SAE Aerospace
An SAE International Group

AEROSPACE INFORMATION REPORT

SAE AIR4903 | REV. A

Issued 1996-01
Reaffirmed 2006-07
Stabilized 2012-05

Superseding AIR4903

Pi-Bus Handbook

RATIONALE

This document has been determined to contain basic and stable technology which is not dynamic in nature.

STABILIZED NOTICE

This document has been declared "Stabilized" by the SAE AS-1A Avionic Networks Committee and will no longer be subjected to periodic reviews for currency. Users are responsible for verifying references and continued suitability of technical requirements. Newer technology may exist.

TO PLACE A DOCUMENT ORDER: **Tel: 877-606-7323 (inside USA and Canada)**
Tel: +1 724-776-4970 (outside USA)
Fax: 724-776-0790
Email: CustomerService@sae.org
SAE WEB ADDRESS: **http://www.sae.org**

SAE values your input. To provide feedback on this Technical Report, please visit
http://www.sae.org/technical/standards/AIR4903A

FOREWORD

This SAE Aerospace Information Report (AIR) was developed as a supplement to SAE AS4710 Pi-Bus standard. The handbook provides an overview of the Pi-Bus, rationale, and example use. The handbook was prepared under the direction of:

John Meyer Chairman, AS-2 Committee
Boeing Defense Group
27310 126th Place, SE
Kent, WA 98031
(206) 657-8935

Chuck Roark Chairman, AS-2C Subcommittee
Chairman, AS-2C1 Pi-Bus Working Group
Texas Instruments Defense Systems & Electronics Group
6550 Chase Oaks Boulevard, MS 8420
Plano, TX 75023

This document is a handbook to aid in the use of the Pi-Bus. SAE AS4710 documents the Pi-Bus standard, while this document provides information that is intended to help the Pi-Bus developer and/or user in using the Pi-Bus. Major emphasis is placed on including an overview of the Pi-Bus, a paragraph by paragraph rationale of SAE AS4710, and example uses of the Pi-Bus.

TABLE OF CONTENTS

1. SCOPE 3
2. REFERENCES 25
3. PI-BUS PHYSICAL LAYER RATIONALE 25
4. PI-BUS DATA LINK LAYER RATIONALE 44
5. ERROR TABLE RATIONALE 74
6. APPLICATION ARCHITECTURES/EXAMPLE APPLICATIONS 108
7. PROGRAMMER'S DESIGN GUIDE 130
8 IC DESIGN GUIDE 130
9 NOTED DISCREPANCIES IN AND CLARIFICATIONS TO SAE PI-BUS STANDARD 131

1. SCOPE:

This section defines the scope of the document, provides a brief history of the Pi-Bus, discusses key features of the Pi-Bus, and provides an overview of the operation of the Pi-Bus.

This document is a handbook intended to accompany AS4710 Pi-Bus standard. The purpose of this document is to provide information to aid users of the Pi-Bus, whether they be implementors of Pi-Bus controllers, architects of systems considering using the Pi-Bus, or programmers who must develop applications in a system which uses the Pi-Bus as the backplane communications bus. This document also provides rationale for many of the Pi-Bus requirements as defined in AS4710 and a discussion of potential enhancements that are being considered for the Pi-Bus.

The following is a mapping of major sections in this document for particular audiences:

a. Overview of Pi-Bus and its capabilities: Section 1
b. Pi-Bus rationale: Sections 3, 4, and 5
c. information for IC designers: Sections 3, 4, 5, 8, and 9
d. Information for system designers/architects: Sections 1, 6, 7, and 9
e. Information for software designer: Sections 1, 6, and 7

The following is a synopsis of the history of the Pi-Bus.

1.1 History of the Pi-Bus:

The Pi-Bus became an SAE standard on 10 May 1993. The history of the Pi-Bus and its path towards standardization is interesting in that the Pi-Bus was first defined in a specification developed for the Very High Speed Integrated Circuit (VHSIC) program, maturing of the specification became possible due to several implementations and interoperability studies, studies were then performed to increase the performance of the Pi-Bus, and finally industry standardization occurred. The remainder of this paragraph provides a brief synopsis of this history.

In 1985, IBM, Honeywell, and TRW developed the Pi-Bus specification as part of the VHSIC program. Its purpose was to provide a highly reliable multidrop backplane bus to support the communication between loosely coupled modules via message passing. The VHSIC Pi-Bus specification defined the Pi-Bus by describing its physical layer and data link layer. Westinghouse developed the first (not completely compliant) Pi-Bus implementation in 1987 as part of the Wright Laboratory VAMP program. This implementation was important in that its device-side interface, which is not specified in the Pi-Bus specification, was used as the baseline for device-side interfaces for many future Pi-Bus implementations. By coincidence in 1987, the original JIAWG platforms (A-12, ATF DEM/VAL, and LH DEM/VAL) all used the Pi-Bus as their backplane communication bus. Because of this commonality, the Pi-Bus became the standard JIAWG backplane intermodule communications bus. IBM, TI and Unisys were the primary Pi-Bus vendors for the JIAWG programs. In early 1988, through discussions within the SAE Pi-Bus Working Group and simulations performed under the Zycad-Air Force Demonstration of Avionic Module Exchangeability via Simulation (DAMES) program, it became apparent that the three

1.1 (Continued):

implementations were not completely interoperable. McDonnell Douglas Aircraft initiated a series of Pi-Bus interoperability working group meetings between IBM, TI and Unisys in the summer of 1988. The purpose of these meetings was to clarify and complete the VHSIC Phase 2 Pi-Bus Specification to insure that post 1991 Pi-Bus implementations provided by different vendors would be interoperable on the JIAWG platforms. These meetings were eventually followed by formation of the JIAWG Pi-Bus Working Group, which was the group within the JIAWG responsible for producing a JIAWG Pi-Bus Specification. The JIAWG Pi-Bus Working Group officially began meeting in April 1989. Also in 1989, the Navy awarded a contract to IBM to look at performance enhancements for the Pi-Bus. One of these, a more efficient type of message called a datagram, was included in AS4710. Also, in this timeframe, DELCO was awarded a contract to develop a 32-bit Pi-Bus for the F-22 program.

The SAE AS-2 Committee had been used as the industry user's group for the Pi-Bus since its inception in VHSIC Phase 2. The JIAWG Pi-Bus Working Group met in concert with the SAE AS-2 Committee. This group developed a revised version of the VHSIC Pi-Bus Specification called the JIAWG Pi-Bus Specification. The updates included specification ambiguity cleanup, detailed specification of error management, and the additions of new features required by JIAWG for a 32-bit datagram message. It was decided in 1989 that the SAE should attempt to make the Pi-Bus into an SAE standard so that the Pi-Bus could be truly an open systems interface standard. During this process, the JIAWG standard was updated to be technically the same as the SAE draft standard in order that the JIAWG could reference AS4710 instead of the JIAWG Pi-Bus Specification. This process completed 10 May 1993, with the Pi-Bus becoming SAE standard AS4710. The JIAWG now references AS4710. Associated with the development of AS4710, the SAE and DoD worked together to form a Pi-Bus Advisory Board. The Advisory Board consists of a single member from each service and the chairman of the SAE Pi-Bus Working Group. The purpose of the Advisory Board is to provide a formal communication mechanism between the DoD and SAE for guidance on revisions of AS4710, handbooks, and user group activities.

1.2 Pi-Bus Key Features:

This section provides a brief discussion of Pi-Bus key features. The view of these features is that of the user. This section is organized as follows:

a. Section 1.3.1 discusses general capabilities of the Pi-Bus.
b. Section 1.3.2 describes Pi-Bus message passing features.
c. Section 1.3.3 provides an overview of a typical Pi-Bus software interface.
d. Section 1.3.4 lists several programming paradigms using the Pi-Bus.

1.2.1 General Capabilities: As noted in Figure 1, the Pi-Bus can be either a 16-bit or 32-bit parallel backplane bus. The Pi-Bus is a loosely coupled, message passing bus. Up to 32 modules can communicate over a single Pi-Bus. A 16-bit Pi-Bus is error detecting (ED), while a 32-bit Pi-Bus is error correcting (EC). [Note this is a change from the VHSIC Pi-Bus Specification in which a 16 or 32-bit Pi-Bus could be either error detecting or error correcting.] The Pi-Bus uses a synchronous clock which is specified to run at a maximum of 12.5 MHz.

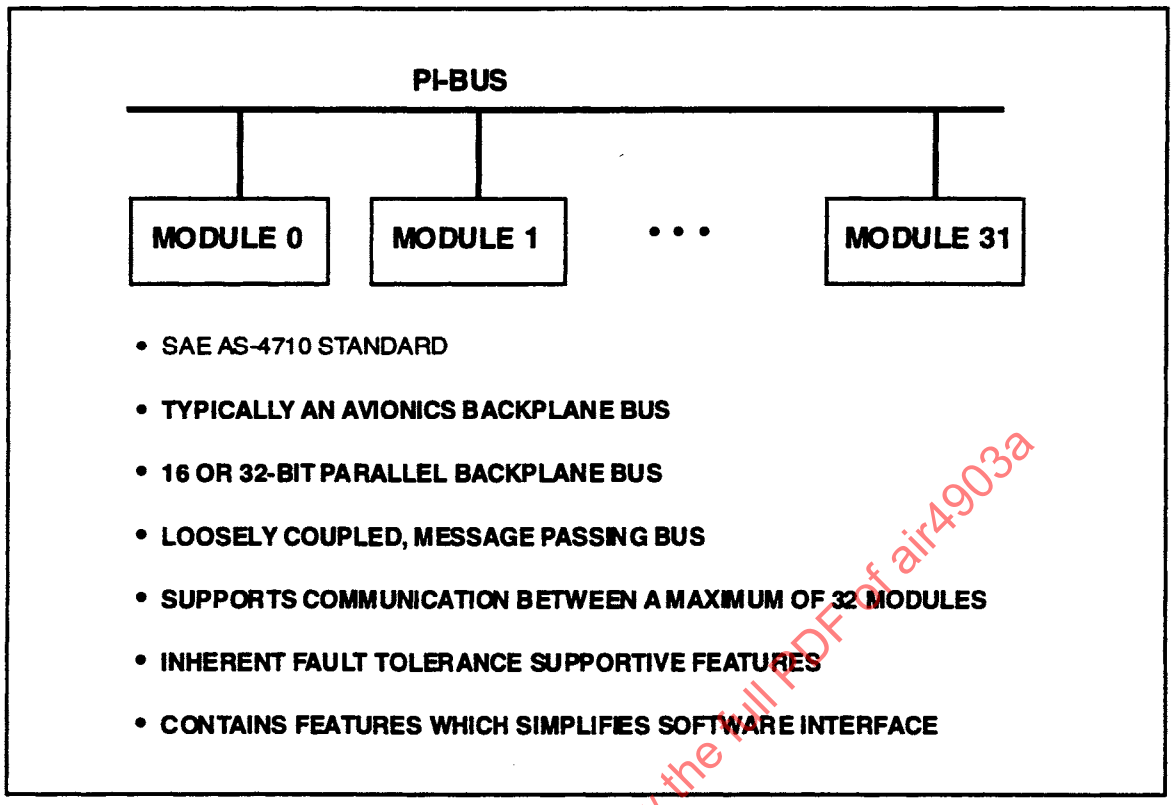

FIGURE 1 - Brief Overview of Pi-Bus

1.2.1 (Continued):

The module connection to the Pi-Bus is a dot-or connection to negatively defined signals. Therefore, any module can assert a logical one. A logical zero is present only if no module is asserting a one. Generally, all modules are constantly presenting signals to the Pi-Bus. However, they are generally logical zeroes.

A Pi-Bus can be in one of two operational states, active or inactive. It is in the inactive state when no module is asserting signals, i.e., no message activity. The Pi-Bus is in the active state when one or more modules are asserting signals, i.e., there is a vie or message activity.

A module consists of a Device and a Bus Interface Unit (BIU) as shown in Figure 2. The Master portion of a BIU can obtain control of the Pi-Bus and initiate messages. The Slave portion responds to Pi-Bus messages addressed to it. There are two types of modules, Master/Slave modules and Slave Only modules. Only the Master/Slave modules can initiate bus messages. Both module types can respond to messages. The Master portion can initiate a message which addresses its companion Slave portion.

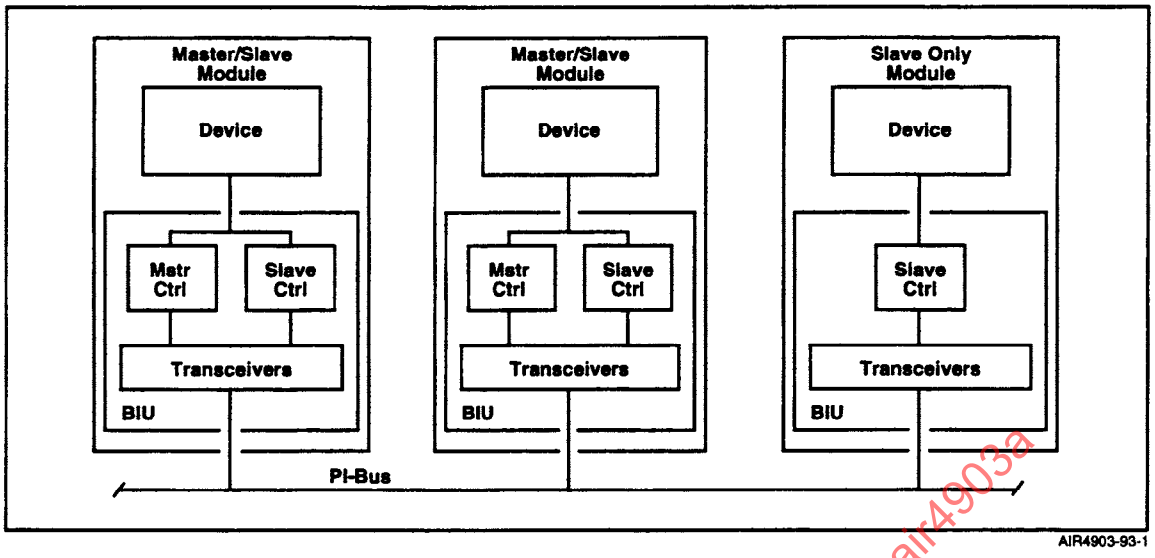

FIGURE 2 - High Level View of Pi-Bus

1.2.1 (Continued):

The Pi-Bus is an open systems standard which has several features which are beneficial for avionics computers. Some of these include the following:

a. fault tolerance support
b. backplane mission time distribution source
c. optimized message passing
d. real-time programming support
e. logical slave identifiers and labels
f. either centralized or distributed initialization
g. security support
h. simplified software interface.

The following subparagraphs describe each of these items in further detail.

1.2.1.1 Fault Tolerance Support: One of the most distinctive features of the Pi-Bus is its support for fault tolerance. This section discusses how the Pi-Bus supports fault tolerance.

