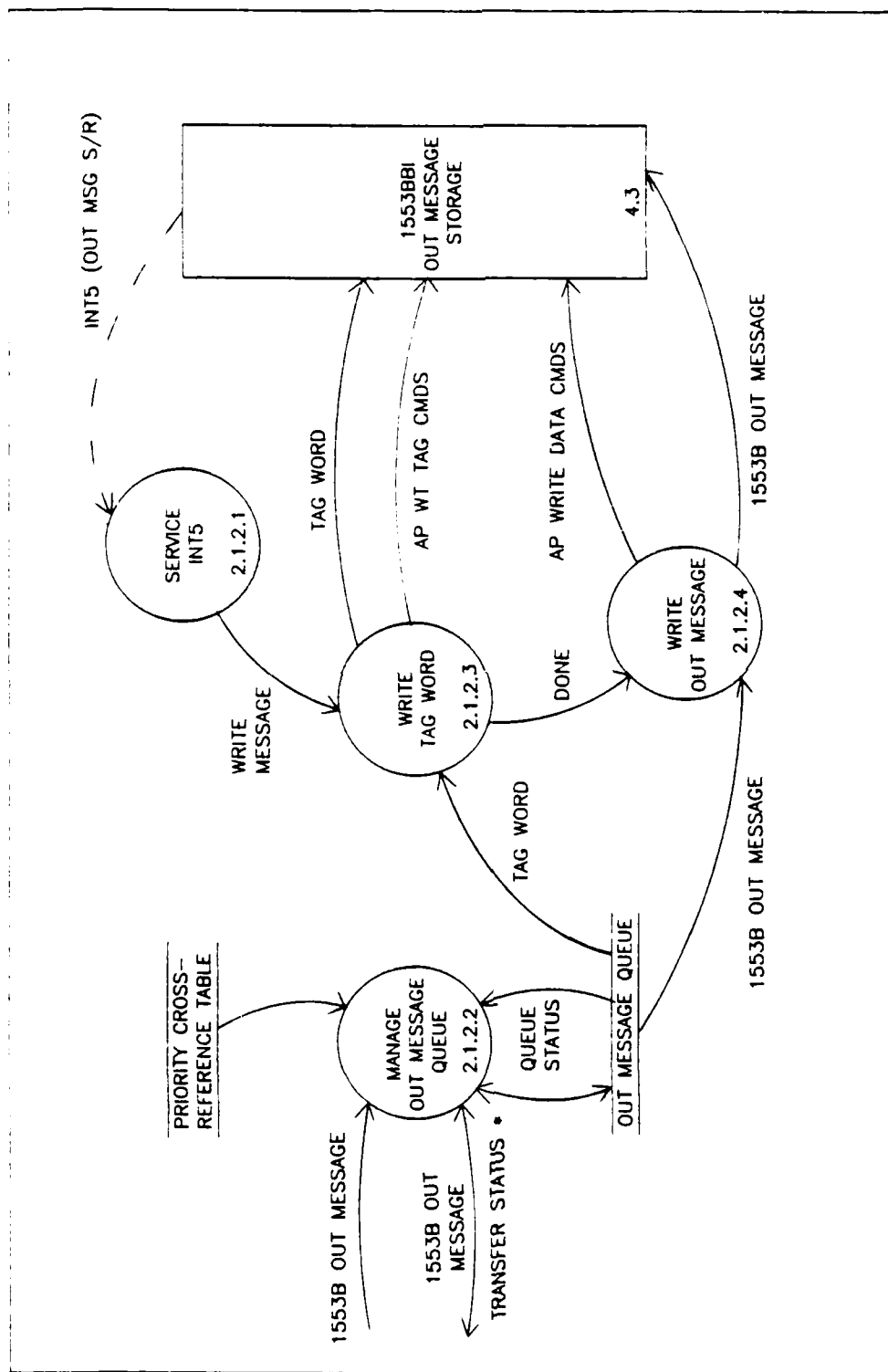


Figure F.2.1.2 Process Out Message Data Flow Diagram

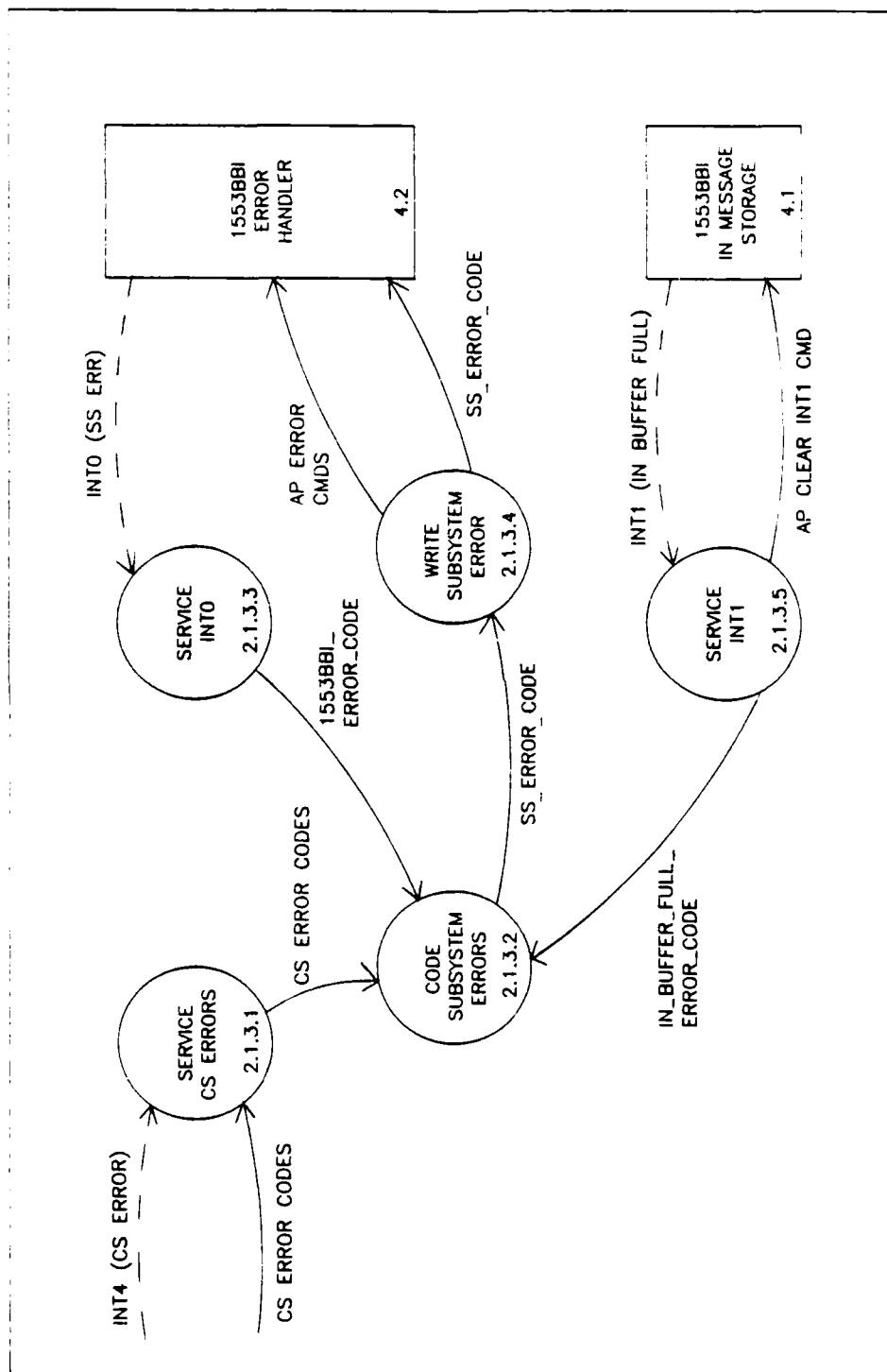
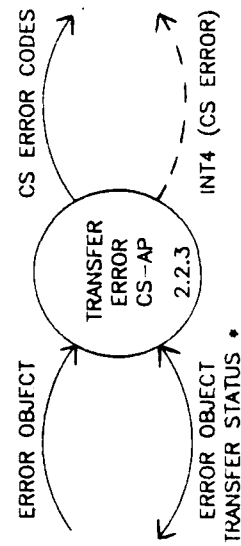
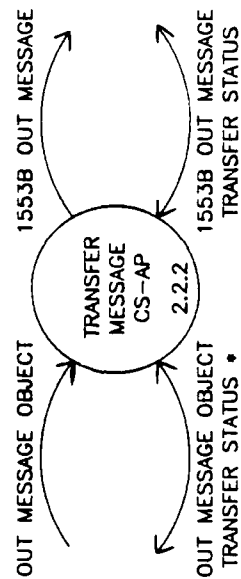
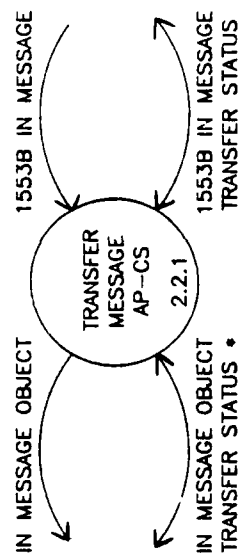


Figure F.2.1.3. Process Subsystem Errors Data Flow Diagram



DATA OBJECT

Figure 2.2.1P Transfer Message Data Flow Diagram

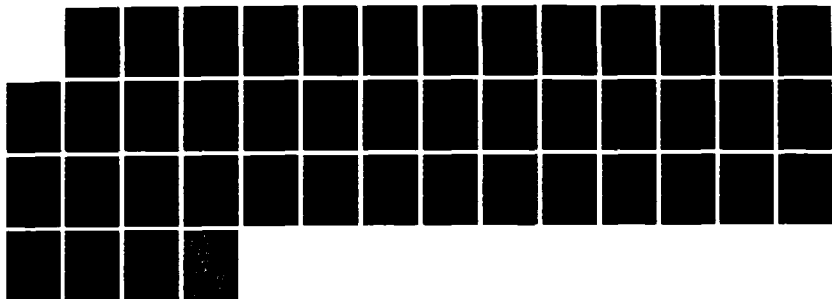
AD-A189 841

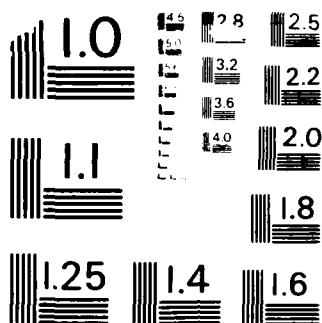
AN INVESTIGATION OF THE INTERFACING OF THE INTEL IAPX
432/670 COMPUTER SY.. (U) AIR FORCE INST OF TECH
WRIGHT-PATTERSON AFB OH SCHOOL OF ENGI.. L H BLACK
SEP 87 AFIT/GE/ENG/875-1 F/G 12/6