The Pi-Bus is inherently supportive of module level fault containment since it is a loosely coupled, message passing bus. This is in contrast to the poor fault containment provided by the typical tightly coupled buses which access memory a word (or byte or multiple bytes) at a time, such as VME or Futurebus+. This follows due to Pi-Bus features such as hardware supported intermodule communication containment boundaries, an error management protocol which supports detection and isolation of contaminated memory, the ability for software to control access to its memory, and explicit software control of intermodule communication.

1.2.1.1 (Continued):

The Pi-Bus provides detection of 100% of single line faults on both data and control lines. In addition, the 32-bit Pi-Bus provides single line error correction on both data and control lines. This is in contrast to most buses which either provide no error detection or if they do, it typically applies only to the data lines.

For a single backplane which desires to support fault tolerance, either a single 32-bit error correcting Pi-Bus is used or dual 16-bit error detecting Pi-Buses are used. The F-22 CIP uses a 32-bit error correcting Pi-Bus. In the 16-bit case, typically one bus is used as a primary, while the second Pi-Bus is used as a redundant backup in case of failure of the primary. Such is the case for the ATF YF-22 Demonstration/Validation Mission Display Processor, the F-16 Main Mission Computer (MMC), the F-22 Vehicle Management System, Comanche Helicopter, and F-15 Computer upgrade. For the 16-bit case, typically a single Pi-Bus Interface Unit (PIU) is used with two sets of transceivers since the typical failure is expected in the connector pins instead of the electrical circuitry. The PIU is commanded by software to select one of the two. In the 32-bit case, only one bus is typically used. Such is the case with the F-22.

In order to minimize the chance of a single module "bringing down" a Pi-Bus, there is no centralized control -- the protocol uses a distributed Vie for gaining Mastership of the bus. In addition, there is an absolute tenure time-out to bound a master's tenure. If a PIU or its transceivers are faulted on a module, the module software may disable transceivers to remove the PIU from one or both buses. In addition, if a TM-Bus is used in the system (as is suggested in AS4710), then a module's transceivers may be disabled from off module via a TM-Bus command.

The Pi-Bus requires a central, synchronous clock. In order to prevent the clock from being a single point of failure, many implementations implement dual clocks. Use of a synchronous clock allows the latching of bus signals after the "wire-or-glitch" which causes problems in other buses that use BTL levels. The synchronous protocol also allows the use of simulation to prove the validity of a design while an asynchronous design can never be completely proven.

The Pi-Bus is compact -- it has a relatively simple protocol, small number of pins, and typically is a single chip implementation. This implies the typical Pi-Bus implementation is robust.

Finally, the Pi-Bus supports hardware acknowledgments not only on singlecast messages but also on multicast/broadcast messages. This not only allows for quicker determination of the success of a message, but reduces bus traffic by not requiring software to transfer acknowledgments to the sender. This allows operating system software to implement a retry protocol without requiring either application interaction or acknowledgment messages being sent on the bus. The hardware acknowledgments also provide additional information which can be used for fault detection/isolation or determinations such as a slave or slave's label was busy when a message transfer was attempted. (What is meant by slave label is discussed in 1.4.1.5.)

1.2.1.2 Backplane Mission Time Distribution: Consistent time distribution is a common problem for distributed systems. The Pi-Bus provides hardware support for system time distribution. A PIU implements a 48-bit system timer register with a 1 MHz clock (i.e., with 1 microsecond resolution). A bus interface message is used for transferring system time values directly between the system timer registers of the Master and Slave(s). When one module transmits system time to another module, the system timers are to be synchronized within 12.5*4/(Pi-Bus clock frequency) microseconds. Thus, if a 12.5 MHz Pi-Bus is being used, at the end of distributing system time, the system time of the Master and Slaves are guaranteed to be synchronized within 4 microseconds.

The following are example uses of system time distribution.

a. If it is important for applications executing on modules within a backplane to be synchronized with system time outside of the backplane, the following method has been used. A module within the backplane (e.g., High Speed Data Bus (HSDB) or 1553B interface modules) has the capability to accurately receive system time external to the backplane. On receiving a system time update, the special module broadcasts the system time to other modules on the backplane. Depending on the accuracy required, the special module may require hardware support for loading the Pi-Bus system timer upon receiving a mission time update from its external source.

b. If applications within a backplane only need to be synchronized among themselves, i.e., not with applications external from the backplane, then there are two general approaches for system time distribution:

   (1) A master time module can be chosen which periodically distributes its system time to the other modules in the backplane. Protocols can be chosen for changing the master time module. For example, this might occur if unacceptable variations are seen in its time distribution.

   (2) A master time module can be chosen whose system timer is periodically read by all modules. Protocols can be chosen for changing the master time module.

1.2.1.3 Optimized Message Passing: The Pi-Bus was designed with message passing as a basic requirement. Because of this, the Pi-Bus includes an optimized message passing protocol which supports fault tolerance and real-time programming. Some of these optimized message passing features are:

a. low overhead message passing protocol
b. message priorities
c. variable length tenure
d. message suspend/resume
e. several types of messages.

1.2.1.3 (Continued):

A PIU is a direct memory access (DMA) device which allows Pi-Bus transfers to occur without causing a scalar processor to stall in either the Master or Slave(s) modules. This not only increases performance but also makes data processor execution more deterministic. This is in contrast to processors using a tightly coupled bus in which the data processor either acts as the DMA controller or is stalled on read accesses to memory.

The typical PIU supports the sending of "chained" messages so that a Master may use a single command to invoke the PIU to transfer a set of messages without requiring additional processor interaction during the transfer of the message(s). In addition, Slave processors are not involved with a message transfer while it is in progress -- Slaves may be notified of the arrival of a message by an interrupt with information about the message queued for OS/application examination. Figure 3 provides a pictorial summary of this discussion for label addressed block message transfers. Reference 1.2.1.5, 1.2.3.2, 1.2.4.3 and 4.2 for a discussion of label addressing.

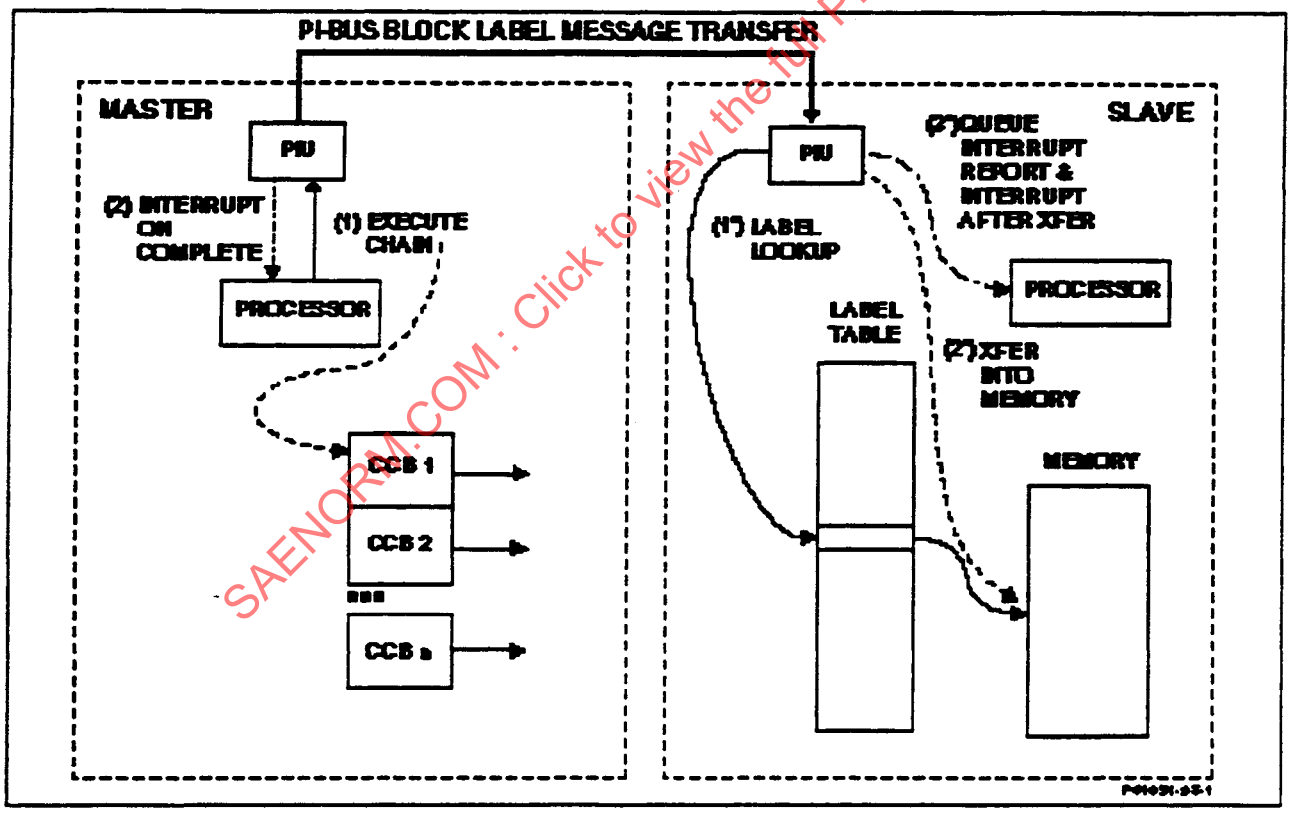

FIGURE 3 - Typical Label Block Message Transfer

1.2.1.4 Real-Time Programming Support: Special provisions were included in the Pi-Bus to support real-time programming based on paradigms such as variations of the Rate Monotonic algorithm and priority based algorithms. The basic requirements for supporting such paradigms is the inclusion of message priorities and the ability to pre-empt a lower priority message by a higher priority message. The Pi-Bus includes message priorities and the ability to suspend/resume messages.

A message priority consists of two components: a 5 bit module ID (MID) and a 7 bit logical priority. During a Vie sequence, the message priority is made up of the concatenation of the logical priority and MID. The use of the MID allows for a deterministic, unique selection of a Master during a Vie sequence. The logical priority alone is used for determining whether a message should be allowed to be suspended. The logical priority is used by an application to denote the importance of a message. Typically, if the application software designates two messages in different modules as having the same logical priority, it considers them to have the same priority regardless of what hardware module the software is on. In particular, if a message is currently being transferred, it will not be suspended because another module with a more urgent MID desires to transfer a message of the same logical priority.

There are two timers associated with a PIU used in the message suspension protocol. These timers are software configurable. These timers are Vie Interval A Register and Vie Interval B Register, which are 16 and 8 bit registers, respectively, expressed in bus cycles. The timers are used to provide a maximum time bound, expressed in bus cycles, for denoting when a higher priority message may begin being transferred on the bus. The first timer is used to denote the amount of time a Master may continue to send messages once a request has been made for it to relinquish the bus due to another module wishing to send a higher priority message. If the Master has not completed sending a message (chain) by the time Vie Interval A Register expires, the Master must relinquish the bus by suspending the current message by the time Vie Interval B Register expires or it will be aborted. Software configures the Vie Interval A and Vie Interval B Registers appropriately to guarantee deterministic message deliveries of critical messages for its application.

1.2.1.5 Logical Slave Identifiers and Labels: Within a message a Slave is denoted by a slave identifier (ID). As depicted in Figure 4, the Slave IDs are of three types: 32 physical IDs denoted by the MIDs, broadcast ID, and 223 logical IDs. The MID can be used to identify a Slave for a message when there is a single Slave and the Master knows the physical ID of the Slave. The broadcast ID is used to send a message to all modules on a backplane. Logical IDs are used to "logically" denote one or more Slaves for message transmission. A PIU can be configured dynamically via either a bus interface message or via on-module software commands to respond to any number of the 223 logical IDs. A Master is not required to know the physical address of a Slave module or how many modules with which it is communicating if logical IDs are used. Logical IDs are suggested for ease of reconfiguring software. For example, if physical IDs were being used and an application which was originally allocated to one module was moved to a different module, software within the Master would require modification to support this change. On the other hand, if logical IDs were used, no change would be required in software.

1.2.1.5 (Continued):

Block Messages which use label addressing were included in AS4710 to extend the concept of logical IDs to a greater number and to provide Slave controlled access to Slave memory. (For a discussion of Block Messages, see 1.2.2.1.) A label extends the logical ID in the following way. Suppose a label block message is sent to a module, using either logical or physical IDs. If the corresponding label in the Slave module is not activated, then the Slave module ceases participation in the bus sequence -- just as if the module was addressed by a logical ID not activated for the module. If the corresponding label in the module is activated (and not busy), the message transfer occurs.

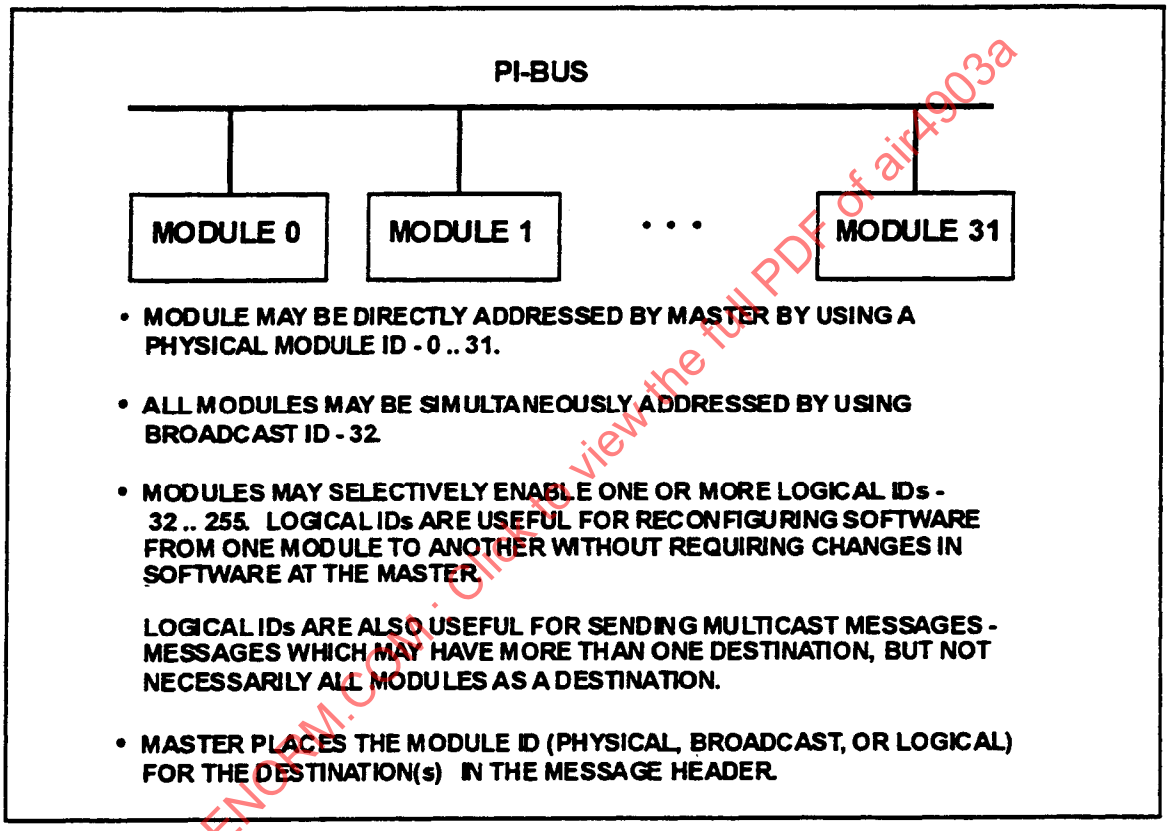

FIGURE 4 - Addressing a Module

The use of labels allows the Slave to designate the physical address in Slave memory to either read or write the message. Therefore, changes in Slave memory allocation do not affect software executing in a Master, since the Master is not required to know the physical address. In addition, if direct memory addressed messages are not used, this allows a Slave to protect its memory from arbitrary access from off module. Most current PIU implementations allow direct memory addressed messages to be disabled.

1.2.1.6 Centralized and Distributed Initialization: AS4710 defines the Pi-Bus data link registers. These registers specify configuration parameters for a PIU. For example, the Vie Interval A and B timers and logical IDs registered for a module are defined in the data link register set. The bus interface message allows a centralized approach to Pi-Bus initialization. This is accomplished by allowing a single module within a backplane to initialize the data link registers for all modules within the backplane by using bus interface messages. A decentralized approach is also allowed by having a module initialize its data link registers by sending a bus interface message to itself or by providing a device-side interface to the data link registers. It is generally felt that distributed initialization is preferred for modules which are capable of Master/Slave operation (and thus typically have a processor on the module), while centralized control should only be used for initializing modules with Slave only capabilities (and thus typically do not have a processor on the module). Allowing distributed initialization for Master/Slave capable modules avoids potential synchronization problems, avoids problems due to the loss of the control module, and allows initialization to occur in parallel.

1.2.1.7 Security Support: Most PIU implementations contain several features which support data security.

As noted in 1.2.1.5, label block messages allow a module to restrict access to its memory. It is suggested that except for possibly accessing Slave only modules, direct addressed block messages not be used. Most PIU implementations provide the capability to disable the acceptance of direct addressed block messages, preventing application software from either accidentally or maliciously accessing a module's memory directly.

In most current implementations, a label table entry includes an entry which states the maximum buffer size allowed for a message sent/received from that label. In addition, the maximum label allowed and maximum event label allowed may be specified to further reduce the access to a Slave module. These features help limit external access to a module's memory.

The block message and datagram message are allowed to have an extended header -- six additional 16-bit words of application dependent information that is passed with the message. This information could contain information which could be used to support security.

1.2.1.8 Simplified Software Interface: An important feature supported by current PIUs is the device-side interface simplifies the software interface to the Pi-Bus. These features include data structures used for transmitting messages and receiving messages, a robust error reporting interface, and the decoupling of the Pi-Bus and software in the sense that a PIU requires little interaction with software to perform message transactions. Section 1.2.3 discusses several of the key software interface features provided in most PIU implementations.

1.2.2 Message Passing Features: The Pi-Bus supports several types of messages. These include:

a. Block Messages, including both direct and label addressing
b. Parameter Write, including both logical and event filter
c. Bus Interface, including both data link register access and system timer
d. Datagram Messages, including logical addressing only

In addition, Block Messages and Datagram Messages support transfers using either short or extended headers. This section provides a synopsis on the use of these message types.

1.2.2.1 Block Messages: A block message may transfer between 1 and 65 536 contiguous words of physical memory. (The word size is 16-bits for a 16-bit Pi-Bus and 32-bits for a 32-bit Pi-Bus.) A block message may be either a read or a write. A read will transfer data from a Slave to the Master. A write will transfer data from the Master to one or more Slaves. Due to the potential size of block messages, block messages are suspendable as noted in 1.2.1.4. However, a module may be configured to not allow the suspension of block messages. If a module is configured not to support the suspension of block messages, then a message which would have been otherwise suspended during a suspend sequence would be aborted at the expiration of the appropriate Vie Interval B Register.

Two modes of addressing are allowed in block messages: direct addressing and label addressing. Direct addressing is interpreted by the PIU on the Slave module. Some PIUs address a module's memory as though the address were a physical address, while others address a module's memory as though it were a logical address. Sending block messages using direct addressing is not generally suggested for communicating with other modules containing a data processor as discussed in 1.2.1.5.

Label addressing, as discussed in 1.3.1.5, supports logical addressing of a slave module. This is the generally accepted addressing approach suggested for block messages.

1.2.2.2 Parameter Write Messages: The typical Pi-Bus message sequence includes a header sequence followed by a data sequence. The header sequence is used to identify the Slave(s), identify the type of message to be sent, and to communicate message type specific information (if there is any), such as label for block label messages and datagram messages, to the Slave(s). If there is data associated with the message, the data is then transferred during the data sequence. The Parameter Write message is an optimized message in the sense that the entire message is transferred during the header sequence, i.e., all data associated with the message is included in the message header.

There are two types of parameter write messages: logical and event filter. The logical parameter write message allows for the transfer of three 16-bit words to a Slave. Upon reception by a Slave, the message header, which includes the three data words, is queued for access by the Slave processor. In order for application software to make use of the data, a subset of the data must contain information used by application code to denote what the data is and possibly who it is for. In contrast, the label used in a block message using label addressing can be used to denote this information.

1.2.2.2 (Continued):

The event filter parameter write message provides an efficient mechanism for the signaling of the occurrence of up to 16 events. This can be used to implement synchronization primitives such as the signal of a barrier. An overview of the event filter parameter write is presented in Figure 5. An event filter parameter write message contains (1) a label which denotes one of a possible 4096 different event labels and (2) event flags which denote between 1 and 16 events. There is an event label table on a Slave module indexed by the label contained in the event filter parameter write messages. Each event label table entry contains a 16-bit event mask which denotes the event set associated with the label. Each event label table entry also contains a 16-bit event flags word which is OR'ed with the event flags entry of a parameter write message using a label associated with the label table entry. The event flag word shows how many different events within the event set have occurred. When all the events as defined by the event mask have occurred, the Slave is signaled. This signaling occurs by queuing an event report which contains among other things the label associated with the event and interrupting the Slave processor.

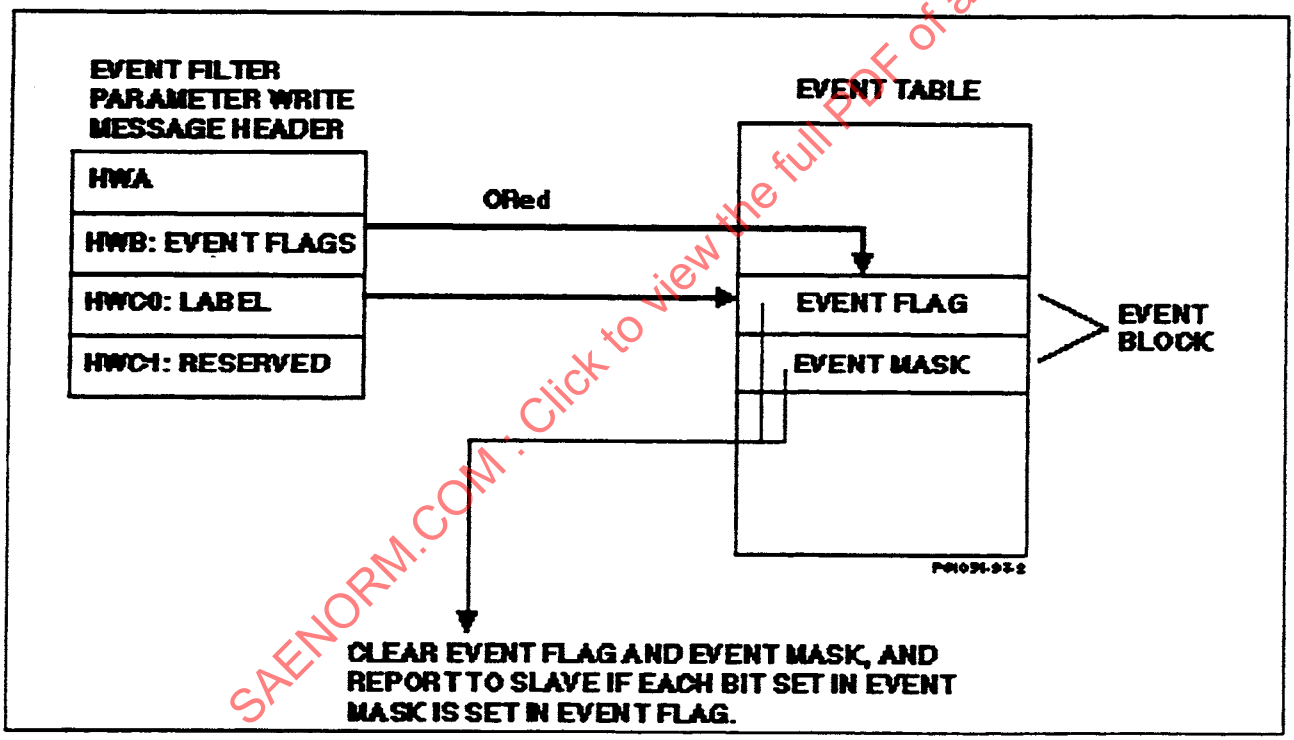

FIGURE 5 - Event Filter Parameter Write

The event filter parameter write message eliminates the need for the processor to be interrupted more than once and software processing required to denote the occurrence of a set of events. If an event set contained N events and the event filter mechanism was not used, then at least N interrupts (and more if some events were repeated before the entire set of events had occurred) would be required along with software processing to determine the entire event set had occurred.

1.2.2.3 Bus Interface Messages: There are two types of Bus Interface Messages: data link register access and system time distribution. The Bus Interface Message which accesses the data link register space of a Slave is used to either read or write a Slave's data link register space. This was discussed in 1.2.1.6.

The Bus Interface Message used for system time distribution is used for transferring the system time value within one PIU to another. This was discussed in 1.2.1.2.

1.2.2.4 Datagram Messages: There are two types of datagram messages: nonacknowledged datagram and acknowledged datagram. Nonacknowledged messages are intended for a send and forget type system where no message acknowledgment from the slave(s) is needed. Acknowledge datagram messages are used in place of block messages when header acknowledge information is not needed, but a data acknowledge is. All datagram messages are multicast.

Both datagram messages provide higher potential throughput than block messages. Because the header acknowledge has been removed, selected slaves can begin buffering data while processing the label information. If any label problems are detected, the slave disconnects from the message and flushes it's data buffer. If all label checks pass, then memory transfer from the slave data buffer to the module memory can begin. Because the label checks can be done in parallel with the Pi-Bus data transfer, datagrams can have a much higher throughput than block messages. The amount of throughput improvement is greatly dependent on PIU design. The slave must have a data buffer large enough to hold all of the Pi-Bus data that comes into the slave, while the label checks are being performed. It is desirable to have a Slave DMA throughput that is greater than the Pi-Bus data rate.

1.2.2.5 Short and Extended Headers: Messages are typically sent using short headers. However, block and datagram messages may also be sent using extended headers. The extended headers contain a short header plus an additional six 16-bit words which are application dependent. One use of extended headers is to pass operating system specific information related to the message without requiring a second message transfer. For example, if a message dealt with file I/O, the extended header could include specific information about the file operation in the same message which transfers file data.

1.2.3 Software Interface: One of the key characteristics of all current PIU implementations is they all provide simplified software interfaces for Pi-Bus transfers and PIU control. This section gives examples of the types of software interface features included in existing implementations. The features discussed are categorized into Master and Slave interfaces. Figure 6 depicts the two most significant data structures used by software to communicate with a PIU. The details given are typical of several interfaces.

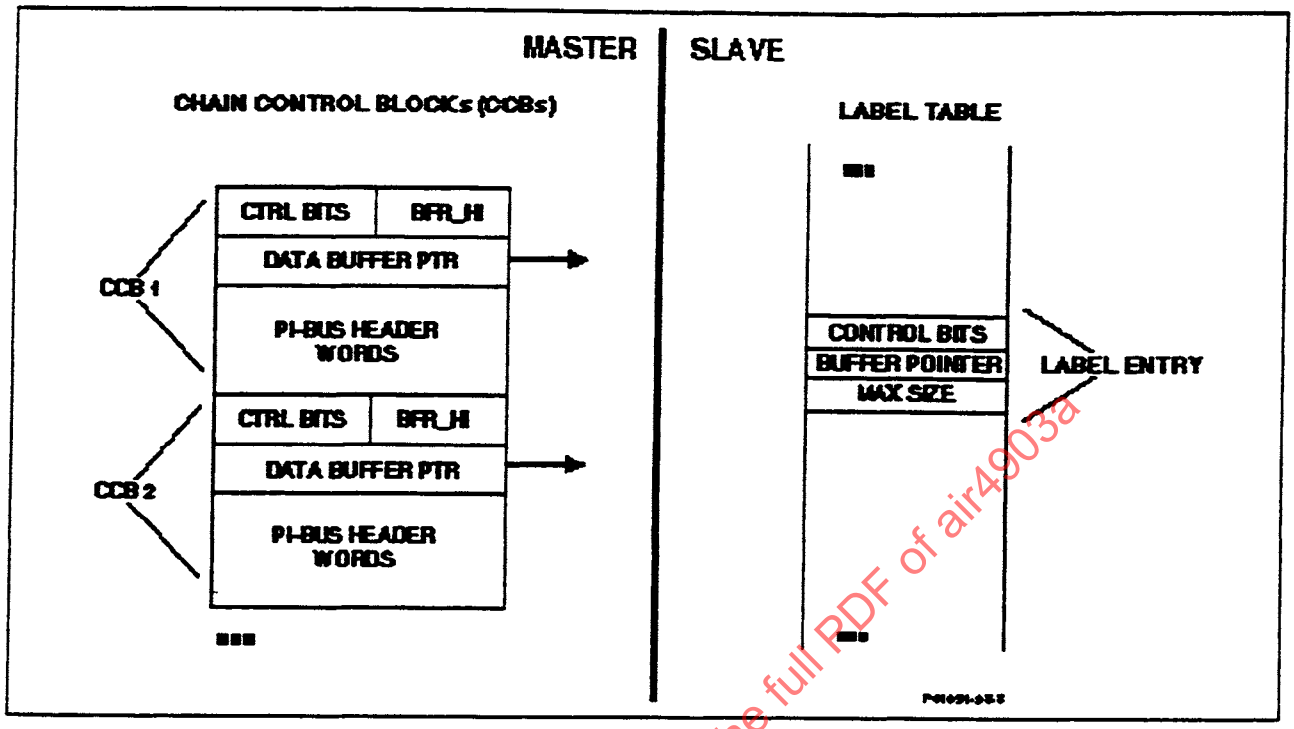

FIGURE 6 - Example Master and Slave Data Structures

1.2.3.1 Master Software Interfaces:

1.2.3.1.1 Chain Control Blocks: The basic structure used for controlling a PIU as a Master is the chain control block (CCB). (Reference Figure 6). The typical CCB denotes a Pi-Bus message to be sent by this module. CCBs may be chained together to allow multiple CCBs to be invoked by a single command from a processor. CCBs in a chain are executed sequentially. As will be noted below, in the typical implementation the separate CCBs in a CCB chain are not required to be contiguous in memory.