4/4

UNCLASSIFIED

NL





U.S. GOVERNMENT PRINTING OFFICE: 1963

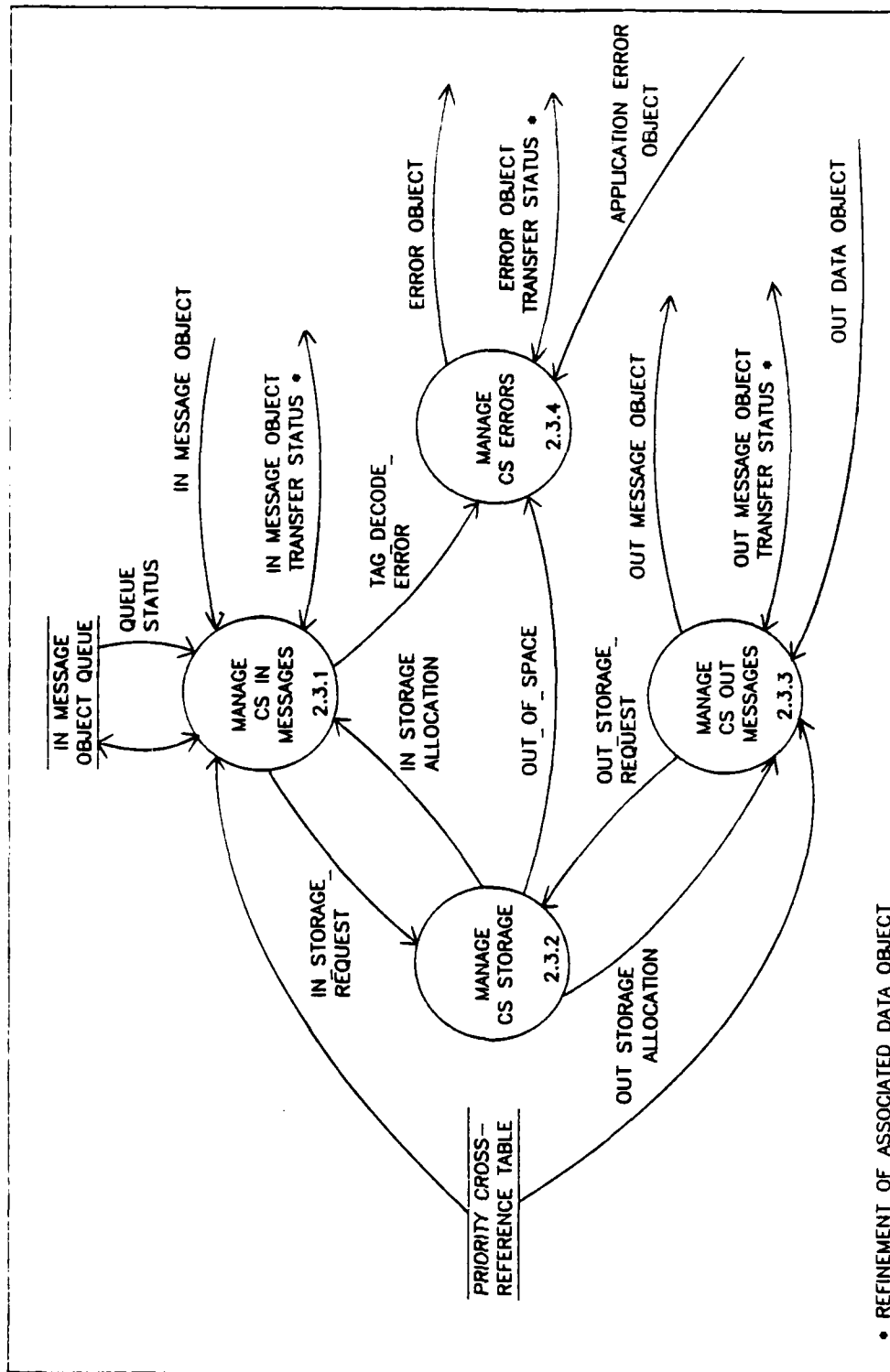


Figure F.2.3. CS Process Message Data Flow Diagram

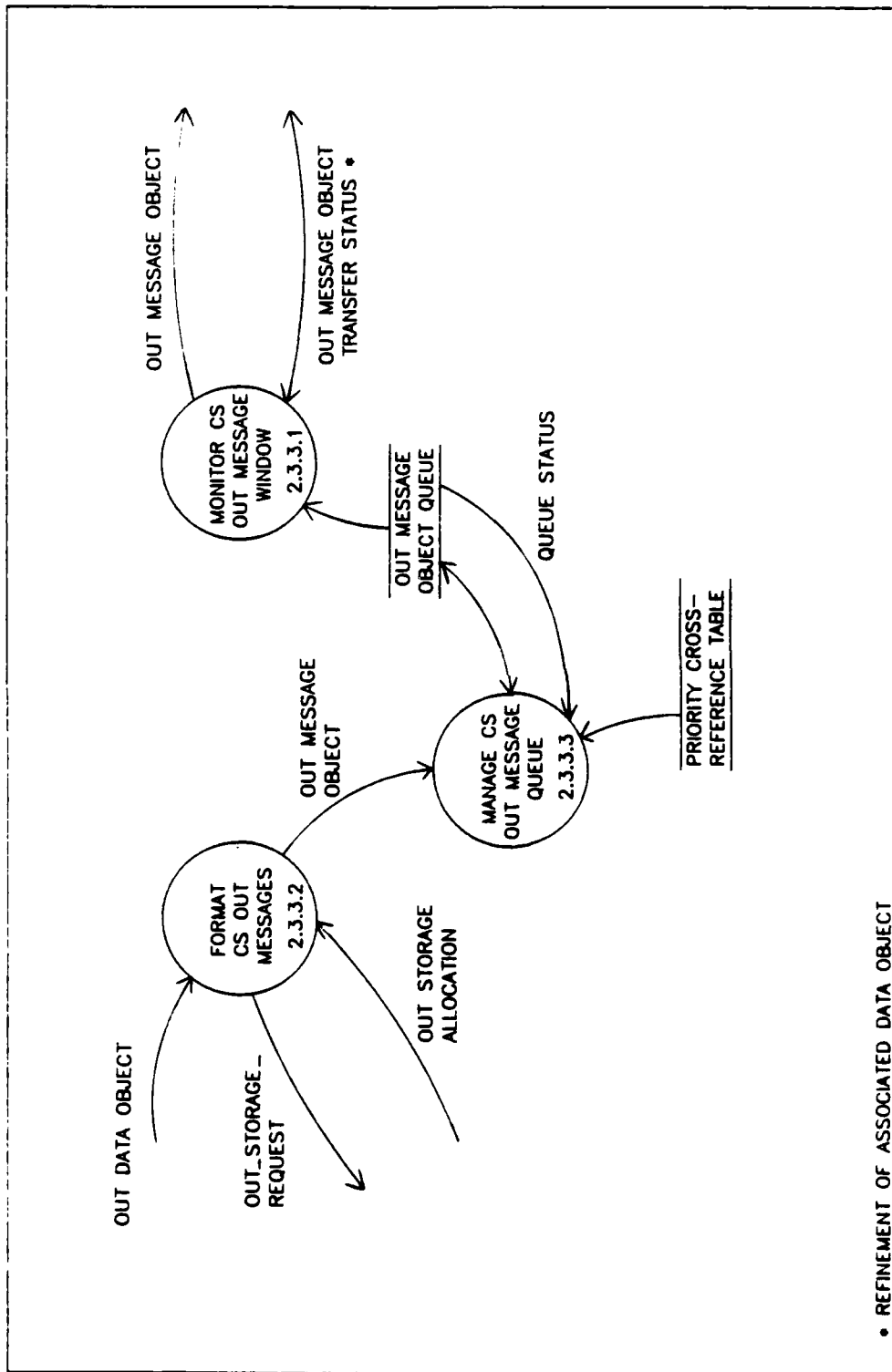


Figure F.2.3.3. Manage CS Out Messages Data Flow Diagram

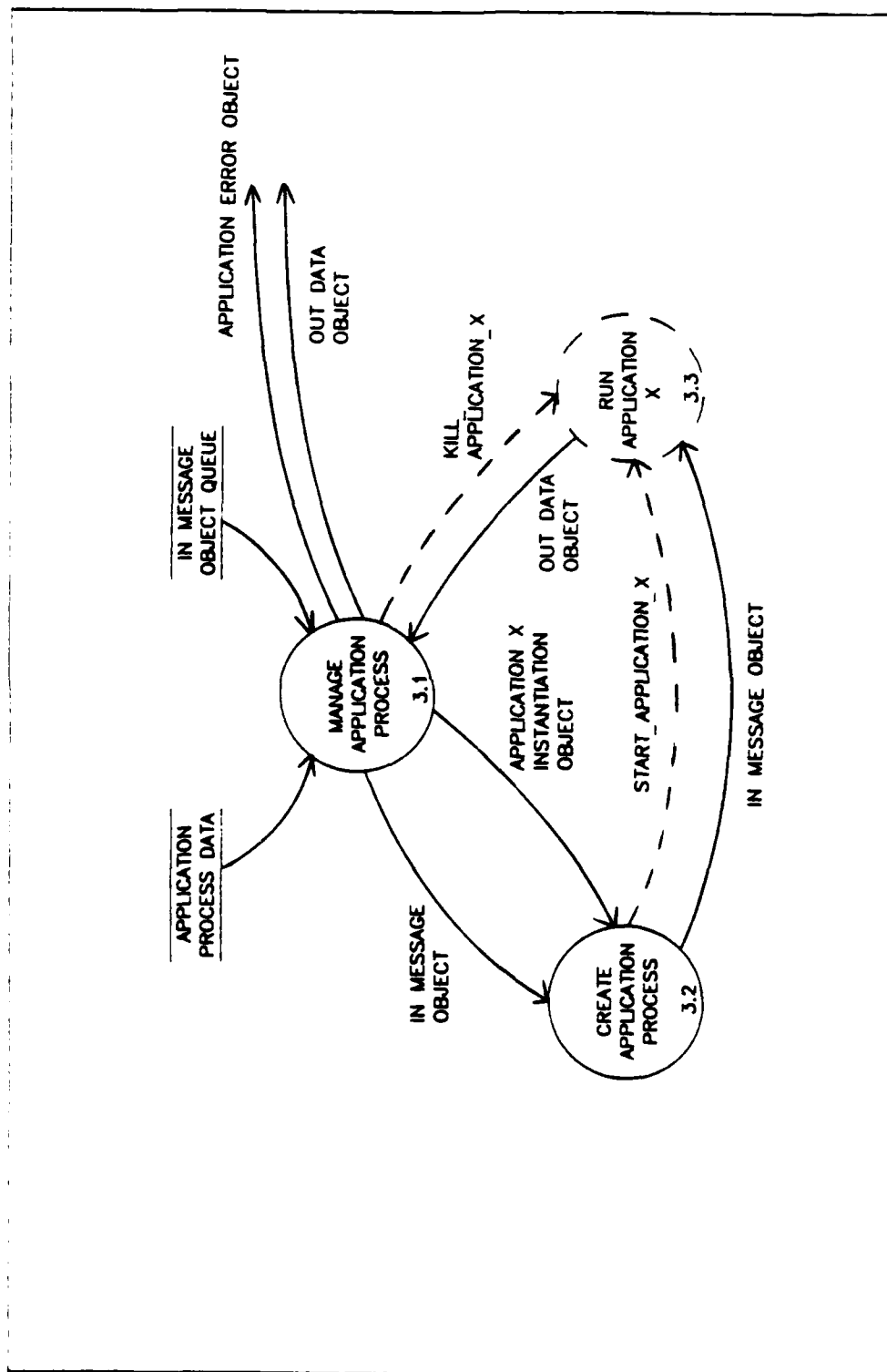


Figure F.3.0. Run Application Process Data Flow Diagram

APPENDIX G

SYSTEM SOFTWARE HIERARCHY CHARTS

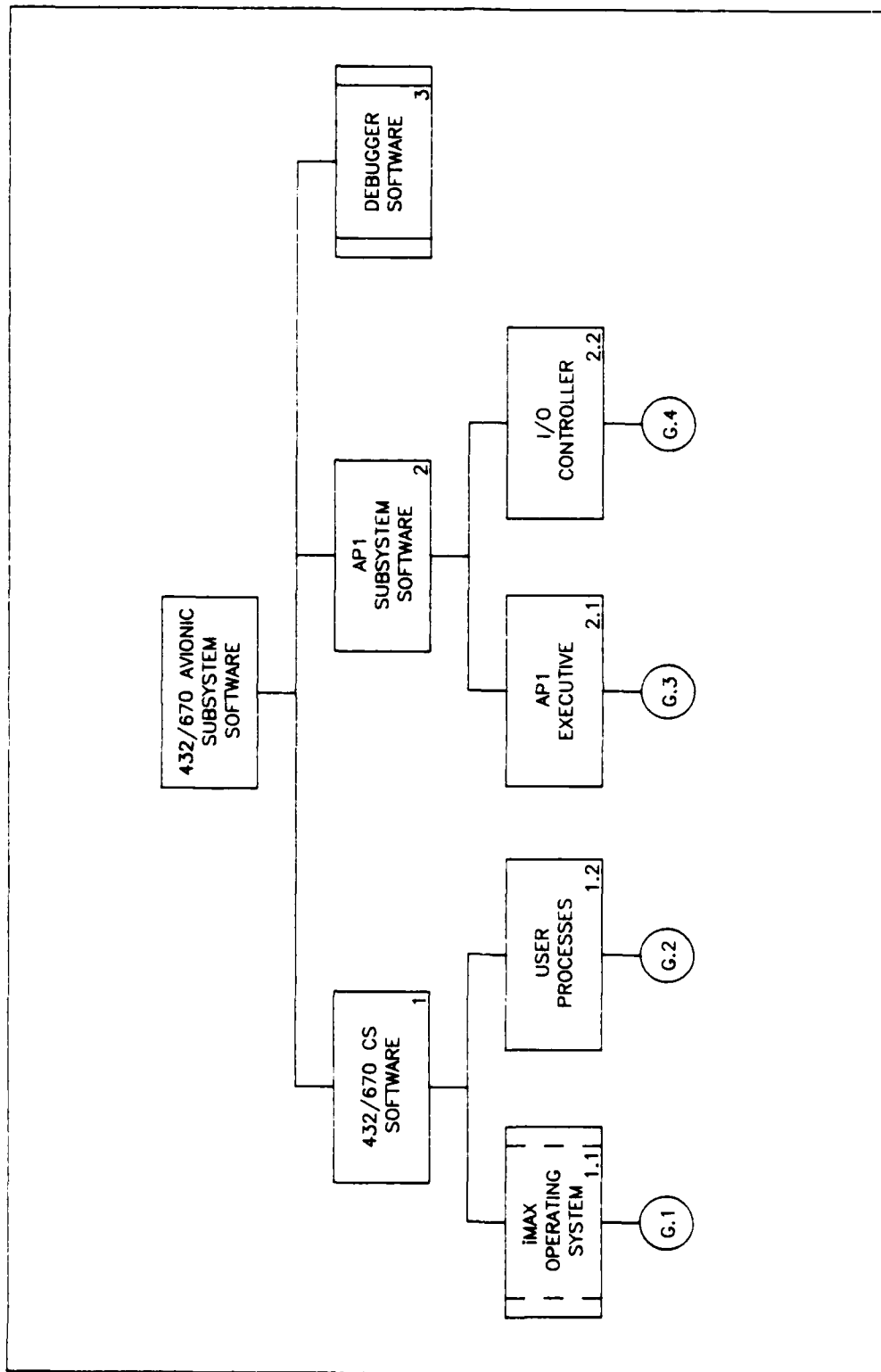


Figure G.O. Top Level Software Hierarchy Chart

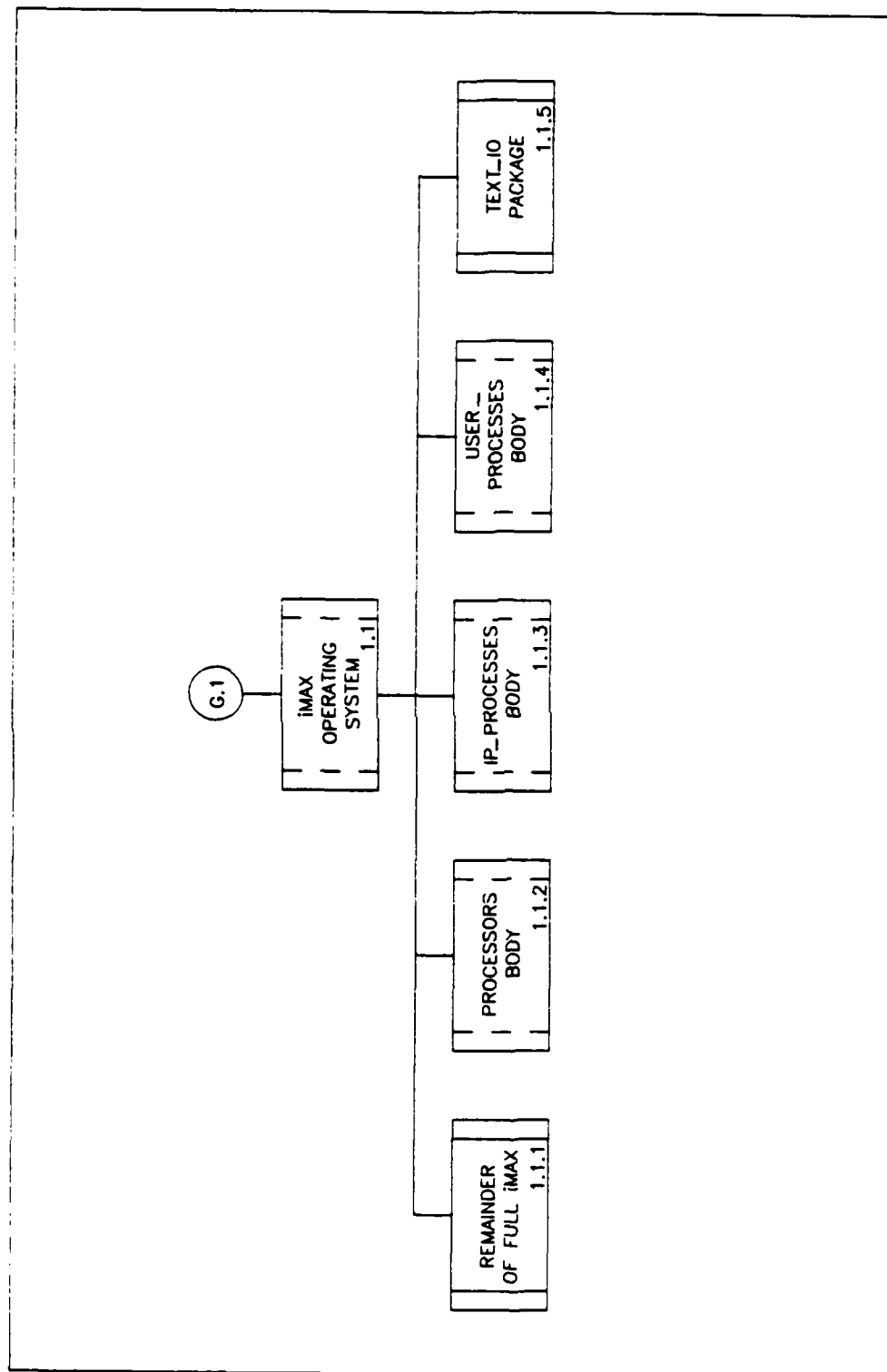


Figure G.1. iMAX Operating System Hierarchy Chart

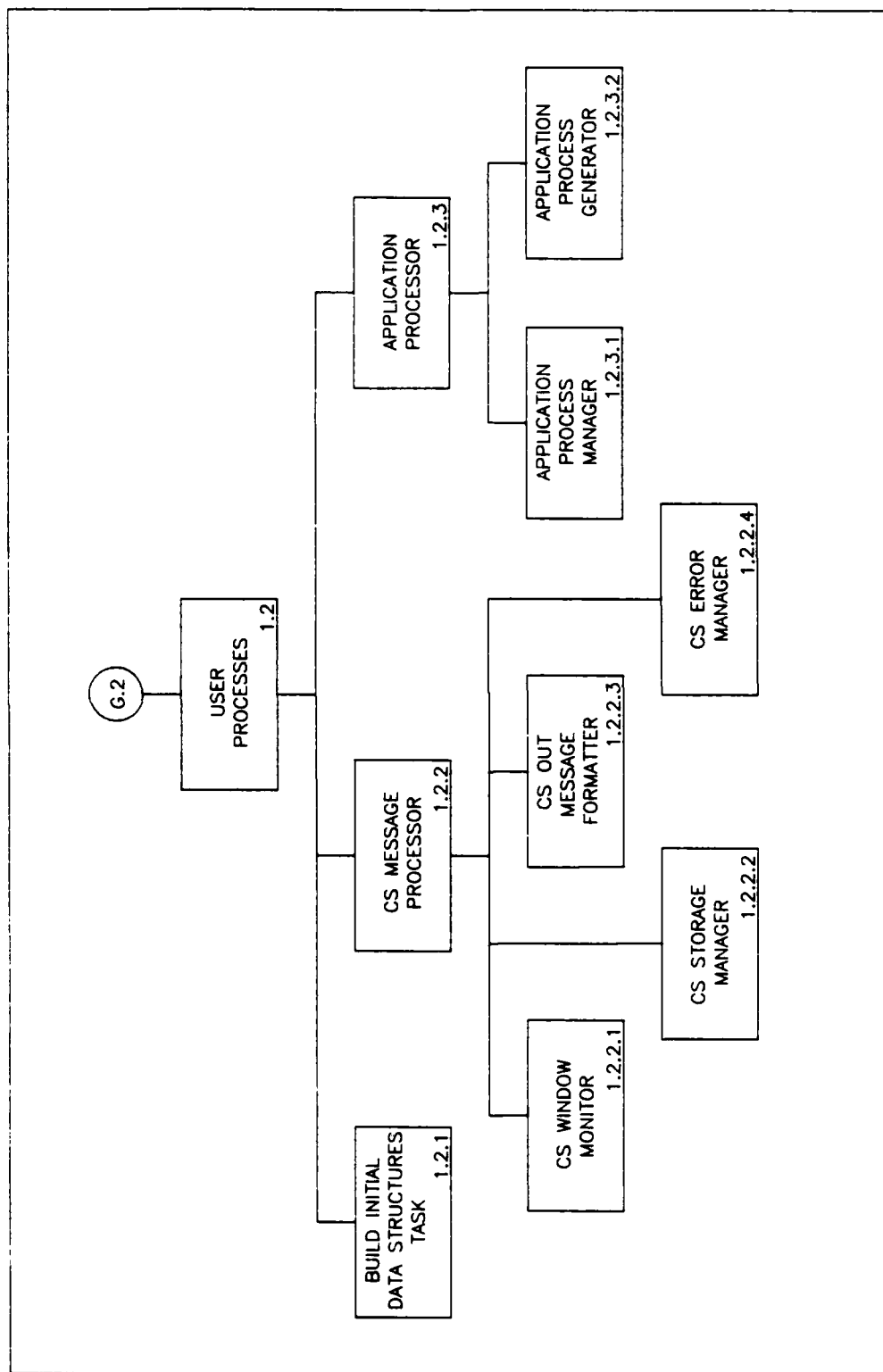


Figure G.2. User Processes Hierarchy Chart

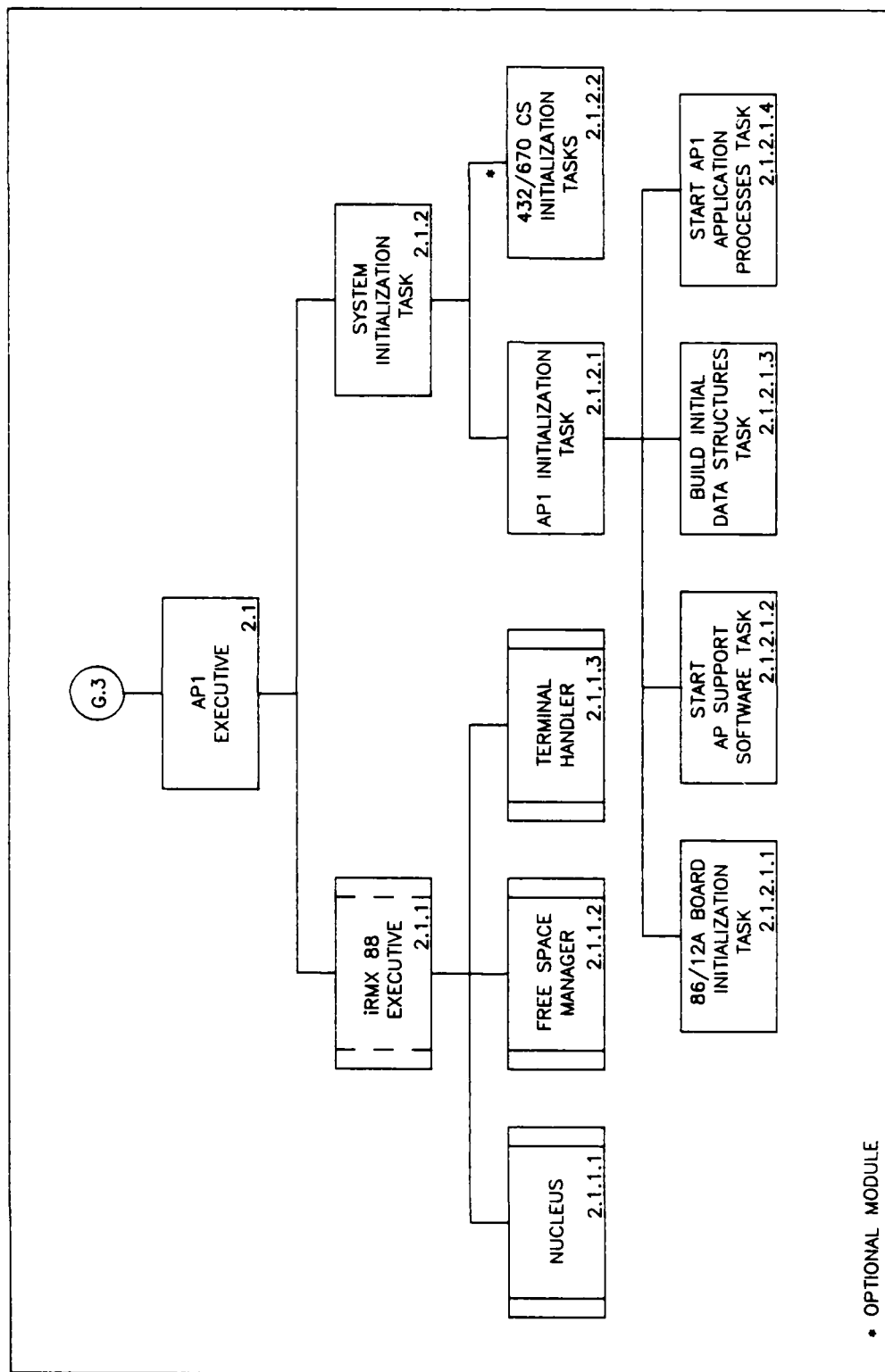


Figure G.3. AP1 Executive Hierarchy Chart

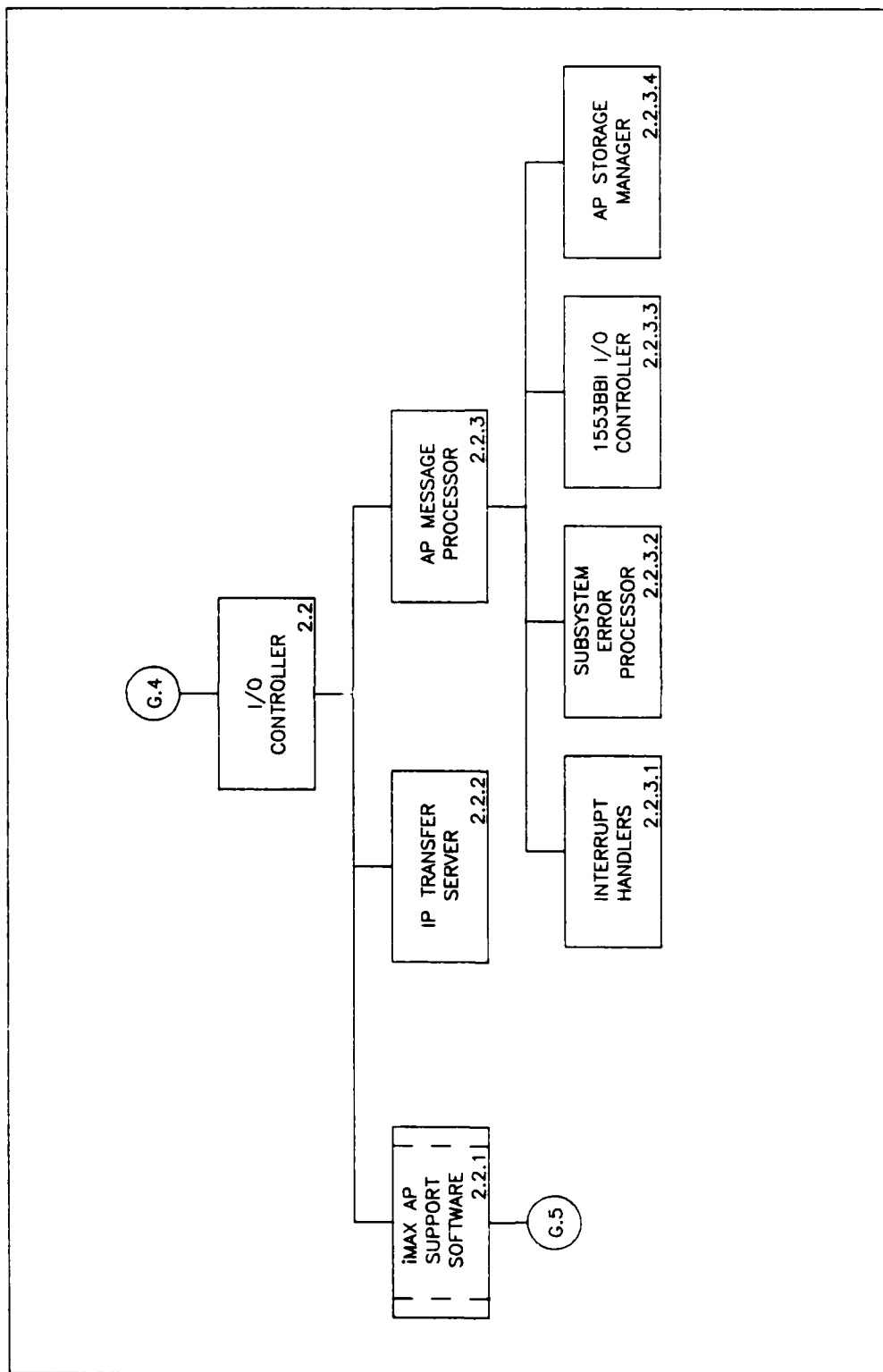


Figure G.4. I/O Controller Hierarchy Chart

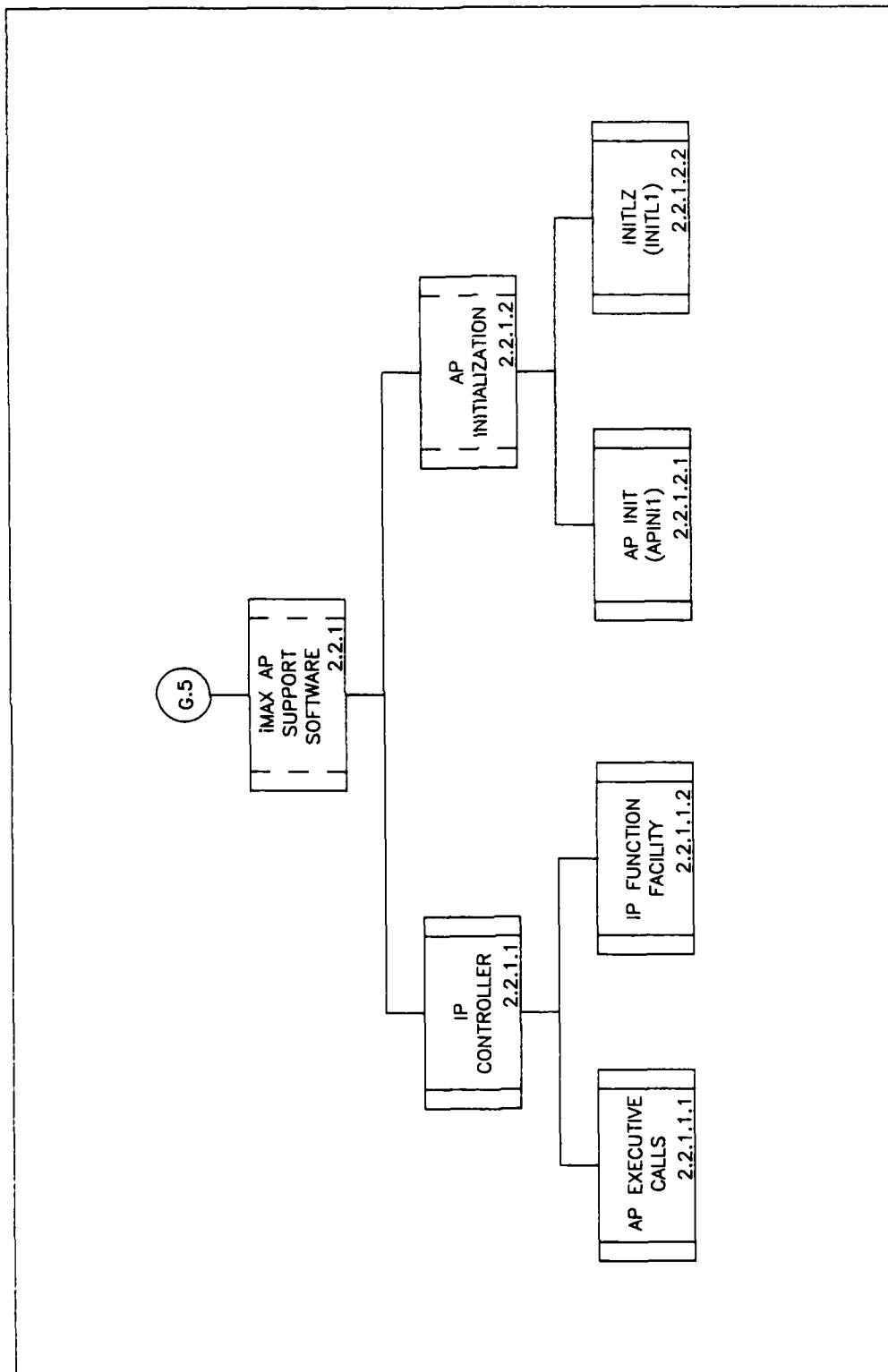


Figure G.5. iMAX AP Support Software Hierarchy Chart

APPENDIX H

SOFTWARE DATA DICTIONARY

Data Flow Diagram Process Definitions

PROCESS: 1 INITIALIZE 432/670 SYSTEM

DESCRIPTION: Initialization process for all 432/670 hardware and software.

PROCESS: 1.1 INITIALIZE CS

DESCRIPTION: Initialization process for the 432/670 CS. Most of this processing is under Debugger/operator control. Most of the remaining processing is controlled by iMAX. See Reference 17, Chapter INI for detailed information on subprocesses below.

PROCESS: 1.1.1 INITIALIZE 432/670 HARDWARE

DESCRIPTION: Debugger process that resets all 432/670 hardware and clears the memory.

PROCESS: 1.1.2 LOAD CS MEMORY

DESCRIPTION: Debugger process that loads the initial execution environment into CS memory.

PROCESS: 1.1.3 SET DEBUGGER IP LOGICAL

DESCRIPTION: Debugger process that enables logical addressing for the Debugger IP.

PROCESS: 1.1.4 START GDP #1

DESCRIPTION: Debugger process that starts the first GDP, which proceeds to execute the initialization code loaded previously.

PROCESS: 1.1.5 INITIALIZE iMAX

DESCRIPTION: iMAX process that creates required system objects and starts the internal support processes.

PROCESS: 1.1.6 START REMAINING GDPS

DESCRIPTION: iMAX process that starts all other GDPS in the system, as indicated by the system configuration information in iMAX.

PROCESS: 1.1.7 START IP PROCESSES

DESCRIPTION: iMAX process that starts the non-Debugger IPs (AP1 IP) and enables their logical addressing mode.

PROCESS: 1.1.8 START STATIC USER PROCESSES