In several implementations, two types of CCB chains may be invoked simultaneously: normal chains and priority chains. Typically, a normal chain is commanded to a PIU. However, if software determines it needs to transmit higher priority messages while a normal chain is executing, a priority chain may be invoked which will suspend the current chain so that the priority chain can be processed.

A processor may also command the PIU to abort a chain that is currently executing .

1.2.3.1.1 (Continued):

A CCB contains control flags used for specifying actions to be performed. Some of these actions include whether the CCB is a NOP (no operation is to be taken for the CCB), whether the CCB is the last in the chain (in which case an end of chain report will be posted and the Master processor interrupted), and whether the CCB is a jump CCB (in which case the CCB denotes the location of the next CCB to execute). A jump CCB allows the CCBs comprising a CCB chain not to be contiguous. If the CCB control flags do not designate the CCB as a NOP or jump CCB, the CCB contains the message headers for a Pi-Bus message to transfer and if appropriate a location in memory for reading the data for message writes or for writing the data for message reads.

1.2.3.1.2 Master Interrupt Interface: For typical implementations, an interrupt is generated to a Pi-Bus Master due to the completion of a CCB chain, the completion of a CCB "marked" to cause an interrupt on completion, and the occurrence of an error.

The typical implementation also includes a data structure or set of data structures which contains information used by software on a Master. Examples of this information include:

a. Information on the CCB that was last executing, such as the CCB address, received header acknowledge words and received data acknowledge words (where appropriate).

b. For interrupts generated due to errors, information is provided describing the errors.

c. Information on the status of the message such as the Pi-Bus message type, whether the message was a multicast, whether the message was a write, if the header acknowledge words are valid, if the data acknowledge words are valid, whether there was a no response error (i.e., no Slaves responded), and whether any Master Pi-Bus errors were detected.

There is typically a PIU register which is accessed upon the occurrence of an interrupt to allow software to determine the cause of the interrupt.

1.2.3.2 Slave Software Interfaces:

1.2.3.2.1 Label Table: The basic data structure used for Slave processing is the block message label table (reference Figure 6) and the event filter label table (reference Figure 5.) These tables define the Slave interface to label addressed block messages and event filter parameter write messages, respectively.

1.2.3.2.1.1 Block Message Label Table: Label block messages provide the most robust type of Pi-Bus message. Not only does the label logically denote the address where to read/write the message but it is also used as an index into a table whose entries allow various paradigms to be used on a Slave for controlling block messages addressed to that label. For example in some implementations, a label entry can denote whether double buffering or single buffering is to be used and whether an interrupt should be generated after the message is received.

1.2.3.2.1.1 (Continued):

AS4710 allows for 64K labels. Typical implementations restrict the number of labels. A label table is contiguous in Slave memory and the maximum number of labels is software configurable. In some implementations there is also a minimum label designator. If a label addressed message, whose label is greater than the maximum label size allowed by a module, is attempted, the Slave module will not participate in the message transfer. The use of the maximum (and minimum) label designator(s) allows for conservation of memory since a label table of only the size bounded by these designators is needed.

For several implementations, a label table entry contains a set of control flags, buffer addresses, and a maximum label buffer size allowed. The control flags are typically used to denote if a label is enabled, if a label is busy, to denote the buffer to be accessed (if the implementation supports multiple buffers per label), denote the suspend status of a label (i.e., whether a message that was being sent to the label is currently suspended), and to denote whether an interrupt is to occur upon completion of a transfer to the label. The maximum label buffer size is used to ensure that the label addressed message does not read/write more data than the Slave intends. If a label addressed message attempts to read/write more data than the maximum label buffer size for the Slave's module label, the module will not participate in the transfer.

For some implementations, there is a single Test & Set Register associated with the PIU used as a semaphore for shared memory access to the label table. This is required since the Pi-Bus and software may be attempting to access label table entries simultaneously.

1.2.3.2.1.2 Event Filter Label Table: The event filter label table is discussed in 1.2.2.2.

1.2.3.2.2 Slave Interrupt Interface: Interrupts can be generated to a Pi-Bus Slave for several reasons; for example, due to the occurrence of an error, due to the Slave Receive List (SRL) being full, and due to receiving a message which indicates that the processor should be notified.

An SRL entry contains information regarding a received message. This information typically includes the header words received, the transmitted header acknowledge word 0 and data acknowledge word 0, the bus priority during the message transfer, Slave detected Pi-Bus errors, and a code stating the reason for the SRL entry and the validity of the stored header words and acknowledge words.

Upon the occurrence of an interrupt, the processor looks at a PIU interrupt cause register(s) to determine if the interrupt was due to an SRL entry being posted, SRL being full, or some other reason.

1.2.3.2.3 Control/Configuration Registers: In order to allow a module to configure itself to accept only the types of messages desired by an application, the typical PIU contains a Slave configuration register(s). This register denotes whether a Slave will accept direct addressed block message reads, direct addressed block message writes, label addressed block message reads, label addressed block message writes, parameter write event filter, and parameter write logical. In addition, it also allows the Slave to denote whether it supports suspension of message reads or suspension of message writes.

A typical use of this register would be to disallow direct memory block message access to a module for reasons as discussed in 1.3.1.5.

1.2.4 Programming Paradigms: There are many programming paradigms which the Pi-Bus supports. This section discusses several of these.

1.2.4.1 Push/Pull Paradigms: Since Pi-Bus block messages can either read data from a Slave or Write data to a Slave, both push and pull mechanisms can be efficiently supported. A push mechanism is a protocol in which the creator of the data "pushes", i.e., sends, the data to users of the data. In contrast a pull mechanism is a protocol in which the users of the data "pull", i.e., retrieve, the data only when they want to use it. A pull protocol can be implemented with Pi-Bus reads. A pull protocol can also be implemented with the user sending a request message to the creator of the data, followed by the creator sending the requested data. The second pull approach requires more Pi-Bus traffic than the first approach.

1.2.4.2 Direct Memory Addressing: The Pi-Bus allows direct addressed block messages. As noted in 1.2.1.5 and 1.2.1.7, direct memory addressing is not considered a safe mechanism for accessing Slave memory. Therefore, modules should have the capability to disallow their reception as a Slave of direct addressed block messages. However, a system can use direct addressed block messages to treat the memory within modules resident on a backplane as a global memory space. In this case, a global memory address would consist of a two part address in the form (MID, module local address), where MID denotes the module ID of the module in which the memory being addressed is resident.

NOTE: Logical IDs could be used in place of MIDs if appropriate logical ID assignments to modules were made.)

One common use of direct addressing is to use it during startup for downloading, reading health results, etc. and then having modules disable direct addressing during normal operation.

1.2.4.3 Label Addressing: As noted in the discussion of block label messages in 1.2.1.5 and 1.2.3.2.1.1 label addressing allows

a. Logical addressing - the Master does not need to know the number or the physical address (i.e., either the module IDs or physical memory address) of the Slaves.

b. The Slave can control access to its memory if label addressing is used.

c. Single buffering or double buffering schemes as well as circular queues can be used.

1.2.4.3 (Continued):

d. Label block messages can either cause or not cause an interrupt at the Slave on arrival - thus, blocking, polling, and application defined synchronous schemes (e.g., time based) are supported.

When label addressing is used, the application software must have some means to associate the message with the appropriate application. One paradigm is to use the label to denote what the message is or who the destination of the message is. Another paradigm (which may be combined with the previous one) is to embed data within the message to provide the desired information.

The event filter also uses (event) labels. As noted in 1.2.2.2, the event filter allows an efficient implementation of a barrier protocol. If it is desired that a Slave should be signaled on the occurrence of n different events occurring (at least once), the event filter implements this in hardware. This reduces the number of processor interrupts and software processing to implement a barrier synchronization protocol.

1.2.4.4 Multicast: Multicast can be efficiently used for sending the same message to multiple Slaves. Multicasts messages are acknowledged messages for all but unacknowledged datagrams. A typical use of multicast messages are periodic health messages exchanged between modules within a backplane. Care must be taken when considering the use of multicast since the software protocol required for error recovery can be complicated when a subset of the Slaves successfully received a transfer.

1.2.4.5 Logical Versus Physical Memory: Some PIUs treat module memory as physical memory -- they do not view memory logically as the processor does. This is common for systems in which caches are used. If data to be transferred is to originate or end up in cached memory, the following types of protocols must be considered:

a. On sending, the data must either be copied to non-cached memory or (if write through caches are not being used) flushed into physical memory.

b. On reception, the data must be copied from the incoming buffer to its final memory destination or the cache must be invalidated (without flushing to physical memory) prior to the use of the data.

A typical protocol used in such systems is (1) copy data to be sent into an OS buffer prior to transfer and (2) after the transfer completes, copy data from an OS buffer which received the data into an application buffer.

If cached memory is not being used or if all I/O is accessed in non-cached memory, then this discussion can be ignored.

1.2.4.6 Extensibility: The Pi-Bus allows up to 32 modules on the same backplane. Often systems are designed with additional slots for growth modules. The use of logical IDs and labels allows for additional modules (and software) to be added possibly without affecting software in existing modules. The point being made here is if logical IDs and labels are used, software can be reconfigured to execute on different modules without modifications to software. Serious consideration should be given to using logical IDs and labels when designing a system.

1.2.4.7 Consistent Mission Time: As noted in 1.2.1.2, the Pi-Bus provides hardware support for consistent system time distribution within the backplane.

1.2.4.8 Application Access Versus OS Access to the PIU: One of the major issues to consider when designing the software for a system using Pi-Bus is what access is going to be allowed to a PIU.

A common approach is to assume that an operating system (OS) exists between the application software and the Pi-Bus. From the application programmer's interface point of view, he is not aware of the type of bus being used. Only the OS directly interfaces with the PIU. The OS accepts requests to send and receive messages from the application. Applications typically pass the data or a pointer to the data to the OS for send operations. The OS takes care of building CCBs and handles the CCB complete interrupt, unblocking the caller, if the send was a blocking operation. Typically for receive operations, a received message generates an interrupt which is handled by the OS. The OS either passes back a pointer to an OS allocated buffer containing the data or returns the data into a buffer allocated by the applications.

Other approaches provide a mixture of using an OS and allowing applications some knowledge of the PIU. The following are two alternatives to consider - but note the OS always handles direct interactions with the PIU, i.e., fields all PIU interrupts and initiates all commands to the PIU.

a. For send processing, the applications build CCB chains and passes them to the OS for sending.

b. For receiving messages, the applications interact directly with the label table. It is up to the application to change label table entries - not the OS. If an interrupt is generated due to the arrival of a message, the OS simply passes the label and message size (and maybe the remainder of the message report) to the application.

1.2.5 Summary: This section discussed the use of a Pi-Bus from the software/programmer's point-of-view. The Pi-Bus and Pi-Bus controllers provide a very robust intermodule communication bus which supports fault tolerance, provides a backplane mission time distribution source, provides optimized message passing, supports modern day real-time programming techniques, includes logical slave identifiers and labels, supports either centralized or distributed initialization, includes data security supportive features, and provide an simplified software interface.

1.3 Pi-Bus Operation Overview:

This sections provides an overview of Pi-Bus message control and a typical implementation's data flow.

1.3.1 Pi-Bus Message Control: Once a module obtains control of the bus, it becomes the bus Master and starts its bus tenure. The Master sources the Cycle Type signals of the Pi-Bus as shown in Figure 7. The state machine in the Master drives these signals. The message proceeds through the following states:

a. H0 (Header 0 Word)
b. H (Other Header Words)
c. A (Header Acknowledge, if required by the message)
d. D (Data, if required by the message)
e. A (Data Acknowledge, if required by the message).

The Slave(s) synchronizes its own state machine with the H0 cycle of the Pi-Bus. It then checks the sequence of the Cycle Type signals to verify that the message is proceeding correctly.

Both the Master and Slave can source data for the Data signals. The Master starts by placing the header words on the Data signals. This specifies the following:

a. Participating Slave(s)
b. Message Type
c. Direction of Data Transfer
d. Data Buffer
e. Amount of Data to be Transferred

The Slave responds to the header by asserting the Header Acknowledge(s) on the Data signals. This provides error data and participating Slave identification to the Master. The Master will abort the message if the header and header acknowledge portion of the message do not complete correctly. The header acknowledge is followed by the Master (for a transmit message) or the Slave (for a receive message) asserting data on the Data signals.

The Slave sources the Acknowledge Set signals. These lines are used to inform the Master of the Slave's participation in the message. The following can be indicated:

a. NS (Not Selected)
b. RCG (Recognized)
c. ACK (Acknowledge)
d. NAK (Negative Acknowledge)

The Master monitors these signals to insure that they follow the proper protocol.

1.3.1 (Continued):

Either the Master or Slave can source the Wait signals. These signals are used to insert Non-Transfer (NT) cycles on the Pi-Bus. The cycle after the assertion of the Wait is the NT cycle. The delayed effect is required because of the pipeline type design of the Pi-Bus. This NT cycle technique is used to cause a pause in the message to permit the Master or Slave time to perform tasks such as:

a. Fetch or Store Data
b. Check Label Tables
c. Allow checking for Aborts after the End of a Message
d. Store Interrupt Reports
e. Updating Label Tables
f. Calculation of Resume Control Words
g. Fetching the Next Bus Message Control Block.

The proper use of the Wait signals is one of the most difficult tasks in the implementation of a BIU. This subject will be covered in detail later.

The Bus Request lines can be sourced by any Master/Slave module. It is used to indicate that a module has a message to transmit which has a priority which is higher than the current bus priority. The Master must follow the specified rules concerning suspending and aborting to relinquish Mastership of the bus.