DESCRIPTION: iMAX process that starts execution of all static application (user) processes declared within iMAX.

PROCESS: 1.1.9 BUILD DATA TABLES

DESCRIPTION: User process that builds data tables required for 1553B message processing support and determines the CS memory allocation for incoming and outgoing message storage.

PROCESS: 1.2 INITIALIZE AP1

DESCRIPTION: Process that initializes AP1 Subsystem hardware and software.

PROCESS: 1.2.1 BOOT iRMX 88

DESCRIPTION: ROM-based process that boots the iRMX 88 operating system on system power-on or reset.

PROCESS: 1.2.2 INITIALIZE AP SUPPORT SOFTWARE

DESCRIPTION: Process that initializes the IP window management software to allow communication with the 432/670 CS.

PROCESS: 1.2.3 BUILD DATA STRUCTURES

DESCRIPTION: Process that builds the Priority Cross-Reference Table, and initializes the Out Message Queue storage blocks and linked lists.

PROCESS: 1.2.4 INITIALIZE AP1 HARDWARE

DESCRIPTION: Process that initializes the 86/12A hardware, including the programmable interrupt controller, programmable interval timer, and USART.

PROCESS: 1.2.5 START AP1 APPLICATION PROCESSES

DESCRIPTION: Process that starts user static application processes after all other AP1 initialization is complete.

PROCESS: 2 TRANSFER MESSAGES

DESCRIPTION: Process that transfers 1553B messages from the 1553BBI hardware interface (designated RX/TX 1553B Messages, Process 4) to the 432/670 CS.

PROCESS: 2.1 AP PROCESS MESSAGE

DESCRIPTION: Collection of message processes which execute on the AP1 Subsystem whose function is to transfer 1553B messages between the 1553BBI (see 2) and the IP-resident transfer processes.

PROCESS: 2.1.1 PROCESS IN MESSAGE

DESCRIPTION: AP process that transfers messages from the 1553BBI In Message Storage Module to temporary storage for subsequent transfer to the CS.

PROCESS: 2.1.1.1 SERVICE INT3

DESCRIPTION: Interrupt task that responds to INT3 (IN MSG S/R, see Appendix D), determines the number of words to be read and initiates reading of incoming messages from the In Message Storage Module's FIFO Buffer.

PROCESS: 2.1.1.2 READ 1553B IN MESSAGE

DESCRIPTION: Process that reads all data words, including the tag word (separate or composite message) from the In Message Storage Module FIFO, and clears INT3 following completion of message transfer.

PROCESS: 2.1.1.3 MANAGE IN MESSAGE STORAGE

DESCRIPTION: Process that manages the temporary storage used by incoming messages while awaiting transfer to the CS. When storage is full, no transfers are accepted from Process 2.1.1.2, thus inhibiting the reading of additional messages.

PROCESS: 2.1.2 PROCESS OUT MESSAGE

DESCRIPTION: AP process that accepts outgoing 1553B messages from the CS for subsequent transfer to the 1553BBI Out Message Storage Module.

PROCESS: 2.1.2.1 SERVICE INT5

DESCRIPTION: Interrupt task that responds to INT5 (OUT MSG S/R, see Appendix D) to initiate transfer of the next outgoing 1553B message to the 1553BBI Out Message Storage Module.

PROCESS: 2.1.2.2 MANAGE OUT MESSAGE QUEUE

DESCRIPTION: Process that manages the priority/FIFO Out Message Queue which the holds outgoing 1553B messages.

PROCESS: 2.1.2.3 WRITE TAG WORD

DESCRIPTION: Process that writes the tag word associated with the outgoing 1553B message to the 1553BBI Tag Buffer.

PROCESS: 2.1.2.4 WRITE OUT MESSAGE

DESCRIPTION: Process that writes all data words of an outgoing 1553B message to the 1553BBI Out Message Buffer and subsequently clears INT5.

PROCESS: 2.1.3 PROCESS SUBSYSTEM ERRORS

DESCRIPTION: Subsystem error handling process.

PROCESS: 2.1.3.1 SERVICE CS ERRORS

DESCRIPTION: AP process that accepts and stores current CS error codes until bus controller is notified of the errors and acknowledges the transmission.

PROCESS: 2.1.3.2 CODE SUBSYSTEM ERRORS

DESCRIPTION: AP process that consolidates all current subsystem error codes into a composite code for transmission to the bus controller.

PROCESS: 2.1.3.3 SERVICE INTO

DESCRIPTION: Interrupt task that responds to INTO (SS ERR), initiating transfer of the subsystem error codes (hardware and software) to the bus controller.

PROCESS: 2.1.3.4 WRITE SUBSYSTEM ERROR

DESCRIPTION: AP process that sends the AP CMDs which set the SS FLAG to instruct the bus controller to read the 1553BBI error registers, and subsequently clears INTO.

PROCESS: 2.1.3.5 SERVICE INT1

DESCRIPTION: Interrupt task that services INT1 (IN BUFFER FULL) by sending the appropriate error code to Process 2.1.3.2 prior to clearing INT1.

PROCESS: 2.2 IP TRANSFER MESSAGE

DESCRIPTION: Set of processes that performs the memory-mapped input/output transfers between the AP1 memory subrange and the 432/670 CS object associated with the windows through which the transfers occur.

PROCESS: 2.2.1 TRANSFER MESSAGE AP-CS

DESCRIPTION: IP process that handles the 1553B message transfers from the AP1 Subsystem to the 432/670 CS.

PROCESS: 2.2.2 TRANSFER MESSAGE CS-AP