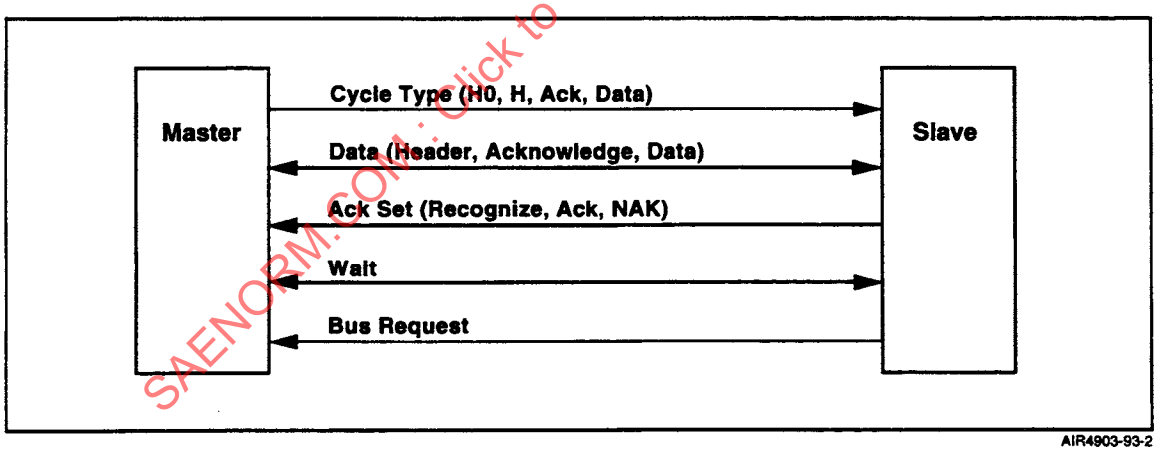

FIGURE 7 - Buses Used for Pi-Bus Messages

1.3.2 Pi-Bus Implementation: A typical implementation of the data flow for a BIU is shown in Figure 8. The basic principal is that all data, control and redundancy bits are loaded into an Output Register to be asserted onto the Pi-Bus for a complete cycle. They are asserted onto the Pi-Bus during the same cycle that they are in the Output Register. The signals are received into an Input Register at the end of the bus cycle. Once the data and controls are loaded in the Input Register, they are checked and decoded during the next cycle. Data is then transferred to a final destination or End Register.

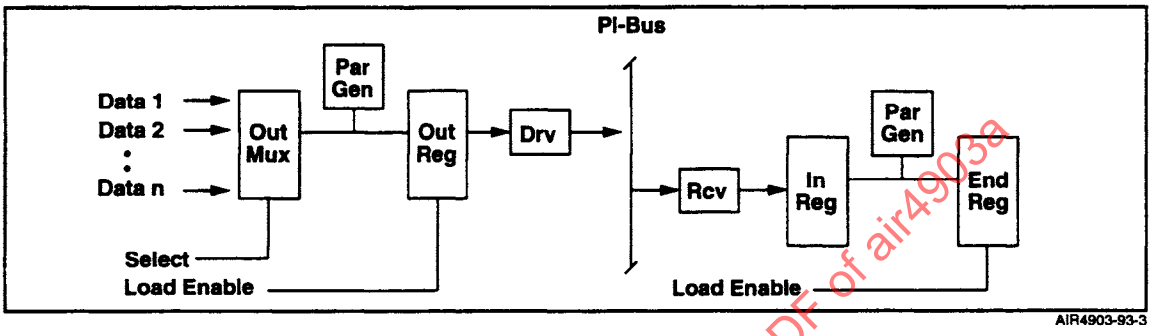

FIGURE 8 - Data and Control Flow for a Pi-Bus

The implementation of a synchronous data flow such as this resembles a pipeline as shown in Figure 9. This shows that a Slave destination register is loaded with data which was selected three cycles earlier by the Master Output Mux.

Data n	D0	D1	D2	D3
Out Reg	n	D0	D1	D2
PI-Bus	n	D0	D1	D2
In Reg	n	n	D0	D1
End Reg	n	n	n	D0

AIR4903-93-4

FIGURE 9 - Data and Control Flow for a Pi-Bus

The "pipeline" concept helps explain the Pi-Bus requirement of having a Slave assert NAK on the bus two cycles after an error is detected. The following sequence illustrates why asserting NAK in two cycles following an error is as fast as it can be accomplished in the natural flow of the protocol:

Cycle 1 Error occurs on a signal bus of the Pi-Bus
Cycle 2 Slave latches bus signals in the Input Register, detects the error and forms a NAK.
Cycle 3 Slave loads the NAK into the Output Register and asserts it on the Pi-Bus

2. REFERENCES:

The following publications form a part of this specification to the extent specified herein. The latest issue of SAE publications shall apply. The applicable issue of other publications shall be the issue in effect on the date of the purchase order. In the event of conflict between the text of this specification and references cited herein, the text of this specification takes precedence. Nothing in this specification, however, supersedes applicable laws and regulations unless a specific exemption has been obtained.

2.1 SAE Publications:

Available from SAE, 400 Commonwealth Drive, Warrendale, PA 15096-0001.

AS4710 Pi-Bus Standard, May 1993

2.2 Other Publications:

[IBM1] Avionics Pi-Bus Study - CDRL A004 Pi-Bus Error Class Feasibility Report, IBM, April 1990

[NAT1] "High-Performance Bus Interface Designer's Guide", Number 550095, National Semiconductor, 1991

[VHSIC] VHSIC Phase2 Interoperatability Standards Pi-Bus Specification, Version 2.2, IBM, Honeywell, TRW, 1988

3. PI-BUS PHYSICAL LAYER RATIONALE:

This section provides rationale for Pi-Bus physical layer requirements. The paragraph numbers in this section map in a one-to-one fashion to the paragraphs in Section 3 of AS4710. For some paragraphs, rationale is not applicable and is so noted. In some cases, the working group has been unable to determine the rationale beyond it being included in [VHSIC].

3.1 Introduction:

Rationale: Not applicable since no requirements specified.

3.2 Line Definitions:

Rationale: Not applicable since no requirements specified.

3.2.1 Nomenclature:

Rationale: Arbitrary

3.2.2 Bused Signal Lines:

A. Pi-Bus lines shall be implemented as wired-or lines.

Rationale: Chosen as a proven mechanism to combine signals from multiple modules.

B. Symbols posted on signal lines shall be valid symbols except during diagnostic operations, invalid symbols are allowed as specified in 4.3.9.5.

Rationale: Rationale not required.

C. Pi-Bus modules shall provide a command path independent of Pi-Bus which provides a way to force all Pi-Bus signals to be released by the module.

Rationale: This is required for fault tolerance and diagnostic support. This requirement implies the requirement for a test and maintenance bus such as the VHSIC TM-Bus.

3.2.2.1 Data Line Group (D//DC):

A. Data line groups.

Rationale: Bi-directional lines used for transfer of header, data, and acknowledge information between bus Master and Slave(s). Also, used to resolve priority during vie.

3.2.2.1.1 Data Lines - Type 16:

A. 16-bit data lines.

Rationale: 16-bit Pi-Bus arbitrarily chosen as minimal Pi-Bus transfer size. Choice was influenced by the assumption that MIL-STD-1750A, standard military 16-bit processor, would be initial module processor.

3.2.2.1.2 Data Lines - Type 32:

A. 32-bit Pi-Bus.

Rationale: Growth option for Pi-Bus which supports higher transfer rate. This allows simpler module design for 32-bit processors that only access memory in 32-bit words. In those systems, doing a 16-bit read or write to memory can be difficult. Using a 16-bit Pi-Bus can result in the need for a 16-bit memory access when an odd number of words are sent, or if a suspend occurs on an odd word boundary.

3.2.2.1.3 Error Protection for the Data Line Group:

A. Even parity for error detection.

Rationale: Even parity chosen since this gives valid parity even when the bus is idle.

B. Duplication and triplication for multisourced signals for error detection and correction, respectively.

Rationale: Duplication and triplication are proven approaches for fault detection and correction, respectively. The following text, based on material from [IBM1], provides rationale for the 32-bit detection and correction of multiple errors.

Detection and Correction of Multiple Errors: The error correction codes chosen for the SAE Pi-Bus has the capability of correcting at least one error and will detect any double bit error. The code, however, is designed to detect all triple bit errors. The code is capable of detecting some percentage of triple bit errors as uncorrectable errors. There are a total of 9880 possible triple errors with the SAE error correction code. This code will detect 6776 of these errors for a detection percentage of 68.6%.

In an effort to maintain the advantages of a single-error-correct, double-error-detect, package-error-detect code while keeping to an 8 transceiver design, a new EC code was developed specifically for the Pi-Bus application by C. L. Chen and J. R. Burns of IBM. It has been modified so that all 40 bits of the code may be wired on 8 transceivers without any loss in the ability to detect a transceiver fail.

The original Pi-Bus ECC as developed by C. L. Chen was to put four data bits and one check bit on each transceiver. However, due to the requirements of allowing message transfer between 16-bit and 32-bit modules on the Pi-Bus, two problems emerged:

a. Only three extra lines are left available on transceivers 5-8 for error correction; an EC code with one check bit per transceiver could not be implemented.

b. At least 29 lines need to be used on transceivers 1-4; without a modification to the EC code, only 27 lines would be used.

Therefore, Modified Pi-Bus EC code was designed to correct all single bit errors and detect all double bit errors. In addition, Modified Pi-Bus EC code will detect any error combination on a single transceiver - up to six errors on any of the eight transceivers. All of the transceivers will protect four data bits; four of the transceivers will protect 1 check bit, two will protect 2 check bits, and two will protect no check bits.

3.2.2.1.3 (Continued):

The parity check matrix for Modified Pi-Bus error correction code is shown in Figure 10. This code requires eight 12-bit XOR trees to encode each check bit and eight 13-bit XOR trees to decode each syndrome bit. Additional logic is required to determine that an error has occurred, find if the error is correctable, fix the error if it is correctable, and notify the device if the error is not correctable. A total of 40 bits is required to implement Modified Pi-Bus ECC. For the error correction properties of this code to work correctly when an entire transceiver module fails, the wiring to the 8 Pi-Bus transceivers must be done as illustrated in the check bit matrix, Figure 10. The 'Tx' reference across the top of Figure 10 indicates what transceiver package the data or check bits are wired to on the module. The check bits are included in the matrix to show that they must be wired to different transceivers. A summary of the error codes is given in Figure 11.

	T1						T2						T3					T4					T5				T6				T7					T8				
CH. BIT	31	30	29	28	C7	C5	27	26	25	24	C6	C4	23	22	21	20	C3	19	18	17	16	C2	15	14	13	12	11	10	9	8	7	6	5	4	C1	3	2	1	0	C0
7	X	X			X				X	X			X					X					X	X					X	X	X					X				
6			X	X			X	X			X			X					X						X	X	X	X				X					X			
5			X	X		X	X	X							X					X			X	X					X	X			X					X		
4	X	X							X	X		X				X					X				X	X	X	X						X					X	
3	X						X						X	X			X			X	X		X				X				X	X					X	X	
2		X						X							X	X		X	X			X		X				X					X	X		X	X			
1			X						X				X	X						X	X				X				X				X	X	X	X	X			
0				X						X					X	X		X	X							X				X	X	X					X	X	X

All bits marked with X in a horizontal row are XORed to generate each of the eight check bits.

Transceivers 3-4, 7-8 protect 4 data bits and 1 check bit.
Transceivers 1-2 protect 4 data bits and 2 check bits.
Transceivers 5-6 protect 4 data bits and no check bits.

Figure 10 - Modified Pi-Bus ECC Check Bit Matrix

The equations for the data check lines would be
P(DC<7>, D<31,30,25,24,23,19,15,14,9,8,7,3>) = 0
P(DC<6>, D<29,28,27,26,22,18,13,12,11,10,6,2>) = 0
P(DC<5>, D<29,28,27,26,21,17,15,14,9,8,5,1>) = 0
P(DC<4>, D<31,30,25,24,20,16,13,12,11,10,4,0>) = 0
P(DC<3>,D<31,27,23,22,17,16,15,11,7,6,1,0>) = 0
P(DC<2>,D<30,26,21,20,19,18,14,10,5,4,3,2>) = 0
P(DC<1>,D<29,25,23,22,17,16,13,9,5,4,3,2>) = 0
P(DC<0>,D<28,24,21,20,19,18,12,8,7,6,1,0>) = 0

	single error correction	double error correction/ detection	triple error correction/ detection	transceiver failure correction/ detection	time to fix a correct. error	time to indentify uncorr. error	bits in code - # trans.	cell size of design
Original VHSIC error correction code	Yes	Detect	40.3% detected	No	20.1 nS	24.7nS	39 bits	697 cells
Error correction code used in SAE Pi-Bus	Yes	Detect	68.6% detected	Detect (up to 6)	20.3 nS	24.9 nS	40 bits 8 trans.	682 cells

FIGURE 11 - Error Correcting Codes - Summary

3.2.2.1.3 (Continued):

Bit Grouping for Error Detection and Error Correction

There are many possible ways to group the Pi-Bus signal lines onto the Pi-Bus transceivers. The main concern in grouping the data, control, and check bits on the transceivers is maximizing the error correction and error detection capabilities of the bus. This is done by:

a. Grouping the data and data check bits as specified in the previous text, so that the failure of a transceiver will be corrected (Product ECC) or detected ((40,32), Modified Pi-Bus codes).

b. Group the control bits so that no more than one line of any control field is wired to a single transceiver. A failure of one transceiver will then result in a correctable control field error.

In addition, the bits need to be grouped so that a Dual 16-bit ED module can operate on the same bus as a 32 EC module. Therefore, the 29 I/O required to implement 16-bit error detection data transfers must all be placed on transceivers 5-8. (The extra lines needed to implement 32-bit error correction data transfers may be placed on transceivers 1-4.) This also means that at least 29 I/O lines must be utilized on transceivers 1-4, so that the dual 16-bit bus can be placed entirely on transceivers 1-4.

One such bit grouping is described below in Figure 12 . The bit grouping in Figure 12 is set up so that it will work with Modified Pi-Bus error correction code. If any single transceiver fails, the data bit errors will be detected and the control bit errors will be corrected.

3.2.2.1.3 (Continued):

The VHSIC Pi-Bus 7-bit modified Hamming code can have the data and data check bits grouped on the Pi-Bus transceivers in any order. The bit grouping in Figure 12 will also work if 7-bit modified Hamming code is implemented.

This bit grouping design assumes that eight 8-bit transceivers are available for use. A total of 60 transceiver lines is used to allow 16-bit error detection and 32-bit error correction message transfer on the module. Four transceiver lines are unassigned with this bit grouping; these lines could be used for new logic added at a later date or for further fault tolerance measures such as bit sparing.