DESCRIPTION: IP process that handles the 1553B message transfers from the 432/670 CS back to the AP1 Subsystem.

PROCESS: 2.2.3 TRANSFER ERROR CS-AP

DESCRIPTION: IP process that handles the error information transfers from the 432/670 CS to the AP1 Subsystem. This process generates an interrupt (INT4) to the AP1 upon detection of a new error object awaiting transfer.

PROCESS: 2.3 CS PROCESS MESSAGE

DESCRIPTION: Set of CS processes which manage the 1553B message objects transferred for processing by the AP1 Subsystem.

PROCESS: 2.3.1 MANAGE CS IN MESSAGES

DESCRIPTION: CS process which accepts incoming message objects from the AP1 Subsystem and manages the In Message Object Queue which feeds the message objects to Process 3.0.

PROCESS: 2.3.1.1 MONITOR CS IN MESSAGE WINDOW

DESCRIPTION: Process which monitors the status field in the incoming message window object to detect arrival of a new message, and requests storage for the new message. The process updates the same status field after storage is allocated and transfer of the object reference to the In Message Object Queue is complete.

PROCESS: 2.3.1.2 MANAGE CS IN MESSAGE QUEUE

DESCRIPTION: CS process which manages the priority/FIFO queue in which all incoming message objects are stored while awaiting servicing. This process accepts the storage allocation from Process 2.3.2 (requested by Process 2.3.1.1). Released storage space is automatically recycled by IMAX.

PROCESS: 2.3.2 MANAGE CS STORAGE

DESCRIPTION: CS process that allocates storage for incoming and outgoing message objects. In order to minimize processing bottlenecks, maximum allowable storage use by either queue is limited by partitioning established during system initialization.

PROCESS: 2.3.3 MANAGE CS OUT MESSAGES

DESCRIPTION: Process which accepts processed message data from Process 3.0 for return to the API Subsystem.

PROCESS: 2.3.3.1 MONITOR CS OUT MESSAGE STORAGE

DESCRIPTION: CS process which monitors the transfer status field in the outgoing message window object, in order to determine when the previous message has been accepted by the API Subsystem. This process also updates this status field when a new message object reference is transferred to the window.

PROCESS: 2.3.3.2 FORMAT CS OUT MESSAGES

DESCRIPTION: CS process which reformats the output data received from Process 3.0. Output data containing in excess of 32 data words requires multiple 1553B messages for return to the avionic subsystem.

PROCESS: 2.3.3.3 MANAGE CS OUT MESSAGE QUEUE

DESCRIPTION: CS process that manages the outgoing message object priority/FIFO queue.

PROCESS: 2.3.4 MANAGE CS ERRORS

DESCRIPTION: CS process that accepts error codes from other CS processes and formats the error data into error objects for transfer to the AP1 Subsystem.

PROCESS: 3 PROCESS MESSAGE DATA

DESCRIPTION: CS process the performs the actual calculations on avionic system data.

PROCESS: 3.0 RUN APPLICATION PROCESS

DESCRIPTION: Alias for Process 3 which is used at the system level to provide a clear differentiation between processing message data and processing or handling the messages themselves.

PROCESS: 3.1 MANAGE APPLICATION PROCESS

DESCRIPTION: CS process which reads highest priority message object from the In Message Object Queue, retrieves the application process information from the Application Process Data Table and builds the object used in creating the application process. This process also accepts the output data and any error data from the application process, and subsequently kills the process.

PROCESS: 3.2 CREATE APPLICATION PROCESS

DESCRIPTION: CS process which accepts the instantiation object built by Process 3.1, builds the dynamic application process (Application X) and starts the process.

PROCESS: 3.3 RUN APPLICATION X

DESCRIPTION: This is the actual dynamic application process which is created in response to a processing requested by the incoming 1553B messages tag word.

Data Flow Diagram Flow and Table/Queue Definitions

FLOW: 1553B IN MESSAGE

DESCRIPTION: An incoming 1553B message, including the tag word and all message data words. The 1553BBI hardware masks the actual transfer mechanism from the AP, so that the AP always perceives the message as a single entity, even when the tag and data words are received as separate messages.

FLOW: 1553B IN MESSAGE TRANSFER STATUS

DESCRIPTION: Information concerning the status of the message currently being transferred through the IP window reserved for transfer of incoming messages from AP1 to the CS. AP1 accesses this information through the reserved memory location in the window memory subrange.

FLOW: 1553B MESSAGES

DESCRIPTION: A composite flow consisting of 1553B IN MESSAGE and 1553B OUT MESSAGE.

FLOW: 1553B OUT MESSAGE

DESCRIPTION: An outgoing 1553B message, including the tag word and all message data words. The only point at which the tag and data words are treated as two separate entities is during the final transfer of the message back to the 1553BBI.

FLOW: 1553B OUT MESSAGE TRANSFER STATUS

DESCRIPTION: Information concerning the status of the message currently being transferred through the IP window reserved for transfer of outgoing message objects from the CS to AP1. AP1 accesses this information through the reserved memory location in the window memory subrange.

FLOW: 1553BBI CONTROL/STATUS

DESCRIPTION: A composite flow which contains all of the 1553BBI control functions and all of the required status responses.

FLOW: 1553BBI_ERROR_CODE

DESCRIPTION: The error code generated by Process 2.1.3.3, SERVICE INTO.

FLOW: 1553BBI INTERRUPTS

DESCRIPTION: A composite flow consisting of the hardware interrupts generated by the 1553BBI which are serviced by the application software.

FLOW: AP CLEAR INT1 CMD

DESCRIPTION: Same as the AP CL INT1 signal. See Appendix D for description.

FLOW: AP CMDS

DESCRIPTION: A composite flow consisting of all of the 1553BBI control signals generated by AP1. See Appendix D for a complete description.

FLOW: AP ERROR CMDS

DESCRIPTION: A subset of AP CMDS which controls the writing of the subsystem error code to the 1553BBI Error Handler and the setting of SS FLAG.

FLOW: AP READ MSG CMDS

DESCRIPTION: A subset of AP CMDS which controls the reading of the tag word and message data words from the 1553BBI In Message Storage Module.

FLOW: AP WRITE DATA CMDS

DESCRIPTION: A subset of AP CMDS which controls the writing of the message data words to the 1553BBI Out Message Storage Module.

FLOW: AP WRITE MSG CMDS

DESCRIPTION: A composite flow consisting of AP WRITE DATA CMDS and AP WRITE TAG CMDS.

FLOW: AP WRITE TAG CMDS

DESCRIPTION: A subset of AP CMDS which controls the writing of the message tag word to the 1553BBI Out Message Storage Module.

FLOW: APPLICATION ERROR OBJECT (ALIAS: APPL ERROR OBJECT)

DESCRIPTION: The error object created by Process 3.1, MANAGE APPLICATION PROCESS, upon detection of an abnormal status code in the OUT DATA OBJECT status field.

TABLE: APPLICATION PROCESS DATA TABLE

DESCRIPTION: The data table containing specific application program information which is used by the 432/670 CS in building the APPLICATION X INSTANTIATION OBJECT.

FLOW: APPLICATION X INSTANTIATION OBJECT

DESCRIPTION: The object containing specific information concerning the application program to be used to process the data contained in the current 1553B message. This object is built by Process 3.1 using information in the APPLICATION PROCESS DATA TABLE, and is sent to Process 3.2, CREATE APPLICATION PROCESS.

FLOW: CS ERROR CODES

DESCRIPTION: The error codes passed from the CS to AP1 via the error object. These codes are combined with the AP1 error codes to create the composite SS_ERROR_CODE.

FLOW: ERROR OBJECT

DESCRIPTION: An object built by Process 2.3.4, MANAGE CS ERRORS, for use in transferring current error status information to AP1 via the reserved IP error window.

FLOW: ERROR OBJECT TRANSFER STATUS

DESCRIPTION: Information concerning the status of the error object currently being transferred through the IP window reserved for transfer of error objects from the CS to AP1. This information is stored as a field in the ERROR OBJECT and is accessed by the CS as a refinement object.

FLOW: ERROR/STATUS DATA

DESCRIPTION: A composite flow consisting of SS_ERROR_CODE and any 1553BBI status data sent in response to the AP CMDS.

FLOW: IN_BUFFER_FULL_ERROR_CODE

DESCRIPTION: The error code generated by Process 2.1.3.5, SERVICE INT1.

FLOW: IN MESSAGE OBJECT

DESCRIPTION: The incoming message object which contains the message tag word, message data words, and transfer status field.

QUEUE: IN MESSAGE OBJECT QUEUE

DESCRIPTION: The priority/FIFO queue used to store the incoming message objects prior to processing by Process 3.0, RUN APPLICATION X.

FLOW: IN MESSAGE OBJECT TRANSFER STATUS

DESCRIPTION: Information concerning the status of the message currently being transferred through the IP window reserved for transfer of incoming messages from AP1 to the CS. This information is stored as a field in the IN MESSAGE OBJECT and is accessed by the CS as a refinement object.

FLOW: IN STORAGE ALLOCATION

DESCRIPTION: Storage allocated for use in adding a new incoming message object to the IN MESSAGE OBJECT QUEUE. The storage is requested by Process 2.3.1.1, MONITOR CS IN MESSAGE WINDOW.

FLOW: IN_STORAGE_REQUEST

DESCRIPTION: The request for storage for a new incoming message which is sent by Process 2.3.1.1, MONITOR CS IN MESSAGE WINDOW, to Process 2.3.2, MANAGE CS STORAGE.

FLOW: INIT COMMAND

DESCRIPTION: The Debugger command that initiates the 432/670 CS hardware initialization.

FLOW: INIT_HARDWARE

DESCRIPTION: The command generated during iRMX 88 startup which initiates the API hardware initialization.

FLOW: INIT_STORAGE

DESCRIPTION: The command generated during iRMX 88 startup which initiates the building of the API data structures and tables which are used in queue management.

FLOW: INITIAL EXECUTION ENVIRONMENT

DESCRIPTION: The linked external object descriptor (EOD) which is loaded into 432/670 CS memory at system initialization. The EOD contains the iMAX operating system and all code for the static application processes.

FLOW: INITIALIZE FLAGS

DESCRIPTION: A composite flow consisting of START_USER_PROCESSES and START_API_TASKS.

FLOW: INTO (SS ERR)

DESCRIPTION: 1553BBI interrupt. See Appendix D for description.

FLOW: INT1 (IN BUFFER FULL)

DESCRIPTION: 1553BBI interrupt. See Appendix D for description.

FLOW: INT3 (IN MSG S/R)

DESCRIPTION: 1553BBI interrupt. See Appendix D for description.

FLOW: INT4 (CS ERROR)

DESCRIPTION: The IP interrupt signal. This interrupt is used to notify API that an error object has been generated and sent to the IP window reserved for error transfers.

FLOW: INT5 (OUT MSG S/R)

DESCRIPTION: 1553BBI interrupt. See Appendix D for description.

FLOW: KILL_APPLICATION_X

DESCRIPTION: The command sent by Process 3.1, MANAGE APPLICATION PROCESS, to destroy the dynamic application process following acceptance of the OUT DATA OBJECT from that process.

FLOW: LOAD COMMAND

DESCRIPTION: The Debugger command that initiates loading of the INITIAL EXECUTION ENVIRONMENT into the 432/670 CS memory.

FLOW: MESSAGE OBJECT(S)

DESCRIPTION: A composite flow consisting of IN MESSAGE OBJECT and OUT MESSAGE OBJECT.

FLOW: OUT DATA OBJECT

DESCRIPTION: The object created by the dynamic application process which holds all data generated by the process. This object contains the tag word, all data words, and a status field which is monitored by the application manager, Process 3.1. This object must be reformatted prior to return to AP1 since it may contain more than 32 data words.

FLOW: OUT MESSAGE OBJECT

DESCRIPTION: The outgoing message object which contains the message tag word, message data words, and transfer status field.

QUEUE: OUT MESSAGE OBJECT QUEUE

DESCRIPTION: The priority/FIFO queue used to store the outgoing message objects prior to transfer back to the AP1 Subsystem.

FLOW: OUT MESSAGE OBJECT TRANSFER STATUS

DESCRIPTION: Information concerning the status of the message currently being transferred through the IP window reserved for transfer of outgoing message objects from the CS to AP1. This information is stored as a field in the OUT MESSAGE OBJECT and is accessed by the CS as a refinement object.

QUEUE: OUT MESSAGE QUEUE

DESCRIPTION: The priority/FIFO queue used to store the outgoing 1553B messages prior to transfer to the 1553BBI Out Message Storage Module.

FLOW: OUT_OF_SPACE

DESCRIPTION: The error message sent by Process 2.3.2, MANAGE CS STORAGE, to Process 2.3.4, MANAGE CS STORAGE, when all of the originally allocated space allowed for either incoming or outgoing message objects has been exhausted.

FLOW: OUT STORAGE ALLOCATION

DESCRIPTION: Storage allocated for use in adding each outgoing message object to the OUT MESSAGE OBJECT QUEUE. Storage is requested for each outgoing object as it is created by Process 2.3.3.2, FORMAT CS OUT MESSAGES.

FLOW: OUT_STORAGE_REQUEST

DESCRIPTION: The storage requests sent by Process 2.3.3.2, FORMAT CS OUT MESSAGES, to Process 2.3.2, MANAGE CS STORAGE.