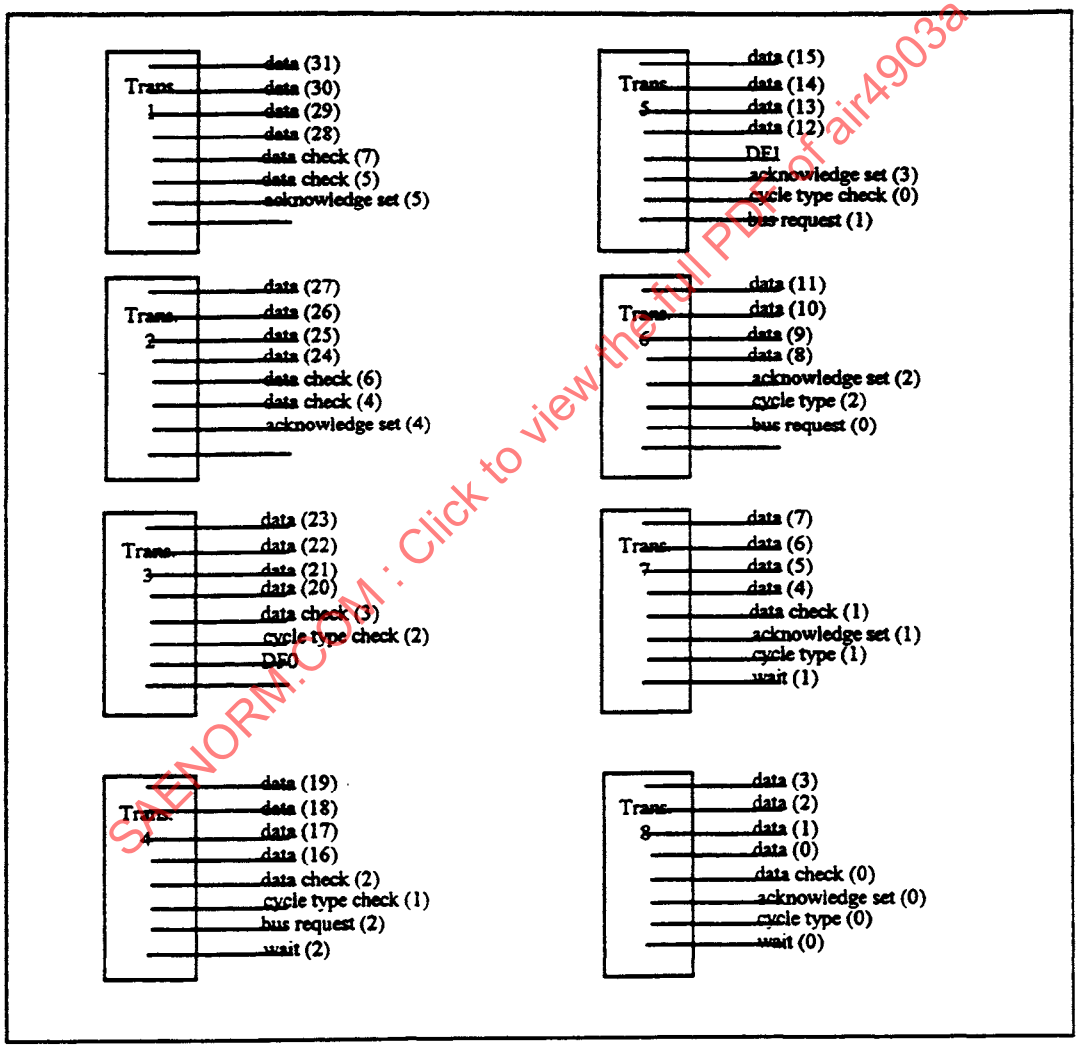

FIGURE 12 - Bit Grouping Scheme for Modified Pi-Bus ECC - Also Works for 7-Bit Modified Hamming Code

3.2.2.1.3.1 Class ED Operation:

Rationale: Not applicable since no requirements specified.

3.2.2.1.3.1.1 Single Source:

A. Even parity for Data Line Group.

Rationale: Even parity chosen since this gives valid parity even when the bus is idle.

3.2.2.1.3.1.1.1 Type 16:

Rationale: The formula presented in this paragraph detects single bit errors when even parity is used.

3.2.2.1.3.1.1.2 Type 32:

A: 32-bit ED not allowed.

Rationale: It was felt that a system would prefer to have a single 32-bit EC instead of two 32-bit ED Pi-Buses due to difference in number of lines and due to seamless error correction capability. Also, it was felt that there would not be a market for 32-bit ED Pi-Bus. Therefore, to reduce the number of options, 32-bit ED is not allowed.

3.2.2.1.3.1.2 Multiple Source:

Rationale: No rationale required.

3.2.2.1.3.2 Class EC Operation:

Rationale: Not applicable since no requirements specified.

3.2.2.1.3.2.1 Single Source:

Rationale: Not applicable since no requirements specified.

3.2.2.1.3.2.1.1 Type 16:

A: 16-EC not allowed.

Rationale: It was felt that there was not a market for 16-bit EC Pi-Bus. The opinion was anyone wanting to implement complexity of error correction would implement a 32-bit Pi-Bus. Therefore, to limit the number of Pi-bus options, 16-bit EC Pi-Bus is not allowed.

3.2.2.1.3.2.1.2 Type 32 Nonmixed Mode:

A: Simultaneous 16-ED and 32-EC not allowed.

Rationale: Full mixed mode operation requires an additional signal line. Many application require either 16ED or 32-EC operation but not simultaneously. This mode allows an optimized implementation with one less signal line than required in a mixed mode implementation.

3.2.2.1.3.2.1.3 Type 32 Mixed Mode:

A: Simultaneous 16-ED and 32-EC allowed.

Rationale: Included to support inclusion of modules developed for 16-ED systems into 32-EC systems.

3.2.2.1.3.2.2 Multiple Sources:

Rationale: Rationale not required.

3.2.2.2 Cycle Type Line Group (CT//CTC):

Rationale: Cycle Type lines are used to indicate current Pi-Bus cycle and the CTC lines are used for error detection/correction of the CT lines.

3.2.2.2.1 Class ED:

Rationale: The formula presented detects single bit errors in the Cycle Type line group when using even parity.

3.2.2.2.2 Class EC:

Rationale: The formula presented detects and corrects single bit errors in the Cycle Type line group.

3.2.2.3 Acknowledge Line Set (AS):

Rationale: AS lines are used by Slaves or contenders to indicate synchronization or indicate uncorrectable detected errors.

3.2.2.3.1 Class ED:

Rationale: Duplication is a proven approach for detection of single bit errors.

3.2.2.3. Class EC:

Rationale: Triplication is a proven approach for correction of single bit errors.

3.2.2.4 Wait (W) Lines:

Rationale: Wait lines are used by Master/Slaves to obtain additional non-transfer cycles. This is necessary when the PIU must "delay" the Pi-Bus to perform its operations.

3.2.2.4.1 Class ED:

Rationale: Duplication is proven approach for detection of single bit errors.

3.2.2.4.2 Class EC:

Rationale: Triplication is proven approach for correction of single bit errors.

3.2.2.5 Bus Request (BR) Lines:

Rationale: The BR lines are used to request that the current bus Master release the bus.

3.2.2.5.1 Class ED:

Rationale: Duplication is proven approach for detection of single bit errors.

3.2.2.5.2 Class EC:

Rationale: Triplication is proven approach to correction of single bit errors.

3.2.2.6 Data Format (DF) Line:

Rationale: The DF lines are used to indicate the message type during mixed mode 32-bit operation. This allows the determination of whether a message is of type 16 or 32.

3.2.2.6.1 Class ED:

Rationale: Since modules of type 16 can only send/receive messages of type 16, they cannot source the DF lines.

3.2.2.6.2 Class EC:

Rationale: Since nonmixed mode modules only source type 32 messages, it cannot source the DF lines.

3.2.2.6.3 Class EC Mixed Mode:

Rationale: Only class EC-mixed modules can source the DF lines. This is required for correct mixed-mode operation.

3.2.3 Bus Clock:

Rationale: Clock is required since Pi-Bus is a synchronous bus.

3.2.4 Module Identification (MID):

Rationale: MID provides unique module identification on the backplane.

3.2.4.1 Module Identification (MID) Lines:

Rationale: Five bits are needed to represent 32 modules.

3.2.4.2 Module Identification Parity (MIP) Line:

Rationale: MIP line is provided to detect single bit module ID errors.

3.2.5 32-Bit EC/Dual 16-Bit ED Interoperability:

Rationale: By proper PIU and module design, a 32-bit EC module can be capable of communication as a 32-bit module or as a dual 16-bit ED module, with no increase in module I/O pins.

3.2.6 Signal Drive Partitioning:

Rationale: Rationale not provided.

3.3 Electrical Requirements.

The following is an overview taken from [NAT1].

a. Pi-Bus History: Through the 70s and 80s the typical backplane was driven by standard TTL logic parts with tristateable outputs such as 54/74XX240 and 245. For design purposes these buses were modeled as lumped capacitances and transmission line effects were ignored because the bus data rates were generally not fast enough to require designers to be concerned with transmission line effects of the backplane. With the lower bus speeds there was sufficient bus setting time for reflections to dampen, and the signals to stabilize close enough to steady state to avoid problems. If a bus was heavily loaded and attempts were made to run the bus faster than 10 MHz, errors in data were frequently seen due to line reflections causing line voltage levels to transition through the threshold more than once for a single transition of the buffer input. As backplane densities increased, loading from the transceivers became so great that this increased the problem of driving the bus quickly and error free.

3.3 (Continued):

f. Precision Threshold: In order to handle the smaller output swing on the bus, it is necessary to use a more precise threshold circuit on the bus receiver input. Figure 17a is a typical TTL input buffer. The threshold set by this type of circuit varies considerably with temperature, and typically ranges from about 1V at high temperatures to about 1.8V at low temperatures. Such movement could not be tolerated with a bus swing which could be as small as from 1.15 to 1.9 V. The circuit shown in Figure 17b is used on Pi-Bus transceivers to set a more precise threshold. A voltage reference with a very small V_{CC} and temperature dependence is placed on the chip to establish a precision threshold for the Pi-Bus to BIU Input buffer. By using a differential pair with one of the pair controlled by the reference voltage, the threshold can be maintained within a 150 mV window centered at 1.52 V.

g. Output Ramps and Noise Immunity: Ramped outputs have been touted as the solution to problems of bus reflections and crosstalk. The amount of ramp time put into rise and fall times is directly related to the propagation delays of a transceiver, so longer ramps require longer delay times. An important question to ask is how much of a ramp buys what degree of decreased bus setting time. Many TTL parts have peak ramps of about 2 V/ns. This rate of ramp certainly seems to increase bus setting times. The Pi-Bus transceiver will have a typical ramp rate of about 0.5 V/ns.

Having some amount of noise rejection on the bus receiver input allows the bus buffer input to ignore small excursions above or below threshold without affecting the data being transmitted to the BIU. However the greater the amount of noise immunity, the greater the propagation delay on the path from the bus to the BIU. The Pi-Bus transceiver offers a compromise between noise immunity and prop delays with typically 4 ns of pulsewidth protection measured at 1.5 V for a 1 to 2 V input.

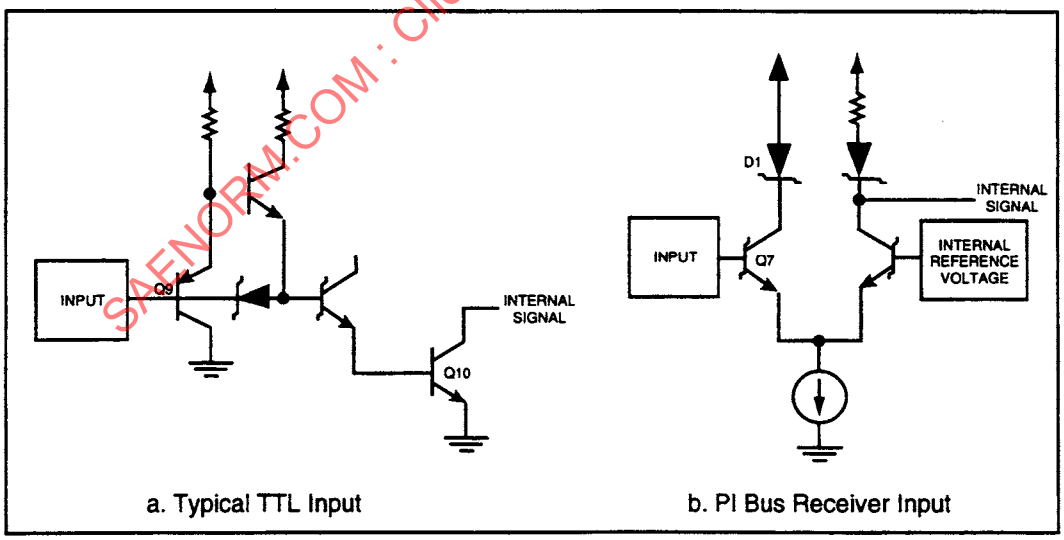

FIGURE 17 - Pi-Bus Driver Input

3.3 (Continued):

h. Bipolar Versus CMOS BICMOS Technology: A full bipolar transceiver could be built to provide excellent performance in most areas with the exception of power consumption in both the active and inactive modes. A pure CMOS part could be designed with a much lower I_{CC}, but it would have the following drawbacks. It would be slower than a bipolar part, no Schottky diode would be available to construct a low capacitance output with limited swing, and stable CMOS voltage references which have a reasonable silicon area are non-existent, so setting a precise input voltage threshold would be difficult.

By making judicious use of bipolar and CMOS circuits where each is appropriate, it was possible to design a Pi-Bus transceiver with an I_{CC} inactive of typically 35 mA and yet maintain the full features which make Pi-Bus a clean and quiet bus to run.

i. Example Pi-Bus Transceiver.: Some BTL transceivers feature a nominal 1 V signal swing for low power consumption, with receivers having precise thresholds for maximum noise immunity, and drivers with low power capacitance to minimize bus loading. These features combine to allow higher bus-data-transfer rates and improved overall system reliability. They also eliminate performance-degrading settling-time delays.

3.3.1 Backplane Requirements:

Rationale: Not applicable since no requirements are specified.

3.3.1.1 Bus Signal Line Characteristic Impedance:

Rationale: See rationale in 3.3. Additionally, reference text within 3.3.1.1 of AS4710.

3.3.1.1.1 Example Bus Analysis:

Rationale: Not applicable since no requirements are specified - this merely provides definitions for an example.

3.3.1.1.2 Analysis:

Rationale: Not applicable since no requirements are specified - this merely provides an example analysis.

3.3.1.2 Bus Signal Line Termination:

Rationale: See rationale in 3.3.

3.3.1.3 Bus Signal Line Resistance:

Rationale: See rationale in 3.3.

3.3.1.4 Module Identification Line Resistance:

Rationale: Rationale not provided.

3.3.1.5 Bus Clock Requirements:

Rationale: Not applicable since no requirements are specified.

3.3.1.5.1 Voltage Levels:

Rationale: Refer to 3.3. for derivation.

3.3.1.5.2 Rise and Fall Time:

Rationale: Rationale not provided.

3.3.1.5.3 Duty Cycle:

Rationale: Rationale not provided.

3.3.2 Module Requirements:

Rationale: Not applicable since no requirements are specified.

3.3.2.1 Bus Clock Requirements:

Rationale: Rationale not provided.

3.3.2.1.1 DC Requirements:

Rationale: Refer to 3.3.

3.3.2.1.1.1 Input Characteristics:

Rationale: Refer to 3.3.

3.3.2.1.1.2 Input Inductance:

Rationale: Refer to 3.3.

3.3.2.1.1.3 Bus Clock Current:

Rationale: Refer to 3.3.

3.3.2.1.1.4 High-Level Input Voltage:

Rationale: Refer to 3.3.

3.3.2.1.1.5 Low-Level Input Voltage:

Rationale: Refer to 3.3.

3.3.2.1.2 AC Requirements:

Rationale: Refer to 3.3.