TABLE: PRIORITY CROSS-REFERENCE TABLE

DESCRIPTION: The table containing the priority information for the various application programs used to process the 1553B message data. Separate tables are built for 432/670 CS and AP1 use in managing their respective priority/FIFO message queues.

FLOW: READ MSG CMD

DESCRIPTION: Command generated by Process 2.1.1.1, SERVICE INT3, which initiates the reading of the incoming 1553B message from the 1553BBI In Message Storage Module.

FLOW: SEND MESSAGE OBJECT

DESCRIPTION: The command sent by Process 2.3.1.2, MANAGE CS IN MESSAGE QUEUE, to Process 2.3.1.1, MONITOR CS IN MESSAGE WINDOW, upon receipt of the requested storage allocation. The window monitor process must hold the incoming message object until receiving this notification.

FLOW: SET_IP_LOGICAL

DESCRIPTION: The hardware signal sent to the AP1 IP to set the IP into the logical addressing mode.

FLOW: SS_ERROR_CODE

DESCRIPTION: The composite subsystem error code which reflects all current AP1 and 432/670 CS errors. This code is sent to the 1553BBI Error Handler by Process 2.1.3, PROCESS SUBSYSTEM ERRORS.

FLOW: START_AP

DESCRIPTION: The command generated during iRMX 88 startup which initiates the AP iMAX Support Software initialization.

FLOW: START_AP1_TASKS

DESCRIPTION: The command generated after completion of the AP1 initialization tasks which starts execution of the application tasks.

FLOW: START_APPLICATION_X

DESCRIPTION: The command which starts execution of the dynamic application process.

FLOW: START COMMAND

DESCRIPTION: The Debugger command which initiates the GDP #1 startup.

FLOW: START CS COMMANDS

DESCRIPTION: A composite flow consisting of INIT COMMAND, LOAD COMMAND, and START COMMAND.

FLOW: START_USER_PROCESSES

DESCRIPTION: The command generated after completion of the 432/670 initialization tasks which starts execution of all user static processes, including Process 1.1.9, BUILD DATA TABLES.

FLOW: STORAGE STATUS

DESCRIPTION: Information concerning the status of the storage allocated to temporary storage of incoming 1553B messages which are awaiting transfer to the 432/670 CS.

TABLE: TABLE DATA TABLE

DESCRIPTION: Tables containing the information used in building the APPLICATION PROCESS DATA TABLE and the PRIORITY CROSS-REFERENCE TABLE.

FLOW: TAG_DECODE_ERROR

DESCRIPTION: The error message sent by Process 2.3.1, MANAGE CS IN MESSAGES, to Process 2.3.4, MANAGE CS STORAGE, when the tag word contains an error.

FLOW: TAG WORD

DESCRIPTION: See Appendix D for description.

FLOW: TRANSFER STATUS

DESCRIPTION: A composite flow consisting of 1553B IN MESSAGE TRANSFER STATUS and 1553B OUT MESSAGE TRANSFER STATUS.

FLOW: WRITE MESSAGE

DESCRIPTION: Command generated by Process 2.1.2.1, SERVICE INT5, which initiates the writing of the incoming 1553B message to the 1553BBI Out Message Storage Module.

Hierarchy Chart Module Definitions

MODULE: 1 432/670 CS Software

DESCRIPTION: All software which resides in the 432/670 memory space for execution. The iMAX operating system and all user software is included.

MODULE: 1.1 iMAX OPERATING SYSTEM

DESCRIPTION: The 432/670 CS operating system which consists of a set of Ada packages which are customized by the user to suit the application. See Reference 17 for detailed information concerning the iMAX configuration and functions.

MODULE: 1.1.1 REMAINDER OF iMAX

DESCRIPTION: All full iMAX packages not explicitly referred to in other modules since user modification is optional. All available full iMAX packages should be included in the configuration.

MODULE: 1.1.2 PROCESSORS BODY

DESCRIPTION: User-customized iMAX package body in which the physical processor identification for all GDPs and IPs is declared.

MODULE: 1.1.3 IP_PROCESSES BODY

DESCRIPTION: User-customized iMAX package body which controls the IP process initialization and execution. This package must be modified to add the AP1 IP.

MODULE: 1.1.4 USER_PROCESSES BODY

DESCRIPTION: iMAX package body supplied as a shell to which the user must add all declarations for the user static processes.

MODULE: 1.1.5 TEXT_IO PACKAGE

DESCRIPTION: The standard Ada input/output package which is supplied with iMAX. This package, which is not part of iMAX, must be included to provide support for Debugger and optional terminal interaction.

MODULE: 1.2 USER PROCESSES

DESCRIPTION: All application software packages required to provide the 1553B message handling and data processing functions.

MODULE: 1.2.1 BUILD INITIAL DATA STRUCTURES TASK

DESCRIPTION: Module in which the software which performs the functions described in Process 1.1.9, BUILD DATA TABLES, resides.

MODULE: 1.2.2 CS MESSAGE PROCESSOR

DESCRIPTION: Module in which the software which performs the functions described in Process 2.3, CS PROCESS MESSAGE, resides.

MODULE: 1.2.2.1 CS WINDOW MONITOR

DESCRIPTION: Module in which the software which performs the functions described in Processes 2.3.1.1, MONITOR CS IN MESSAGE WINDOW, and 2.3.3.1, MONITOR CS OUT MESSAGE WINDOW, resides.

MODULE: 1.2.2.2 CS STORAGE MANAGER

DESCRIPTION: Module in which the software which performs the functions described in Processes 2.3.2, MANAGE CS STORAGE, 2.3.1.2, MANAGE CS IN MESSAGE QUEUE, and 2.3.3.3, MANAGE CS OUT MESSAGE QUEUE, resides.

MODULE: 1.2.2.3 CS OUT MESSAGE FORMATTER

DESCRIPTION: Module in which the software which performs the functions described in Process 2.3.3.2, FORMAT CS OUT MESSAGES, resides.

MODULE: 1.2.2.4 CS ERROR MANAGER

DESCRIPTION: Module in which the software which performs the functions described in Process 2.3.4, MANAGE CS ERRORS, resides.

MODULE: 1.2.3 APPLICATION PROCESSOR

DESCRIPTION: Module in which the software which performs the functions described in Process 3.0, RUN APPLICATION PROCESSES, resides.

MODULE: 1.2.3.1 APPLICATION PROCESS MANAGER

DESCRIPTION: Module in which the software which performs the functions described in Process 3.1, APPLICATION PROCESS MANAGER, resides.

MODULE: 1.2.3.2 APPLICATION PROCESS GENERATOR

DESCRIPTION: Module in which the software which performs the functions described in Process 3.2, CREATE APPLICATION PROCESS, resides.

MODULE: 2 AP1 SUBSYSTEM SOFTWARE

DESCRIPTION: All software which resides in the AP1 memory space for execution. The iRMX 88 operating system and all user software is included.

MODULE: 2.1 AP1 EXECUTIVE

DESCRIPTION: The module which contains all software which performs executive management functions for the AP1 Subsystem, including the operating system and system initialization code.

MODULE: 2.1.1 iRMX 88 EXECUTIVE

DESCRIPTION: The user-configurable real-time, multitasking AP1 operating system. See Reference 34 for detailed information concerning iRMX 88 configuration and functions.

MODULE: 2.1.1.1 NUCLEUS

DESCRIPTION: The kernel of the iRMX 88 operating system. The version which supports interrupts and timed waits is required.

MODULE: 2.1.1.2 FREE SPACE MANAGER

DESCRIPTION: The iRMX 88 module which provides efficient memory management utilities for use by the application software.

MODULE: 2.1.1.3 TERMINAL HANDLER

DESCRIPTION: The iRMX 88 module which supports real-time, asynchronous input/output between a terminal and the application software. The input-output support version is required.

MODULE: 2.1.2 SYSTEM INITIALIZATION TASK

DESCRIPTION: All AP1-resident user-supplied code which is used to initialize the 432/670 hardware and software.

MODULE: 2.1.2.1 AP1 INITIALIZATION TASK

DESCRIPTION: All user-supplied code which is used to initialize the AP1 hardware and software.

MODULE: 2.1.2.1.1 86/12A BOARD INITIALIZATION TASK

DESCRIPTION: Module in which the software which performs the functions described in Process 1.2.4, INITIALIZE AP1 HARDWARE, resides.

MODULE: 2.1.2.1.2 START AP SUPPORT SOFTWARE TASK

DESCRIPTION: Module in which the software which performs the functions described in Process 1.2.2, INITIALIZE AP SUPPORT SOFTWARE, resides.

MODULE: 2.1.2.1.3 BUILD INITIAL DATA STRUCTURES TASK

DESCRIPTION: Module in which the software which performs the functions described in Process 1.2.3, BUILD DATA STRUCTURES, resides.

MODULE: 2.1.2.1.4 START AP1 APPLICATION PROCESSES TASK

DESCRIPTION: Module in which the software which performs the functions described in Process 1.2.5, START AP1 APPLICATION PROCESSES, resides.

MODULE: 2.1.2.2 432/670 CS INITIALIZATION TASKS

DESCRIPTION: Optional module which contains the code which allows the AP1 Subsystem to perform the 432/670 CS initialization tasks currently performed by the Debugger Subsystem.

MODULE: 2.2 I/O CONTROLLER

DESCRIPTION: Module which contains all code which supports transfer of data to and from AP1.

MODULE: 2.2.1 iMAX AP SUPPORT SOFTWARE

DESCRIPTION: The set of PL/M-86 packages supplied with iMAX which support the AP to CS communication via the IP windows. The user selects specific packages from the set to support the specific application. See Reference 17, Chapter IOI, for detailed information concerning this software and input/output implementation for peripheral subsystems.

MODULE: 2.2.1.1 IP CONTROLLER

DESCRIPTION: The iMAX packages which are used to control the IP. The complete set is used as supplied by Intel.

MODULE: 2.2.1.1.1 AP EXECUTIVE CALLS

DESCRIPTION: The package that provides system programmers with a standard interface to the AP executive and allows porting of peripheral subsystem software between different AP processor/executive combinations.

MODULE: 2.2.1.1.2 IP FUNCTION FACILITY

DESCRIPTION: The set of five packages that supports all high and low level IP control functions.

MODULE: 2.2.1.2 AP INITIALIZATION

DESCRIPTION: The set of packages which is used in initializing the AP hardware, IP CONTROLLER software and application tasks. The user selects the appropriate subset of two packages from the set provided.

MODULE: 2.2.1.2.1 AP INIT

DESCRIPTION: The package that provides access to the AP hardware and software initialization functions. The single terminal support version, APINI1, is required.

MODULE: 2.2.1.2.2 INITLZ

DESCRIPTION: The package that contains the initial AP system task. The single terminal support version, INITL1, is required.

MODULE: 2.2.2 IP TRANSFER SERVER

DESCRIPTION: Module in which the software which performs the functions described in Process 2.2, IP TRANSFER MESSAGE, resides.

MODULE: 2.2.3 AP MESSAGE PROCESSOR

DESCRIPTION: Module in which the software which performs the functions described in Process 2.1, AP PROCESS MESSAGE, resides.

MODULE: 2.2.3.1 INTERRUPT HANDLERS

DESCRIPTION: Module which contains all of the 1553BBI interrupt handling tasks.