3.3.2.2 Signal line requirements.

Rationale: Not applicable, since no requirements are specified.

3.3.2.2.1 DC Requirements:

Rationale: Refer to 3.3.

3.3.2.2.1.1 Input Capacitance:

Rationale: Refer to 3.3.

3.3.2.2.1.2 Input Inductance:

Rationale: Refer to 3.3.

3.3.2.2.1.3 Leakage current.

Rationale: Refer to 3.3.

3.3.2.2.1.4 Low-Level Sink Current:

Rationale: Refer to 3.3.

3.3.2.2.1.5 High-Level Output Voltage:

Rationale: Refer to 3.3.

3.3.2.2.1.6 Low-Level Output Voltage:

Rationale: Refer to 3.3.

3.3.2.2.1.7 High-Level Input Voltage:

Rationale: Refer to 3.3.

3.3.2.2.1.8 Low-Level Input Voltage:

Rationale: Refer to 3.3.

3.3.2.2.2 AC Requirements:

Rationale: Not applicable since no requirements are specified.

3.3.2.2.2.1 Signal Line Inputs:

Rationale: Not applicable since no requirements are specified. - it just refers to the figure illustrating timing relationships specified in subparagraphs.

3.3.2.2.2.1.1 Setup Time:

Rationale: Refer to 3.3.

3.3.2.2.2.1.2 Hold-Time:

Rationale: Refer to 3.3.

3.3.2.2.2.1.3 Noise Rejection:

Rationale: Refer to 3.3.

3.3.2.2.2.2 Signal Line Outputs:

Rationale: Refer to 3.3.

3.3.2.2.2.2.1 Propagation Delay:

Rationale: Refer to 3.3.

3.3.2.2.2.2.2 Rise and Fall Time:

Rationale: Refer to 3.3.

3.3.2.3 MID and MIP Lines:

Rationale: Rationale not provided.

4. PI-BUS DATA LINK LAYER RATIONALE:

This section provides rationale for Pi-Bus data link layer requirements. The paragraph numbers in this section map in a one-to-one fashion to the paragraphs in Section 3 of AS4710. In some cases rationale is not applicable and is so noted. In some cases, the working group has been unable to determine the rationale beyond it being included in VHSIC Phase 2.

4.1 Introduction:

Rationale: Not applicable since no requirements specified.

4.2 General Requirements:

Rationale: Not applicable since no requirements specified.

4.2.1 Introduction:

Rationale: Rationale not provided.

4.2.2 Protocol State Transitions:

Rationale: Rationale not provided.

4.2.2.1 Idle:

Rationale: Rationale not provided.

4.2.2.2 Vie:

Rationale: Rationale not provided.

4.2.2.3 Header:

Rationale: Rationale not provided.

4.2.2.4 Header Acknowledge:

Rationale: Rationale not provided.

4.2.2.5 Data:

Rationale: Rationale not provided.

4.2.2.6 Suspend:

Rationale: Rationale not provided.

4.2.2.7 Data Acknowledge:

Rationale: Rationale not provided.

4.2.2.8 Abort:

Rationale: Rationale not provided.

4.2.2.9 Tenure Limitations:

Rationale: Not applicable since no requirements specified.

4.2.2.9.1 Bus Request to Vie Interval:

Rationale: Rationale not provided.

4.2.2.9.2 Absolute Tenure Limit:

Rationale: Included to preclude master device errors from precluding bus lock out. Functions as a watchdog timer.

4.2.3 Generic Message:

Rationale: Not applicable - paragraph contains rationale.

4.2.3.1 Generic Message Sequence:

Rationale: Not applicable - paragraph contains rationale.

4.2.3.1.1 Normal Operations:

Rationale: Not applicable - paragraph contains rationale.

4.2.3.1.1.1 Header.

Rationale: Not applicable - paragraph contains rationale.

4.2.3.1.1.2 Header Acknowledge:

Rationale: Not applicable - paragraph contains rationale.

4.2.3.1.1.3 Data:

Rationale: Not applicable - paragraph contains rationale.

4.2.3.1.1.4 Data Acknowledge:

Rationale: Not applicable - paragraph contains rationale.

4.2.3.1.2 Operations Under Exception Conditions:

Rationale: Not applicable since no requirements are specified.

4.2.3.1.2.1 Suspend:

Rationale: Rationale not provided.

4.2.3.1.2.2 Uncorrectable Line Errors:

Rationale: Rationale not provided.

4.2.3.2 Generic Header Information:

4.2.3.2.1 Header Word A:

Rationale: Rationale not provided.

4.2.3.2.1.1 Slave ID Field:

Rationale: Rationale not provided.

4.2.3.2.1.2 Format (F) Field:

Rationale: Rationale not provided.

4.2.3.2.1.2.1 Type 16 ED:

Rationale: Rationale not provided.

4.2.3.2.1.2.2 Type 32 EC Nonmixed Mode:

Rationale: Rationale not provided.

4.2.3.2.1.2.3 Type 32 EC Mixed Mode:

Rationale: Rationale not provided.

4.2.3.2.1.3 Message Type (MSG TYPE):

A: Inclusion of Block Messages.

Rationale: Block Messages were included since it is basic message type used to transfer application data between modules over the Pi-Bus. Both extended and nonextended header block messages are required. Nonextended headers were required since they are more efficient in cases where extended headers are not needed. Extended headers were required to support associating operating system information and possibly security information with the message without this information being required to be embedded in the message data.

Label addressing is required since it supports a logical messaging mode which does not require the Master to know the buffer location in Slave memory and, thus, allows the Slave to have control/protection over its own memory as far as placement of received messages is concerned. Also, the label is treated as an extension to the Slave ID field. In many applications, it was noticed that 32 physical, one broadcast, and 223 logical Slave IDs was insufficient.

4.2.3.2.1.3 (Continued):

The direct addressing mode was defined to support "dumb" modules, i.e., modules without a processor. The address is as specified by the Slave module, (i.e., it is the Slave's memory map). It is intended that this mode not be used in smart modules in support of security concepts.

Both new message and resume message AT codes are supported for label and direct messages in support of Pi-Bus suspend sequences.

B: Inclusion of Parameter Write Messages.

Rationale: The parameter write message type was included to allow an efficient message in which there were 1-3 words of data, all included in the Pi-Bus message header. Only a logical AT code was included since this was all that was required in JIAWG systems.

C: Exclusion of Tenure Pass.

Rationale: Tenure Pass was not included since it was not required for JIAWG systems. In particular, the JIAWG Pi-Bus backplane is used for loosely coupled modules, and passing tenure between modules via tenure pass implies that the modules are tightly coupled.

D: Inclusion of Bus Interface Messages.

Rationale: Bus Interface Messages were included to allow the data link registers to be read/modified from another module and to allow system time to be distributed over the Pi-Bus. The data link access requirement is not absolutely required for "smart" modules (i.e., this is not needed in a system with distributed control), but it may be required for setting up "dumb" modules.

E: Inclusion of Datagram Messages

Rationale: Datagram messages were included to create a higher throughput message type than provided by Block messages. By removing the header acknowledge cycles, the potential Slaves in a message can perform label processing in parallel with data transfers.

4.2.3.2.1.4 Access Type (AT) Field:

Rationale: Rationale not provided.

4.2.3.3 Header and Data Sequence Acknowledge:

A: Acknowledge sequences.

Rationale: The acknowledge sequences of both the header and data provide two types of error indications to the Master device:

a. Pi-Bus errors (line, sequence, etc.)
b. Device errors (device interface line errors, busy, etc.)

Therefore they can give the Master device complete knowledge regarding the successful or unsuccessful completion of a message.

B: Separate header and data acknowledge sequences.

Rationale: The header and data acknowledge sequences are separate to give the Master device visibility into which part of the sequence did not successfully complete. This is important since responses to errors in header and data are often different. For example, most line errors occurring in the header result in abort since the message control data has been corrupted. This eliminates the need to proceed with a potentially long data sequence that may be invalid. However, some applications may work through errors that occur during the data sequence.

4.2.3.3.1 Single Slave Acknowledge:

Rationale: Rationale same as 4.2.3.3.

4.2.3.3.1.1 Slave Module Identification (MID) Field:

Rationale: Can be used as diagnostic and also allows Master to identify slave in case logical addressing is used.

4.2.3.3.1.2 Acknowledge Word Type (AWT) Field:

A: States that for exception situations where Slave ceases to participate in the bus sequence after header ACK completion, the AWT field is set to 00.

Rationale: AWT = 00 represents message complete. This statement was added since this situation was implied but not explicitly called out in the VHSIC Pi-Bus Spec. except for data line errors during header (Reference Table 5-35, p.5-88)

4.2.3.3.1.3 Errors Field:

Description: AS4710 states that the defined format for command error refers only to command format.

Rationale: Rationale not provided.

4.2.3.3.1.4 Busy (B) Field:

Description: States if Slave is busy during header sequence, the Slave shall cease to participate in bus sequence after header ACK completion and the AWT field shall be set to 00 as defined in 4.2.3.3.1.2.

Rationale: If a Slave knows it is busy, there is no reason for it to continue in the bus transfer. In VHSIC Pi-Bus specification it sets AWT as if not busy but does not require the Master to abort. Having the message complete (and thus setting AWT = 00) was deemed more consistent with the way error conditions are treated.

4.2.3.3.1.5 Suspend (S) Field:

Description: States that during header sequence in which an error or Busy is detected by a Slave and the Slave will cease to participate after header ACK, the "S" bit is set to one in header ACK word.

Rationale: There is no reason to allow a message to be suspended and resumed to a Slave who detects errors or is busy. This will also help prevent such a Slave from corrupting a multicast message transfer during a suspend/resume sequence.

4.2.3.3.2 Multiple Slave Acknowledge:

Rationale: Not applicable since no requirements are applicable.

4.2.3.3.2.1 Nondatagram:

A: Multiple Slave acknowledge symbols.

Rationale: During HA1 - HA4, the acknowledge symbols should be asserted by all selected Slaves (unless a module has ceased being a Slave due to a specific condition from the error tables) to indicate to the Master device which Slaves are participating in a given message. During DA1 - DA4, only modules which completed without error should post "Acknowledge" symbols to indicate a successful completion. The Master device can determine from this information which modules may need re-transmission of data.

4.2.3.3.2.2 Datagram:

Rationale: Multiple slave acknowledge not included in interest of performance.

4.3 Detailed Requirements:

Rationale: Not applicable since no requirements are specified.

4.3.1 Introduction:

Rationale: Not applicable since no requirements are specified.

4.3.2 Bus State Definitions:

Rationale: Rationale not provided.

4.3.3 Bus Mastership:

Description: Defines protocol governing bus Mastership.

Rationale: Only a single module may be a Master at any time. The bus Mastership protocol defines a protocol for ensuring that at most a single module is a Master and for allowing suspension of a current block transfer by a module attempting to send a message with a more urgent logical priority.

4.3.3.1 Vie Sequence:

Description: The paragraph on pages 5-29 of the VHSIC Pi-Bus Spec. was clarified to state that "when a module loses contention and quits driving the bus for the remainder of the vie sequence" is meant that "the Slave ceases to be a contender but does remain an active module".

Rationale: This rewording uses better terminology and makes clear that the module is still active and still follows the vie sequence.

4.3.3.1.1 16-Bit and 32-Bit Nonmixed Mode Vie:

Rationale: Rationale not provided.

4.3.3.1.2 Mixed Mode Vie:

A. Vie_Mode_EC bit needed.

Rationale: The Vie_Mode_EC bit is specified to determine what type of Vie sequence to perform. Basically, it tell the BIU whether there are any Type 16 ED modules on the bus. If there are any, the Vie must be performed using error detection only. If some modules used detection while others used correction, multiple Masters could result when error occurred during the Vie.

4.3.3.2 Tenure Pass Message:

Description: States tenure pass shall not be implemented.

Rationale: Reference rationale paragraph 4.2.3.2.1.3.C. Causes command error since Tenure Pass is an illegal message type in JIAWG systems.

4.3.3.3 Bus Request:

Rationale: Rationale not provided.

4.3.4 Message Sequences:

Rationale: Not applicable since no requirements are specified.

4.3.4.1 Parameter Write Message Sequence:

Description: Defines parameter write header words and states that a Slave shall have a mechanism to access the transferred data.

Rationale: Parameter write message was included to allow an optimized message passing mechanism (as compared to block message) when less than or equal to three 16-bit words need to be transferred. The optimization occurs by this type of message not having a data sequence. The header words are defined per the Pi-Bus spec except that HWC0 and HWC1 are allowed to contain data. This follows since no address indicator is needed for a logical parameter write - the Slave BIU is responsible for saving the data and making it accessible to the Slave.

4.3.4.1.1 Logical Type:

Rationale: Optimized block message in case data is three words or less.

4.3.4.1.2 Event Filter Type:

A: Event Filter Parameter Write Message.

Rationale: This message type was included as an efficient mechanism of a barrier synchronization protocol. Two example applications of Event Filter include the following. (1) It can be used to aid synchronization of a system's startup sequence. In a loosely coupled system, each module will take a slightly different time period to get through its start up sequence. In addition, it cannot be assumed that each module will even come up. Typically, there are a couple of points in the startup sequence to which it is desirable that all modules progress prior to all modules proceeding to the next phase. It is desirable that a system come up as fast as possible, but it must be guaranteed that all modules have enough time to complete their startup. The solution to this is an event filter message for each startup phase. When all modules have hit that phase point and have sent their part of the event filter message, all will proceed automatically. To handle the dead module case, the TM-Bus Master has a timer going so that if the system gets stuck at an event point, he will time out and handle the failure condition. However, this timer can have a long time-out value to be sure the module is really dead because its use is not the normal case. This is the typical "barrier" application. (2) The second example is the distribution of external I/O data (say MIL-STD-1553) to a system consisting of several backplanes, referred to as a cluster. For example, a system might have two different 1553 modules distributing data from four 1553 busses. For some systems it is desirable that calculations for a specific I/O cycle to proceed on all the cluster modules only when all 1553 data has been distributed from all 1553 channels. One approach would be to have an interrupt on each distribution message arriving at each module, and use some logic to figure out when they're all there.

3.3 (Continued):

With today's higher bus data rates it has become imperative that the transmission line characteristics of the backplane be accounted for in backplane analysis. By doing so it becomes apparent that there may be better ways to drive backplanes than with traditional TTL family logic. To solve some of the inherent problems with running densely loaded buses, the Pi-Bus structure was developed. Pi-Bus was developed by IBM, Unisys and TRW for the VHSIC 2.2 Interoperability program.

b. Transmission Line Physics: Several problems must be overcome to run a faster, more heavily loaded bus. When the bus propagation delay plus setting time decreases below about 100 ns the backplane must be considered as a transmission line. If the backplane is densely populated the problem of running it at high data rates is exacerbated.