MODULE: 2.2.3.2 SUBSYSTEM ERROR PROCESSOR

DESCRIPTION: Module containing the software which performs all functions described in Process 2.1.3, PROCESS SUBSYSTEM ERRORS, except for the 1553BBI interrupt servicing tasks.

MODULE: 2.2.3.3 1553BBI I/O CONTROLLER

DESCRIPTION: Module containing all of the software which directly controls communication with the 1553BBI, except for interrupt handling. The functions included are described in Processes 2.1.1.2, READ 1553B IN MESSAGE, 2.1.2.3, WRITE TAG WORD, and 2.1.2.4, WRITE OUT MESSAGE.

MODULE: 2.2.3.4 AP STORAGE MANAGER

DESCRIPTION: Module in which the software that performs the functions described in Processes 2.1.1.3, MANAGE IN MESSAGE STORAGE, and 2.1.2.2, MANAGE OUT MESSAGE QUEUE, resides.

MODULE: 3 DEBUGGER SOFTWARE

DESCRIPTION: The software hosted on the Debugger Subsystem which controls user interaction with the 432/670 CS for system initialization and troubleshooting.

Bibliography

1. Konno, Leslie T. Bus Interface Unit Design for the Distributed Processor/Memory System. MS Thesis. Wright-Patterson AFB, Ohio: School of Engineering, Air Force Institute of Technology, December 1977.
2. PX 13394. AN/AYK-15A Computer Maintenance Training Manual. Sperry Univac DSD Education Department.
3. SA 321 200. Prime Item Development Specification for DAIS Digital Command/Response, Time Division Multiplexing Data Bus. Wright-Patterson AFB, Ohio: DAIS Program Branch, Systems Avionics Division, Air Force Avionics Laboratory, February 5, 1979.
4. 171867-001. Intel 432 System Summary: Manager's Perspective. Santa Clara, California: Intel Corp., 1981.
5. 171873-001. iAPX 43201 iAPX 43202 VLSI General Data Processor, Preliminary Information. Santa Clara, California: Intel Corp., 1981.
6. MIL-STD-1553B Multiplex Applications Handbook. Wright-Patterson AFB, Ohio: ASD/ENASD, May 1, 1980.
7. MIL-STD-1553 Designer's Guide. Bohemia, New York: ILC Data Device Corp., 1982.
8. 171861-001. System 432/600 32-Bit Extensible Computer System, Preliminary Information. Santa Clara, California: Intel Corp., 1981.
9. 172098-001. System 432/600 System Reference Manual. Santa Clara, California: Intel Corp., 1981.
10. Rattner, Justin and Lattin, William W. "Ada Determines Architecture of 32-Bit Microprocessor", Electronics, Vol. 4, No. 54: 119-126 (February 24, 1981)
11. 171874-001. iAPX 43203 VLSI Interface Processor, Preliminary Information. Santa Clara, California: Intel Corp., 1981.
12. 171821-001. Introduction to the iAPX 432 Architecture. Santa Clara, California: Intel Corp., 1981.
13. 171863-003. iAPX 432 Interface Processor Architecture Reference Manual. Santa Clara, California: Intel Corp., 1981.

14. 171954-002. Introduction to the Intel 432 Cross Development System. Santa Clara, California: Intel Corp., 1982.
15. Rattner, Justin and Cox, George. "Object-Based Computer Architecture", Computer Architecture News, Vol. 8, No. 6:4-11 (October 15, 1980)
16. 171858-001. iAPX 432 Object Primer. Santa Clara, California: Intel Corp., 1981.
17. 172103-002. iMAX 432 Reference Manual. Santa Clara, California: Intel Corp., 1982.
18. Bayliss, John A., et al. "The Instruction Decoding Unit for the VLSI 432 General Data Processor", IEEE Journal of Solid State Circuits, Vol. SC-16, No. 5: 531-536 (October 1981)
19. Budde, David L., et al. "The Execution Unit for the VLSI 432 General Data Processor", IEEE Journal of Solid State Circuits, Vol. SC-16, No. 5: 514-521 (October 1981)
20. Bayliss, John A., et al. "The Interface Processor for the Intel VLSI 432 32-Bit Computer", IEEE Journal of Solid State Circuits, Vol. SC-16, No. 5: 522-529 (October 1981)
21. Kahn, Kevin C. and Pollack, Fred. "An Extensible Operating System for the Intel 432", COMPCON 81. New York, New York: IEEE Computer Society, 1981.
22. Zeigler, Stephen, et al. "The Intel 432 Ada Programming Environment", COMPCON 81. New York, New York: IEEE Computer Society, 1981.
23. 171954-002. Introduction to the Intel 432 Cross Development System. Santa Clara, California: Intel Corp., 1982.
24. 171870-002. Intel 432 CDS VAX Host User's Guide. Santa Clara, California: Intel Corp., 1982.
25. 172097-002. Intel 432 CDS Workstation User's Guide. Santa Clara, California: Intel Corp., 1982.
26. 9803074-01. iSBC 86/12A Single Board Computer Hardware Reference Manual. Santa Clara, California: Intel Corp., 1979.
27. Yourdon, Edward and Constantine, Larry L. Structured Design, Fundamentals of a Discipline of Computer Program and Systems Design. New York, New York: Yourdon Press, 1978.
28. DeMarco, Tom. Structured Analysis and System Specification. New York, New York: Yourdon Press, 1979.

29. MCE-AN-1001. MIL-STD-1553B LSI Chip Set Detailed Specification. Ashchurch, Tewkesbury, Gloucestershire, GB: Microcircuit Engineering Ltd., 1982.
30. BUS-25679 and BUS-27765 MIL-STD-1553B Data Bus Transformers Data Sheet. Bohemia, New York: ILC Data Device Corp., 1981.
31. BUS-63105 Series Single and Dual Redundant MIL-STD-1553B Transceivers Data Sheet. Bohemia, New York: ILC Data Device Corp., 1983.
32. 9800722-03. The 8086 Family User's Manual. Santa Clara, California: Intel Corp., 1979.
33. The TTL Data Book, Volumes 1 and 3. Dallas, Texas: Texas Instruments, 1984.
34. 143232-002. iRMX 88 Reference Manual. Santa Clara, California: Intel Corp., 1981.

VITA

Lynn M. Black was born on 10 February 1955 in Cuyahoga Falls, Ohio. She graduated from Cuyahoga Falls High School, Cuyahoga Falls, in 1973. She subsequently attended Kent State University, Kent, Ohio, graduating in 1977 with a Bachelor of Science Degree in Chemistry and a Bachelor of Arts Degree in Mathematics. She received her commission through the four year ROTC program in June 1977, and was assigned to the 4950th Test Wing at Wright-Patterson AFB, Ohio, in October 1977. She worked as a scientific analyst in the 4950th's Directorate of Flight Test Engineering until entering the AFIT Engineering Conversion Program in August 1980. After receiving her Bachelor of Electrical Engineering, as a Distinguished Graduate, in March 1982, she received a follow-on assignment at AFIT to work on her Masters Degree in Electrical Engineering. Following a short assignment, from June 1983 through December 1983, to the Aerospace Medical Research Laboratory, at Wright-Patterson AFB, she was transferred to Strategic Air Command Headquarters, Offutt AFB, Nebraska. She is currently assigned there as an electrical engineer in the Directorate of Electronic Combat, Test and Tactics, Deputy Chief of Staff for Operations. She is a member of Phi Beta Kappa, Tau Beta Pi, and Eta Kappa Nu. Capt Black is married to Maj Roie R. Black.

Permanent Address: 2334 Riverfront Drive
Cuyahoga Falls, Ohio 44221

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT SECURITY CLASSIFICATION UNCLASSIFIED			1b. RESTRICTIVE MARKINGS		
2. SECURITY CLASSIFICATION AUTHORITY			3. DISTRIBUTION/AVAILABILITY OF REPORT APPROVED FOR PUBLIC RELEASE: DISTRIBUTION UNLIMITED		
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE			5. MONITORING ORGANIZATION REPORT NUMBER(S)		
4. PERFORMING ORGANIZATION REPORT NUMBER(S) AFIT/GE/ENG/87S-1			7a. NAME OF MONITORING ORGANIZATION		
6a. NAME OF PERFORMING ORGANIZATION School of Engineering		6b. OFFICE SYMBOL (If applicable) AFIT/ENG	7b. ADDRESS (City, State, and ZIP Code)		
6c. ADDRESS (City, State, and ZIP Code) AIR FORCE INSTITUTE OF TECHNOLOGY WRIGHT-PATTERSON AFB, OH 45433			9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER		
8a. NAME OF FUNDING/SPONSORING ORGANIZATION		8b. OFFICE SYMBOL (If applicable)	10. SOURCE OF FUNDING NUMBERS		
8c. ADDRESS (City, State, and ZIP Code)		PROGRAM ELEMENT NO	PROJECT NO	TASK NO	WORK UNIT ACCESSION NO
11. TITLE (Include Security Classification) AN INVESTIGATION OF THE INTERFACING OF THE INTEL iAPX 432/670 COMPUTER SYSTEM TO THE MIL-STD-1553B MULTIPLEX DATA BUS.					
12. PERSONAL AUTHOR(S) Lynn M. Black, B.S.E.E., Capt, USAF					
13a. TYPE OF REPORT MS Thesis		13b. TIME COVERED FROM _____ TO _____		14. DATE OF REPORT (Year, Month, Day) 1987, September	
15. PAGE COUNT 329					
16. SUPPLEMENTARY NOTATION <i>Approved for public release: IAW AFR 190-1.</i> <i>Lynn M. Black 29 Oct 87</i>					
17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)		
FIELD	GROUP	SUB-GROUP			
09	02		MIL-STD-1553B Data Bus, Intel 432/670, Ada, Avionic System, iRMX 88, Multibus		
19. ABSTRACT (Continue on reverse if necessary and identify by block number)					
Chairman: Harold W. Carter, LtCol, USAF This project was an investigation into the interfacing of the Intel iAPX 432/670 computer system to the MIL-STD-1553B data bus. The project involved developing a conceptual design for a military avionic subsystem based on the 432/670 computer system, defining the specific requirements for the subsystem, and demonstrating the feasibility of implementing the interface. The conceptual design of the 432/670 Avionic Subsystem includes development, operational, and maintenance configurations for the subsystem. The feasibility of interfacing this subsystem to the 1553B data bus was proven by designing the subsystem to a level at which the design could be evaluated against the specific requirements levied upon the interface. Strict adherence to structured design techniques, and implementation of time-critical message processing functions in the hardware, resulted in relative isolation of the 432/670-related characteristics. The design presented in this project is thus readily adaptable for use in subsystems based on processors with more standard architectures.					
20. DISTRIBUTION/AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS			21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED		
22a. NAME OF RESPONSIBLE INDIVIDUAL Harold W. Carter, Lt Col, USAF			22b. TELEPHONE (Include Area Code) 513-255-3576		22c. OFFICE SYMBOL AFIT, ENG

DD Form 1473, JUN 86

Previous editions are obsolete

SECURITY CLASSIFICATION OF THIS PAGE

UNCLASSIFIED

END

DATE

FILMED

APRIL

1988

DTIC