Z_O is the characteristic impedance of the backplane, and is calculated from the physical characteristics of the backplane. From Figure 13, a simple model of a transmission line, the following equation may be written:

$$Z_o = [(R + jwl) / (G + jwc)]^{1/2} \quad \text{(Eq.1)}$$

For higher frequency, the R and G terms become insignificant due to the increases in the ω terms,

$$w = 2\pi f, \text{ where f is frequency} \quad \text{(Eq.2)}$$

and the equation simplifies to:

$$Z_o = (L / C)^{1/2} \quad \text{(Eq.3)}$$

where:

L = is in units of inductance per unit length
C = is in units of capacitance per unit length

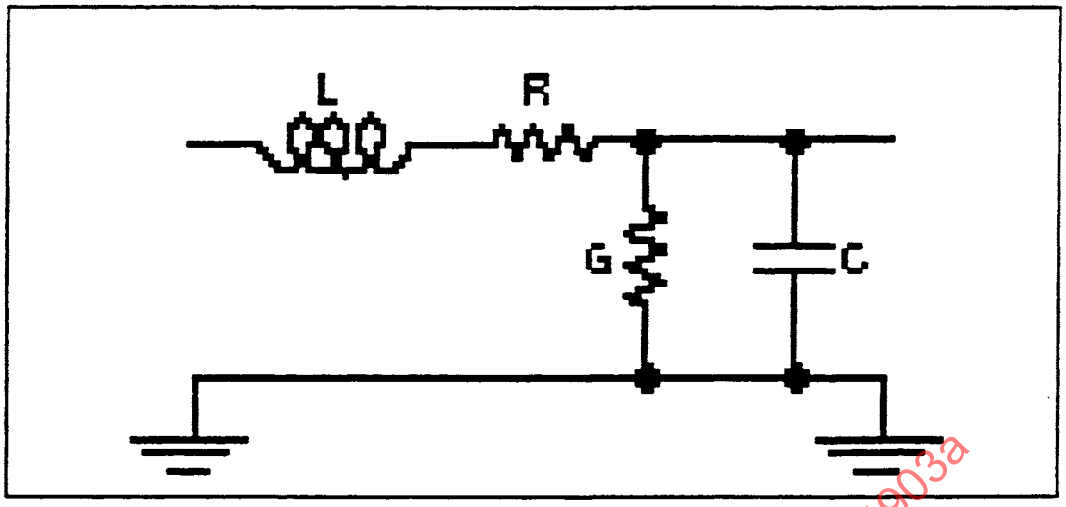

FIGURE 13 Simple Transmission Line Model

3.3 (Continued):

c. Effects of Capacitive Loading on Bus: Values for Z_O can be calculated from the physics of the backplane construction. For a typical Pi-Bus epoxy-glass backplane the value for Z_O is about 65W. The basic equation describing the impedance of a loaded bus is:

$$Z_1 - Z_o / (1 + C_1 / C)^{1/2} \quad \text{(Eq.4)}$$

where:

C_1 = is the additional load added to the bus by modules, connectors and connector mounting vias

So as the capacitive load of the modules increases, the impedance of the bus decreases. As the impedance of the bus decreases, the current required to drive the bus between state changes in a given time period increases (1 = V/Z). The equation for the propagation delay of a signal on an unloaded transmission line is:

$$T_{PO} - 1(L / C)^{1/2} \quad \text{(Eq.5)}$$

where:

1 = length of the bus.

For a loaded bus:

$$T_{P1} - T_{PO}(1 + C_1 / C)^{1/2} \quad \text{(Eq.6)}$$

3.3 (Continued):

describes propagation delay on a transmission line. From the equation for the loaded bus, it is obvious that as the capacitive loading of the bus increases, the propagation delay for a given length of bus increases. As this time increases bus setting time increases, and hence affects how fast the bus may be operated. So a bus transceiver, which had low capacitive loading, would improve bus operation in several ways. A low capacitive transceiver would raise the value of Z_1, and hence the drive current requirements for a given performance would be reduced, the propagation delay for a given length of backplane would be reduced, and any bus settling time required would also be reduced.

d. Output V_{OH} Swing: The standard TTL swing for an output is from 0.2 V to V_{CC} - two diodes (for CMOS OV to V_{SS}). For a bipolar device, this swing could be as much as from 0.2 to 4.1 V. The smaller this swing between high and low, the less charge which must be moved in any given time to effect a change of state. So from the standpoint of power usage, smaller in this case would be better.

Pi-Bus was developed to address both of these conclusions in a manner similar to Future Bus. The bus side outputs are open Schottkys, and the bus is terminated on both ends by a resistor to a voltage source with limits of 1.9 to 2.1 V (Figure 14). The resistors used to terminate the bus may be from 30 to 40 Ω. By terminating the bus in this way, (Z_0 matches a 40 Ω termination with no loads on the bus, but including loading from raw backplane plus vias and mating connectors, and a 30 Ω. termination matches a fully loaded Pi-Bus backplane) the bus is somewhat tuned to the Z_1 of the bus so that reflections can be minimized. By terminating the bus to a maximum of 2.1 V, the maximum voltage swing on the bus will perform V_{OL} minimum spec of 0.4 V to the maximum 2.1 V. Thus the voltage swing for the Pi-Bus transceiver would be 1.7 V versus the 3.9 V for a standard bipolar transceiver.

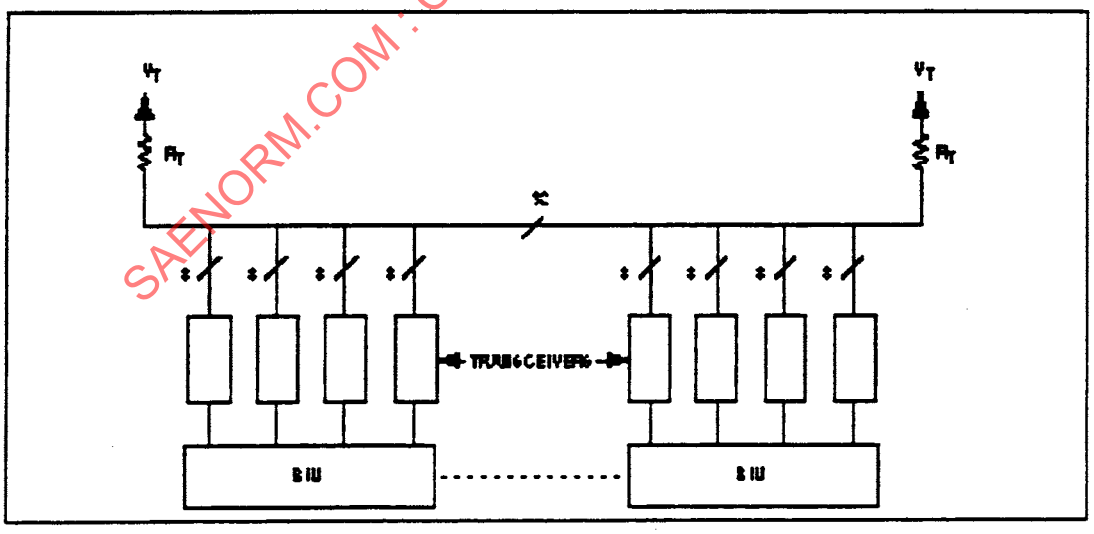

FIGURE 14 - Output V_{OH} Swing

3.3 (Continued):

e. Output Capacitance of Transceiver: In order to decrease the capacitive loading of each transceiver, Schottky diode outputs are used, and are reverse biased when not active low. The Pi-Bus driver output is shown in Figure 15(a). When the output is in a high state both the output Schottky diode and the collector-substrate diode of Q1 are reverse biased. By reverse biasing both the Schottky diode junction and the collector-substrate junction, parasitic capacitive loading of the bus driving output can be greatly decreased. The following equation gives an approximation of the impact reverse biasing has upon junction capacitance:

$$C_J = C_{J0} / (1 + V_d / V_0) \quad \text{(Eq.7)}$$

where:

C_{J0} = equals the capacitance of the diode junction under unbiased conditions
V_d = equals the reverse biased voltage of the diode junction
V_0 = is the built-in zero bias potential across a diode junction, and would be on the order of 0.7 V

So, by reverse biasing the cathode of both D1 and Q1, their parasitic capacitive loading on the bus is decreased. If the bus were at 2.0 V and VCC was at 5 V, C_{J0} would be decreased by a factor of 2.1. Then of course, C1 and C2 are in series, so their total capacitance would be decreased according to the following equation:

$$C_T = C_P + [(C_1)(C_2 + C_3)] / (C_1 + C_2 + C_3) \quad \text{(Eq.8)}$$

The capacitance of C3 (from D3) is basically negligible in comparison to C1 because it is approximately 0.5% the area Q1. The package capacitance (Cp) may vary from 0.3pF to 1.5pF, depending upon the package and the pin location on the packages. The other capacitive loading on each bus pin would be the base-collector and base emitter capacitance of Q7 (Figure 17(b)), which connects to the anode of D1. For a typical TTL part (Figure 16), the capacitive loading of the output would be primarily C5 because C6 is in series with C7 and the parallel combination of C8 and C9. For Q4 and Q5 being about the same size devices, the capacitive loading of the TTL circuit would be several times larger than that of the Schottky terminated circuit.

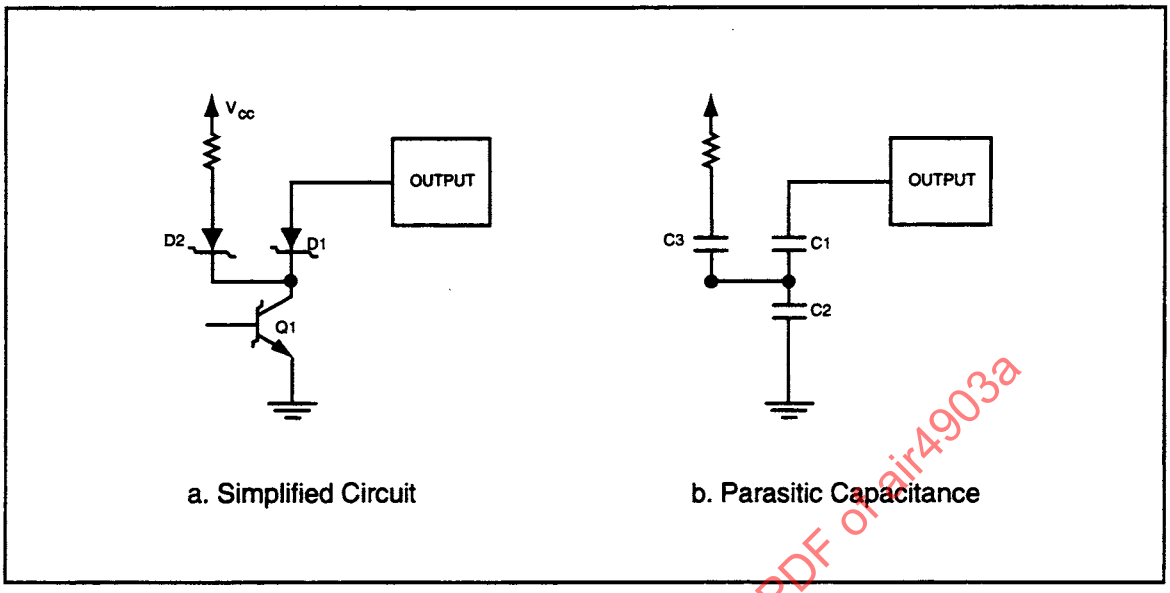

FIGURE 15 - Pi-Bus Driver Output

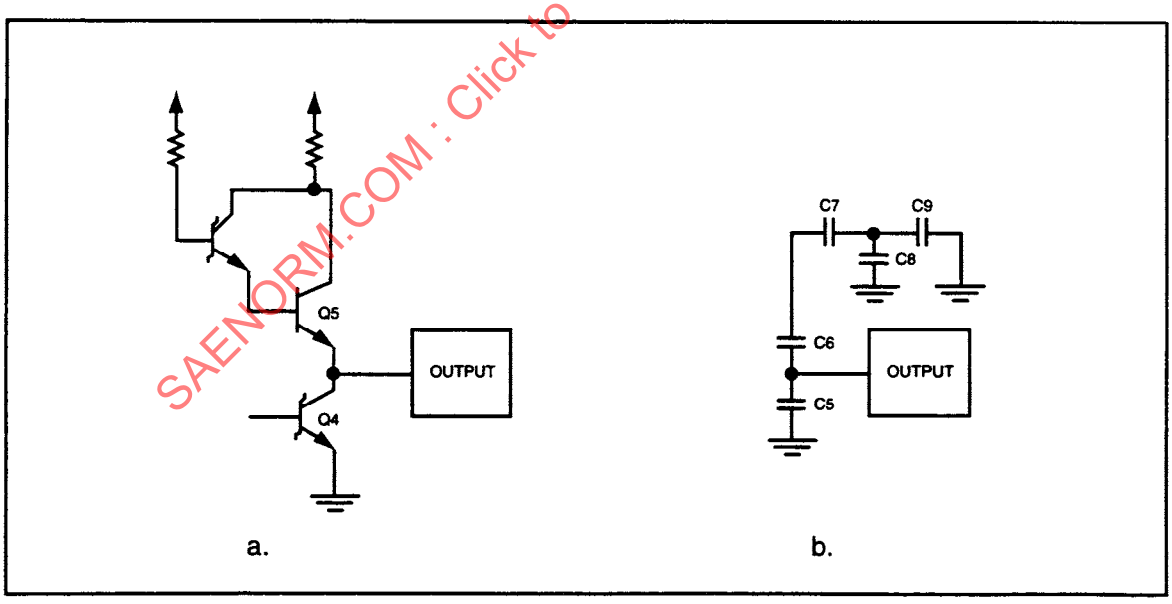

FIGURE 16 - Capacitance Loading

4.3.4.1.2 (Continued):

However, if distribution messages from a single 1553 module are placed in a Pi-Bus chain ending in an event filter message, all modules can know that all the data sent via the chain has been distributed with only one interrupt, on the event filter message itself. This is more of a duty cycle saving application, but it still is acting like a barrier.

B: Event Filter Label size is restricted to 12-bits.

Rationale: Figure 12, page 60, of AS4710 shows there are 12-bits of label for the event filter type. This restriction was arbitrarily chosen to be in agreement with the PIU that first implemented event filter. There is no technical reason to disallow the event filter label being 16-bits.

4.3.4.2 Block Message - Short Header (SH) Sequence:

Description: States that both label and direct block messages are supported, and the ability of a module to accept a particular message block message type is as defined in the module specification.

Rationale: See 4.2.3.2.1.3.A.

4.3.4.2.1 Label Addressing:

A: Definition of the header words for a block message short header using label addressing.

Rationale: The Header words are defined in the same way all the JIAWG DEM/VAL vendors define them. In order for the BIUs to know how many words are being sent/received, one of the header words must contain the count of the words to be transferred. HWB is required by Pi-Bus standard. Note that if the message is suspended, when it is resumed, HWB contains the number of words of data remaining to be transferred. One of the header words also needs to contain the label to be sent. HWC0 was arbitrarily chosen for this. In order to support suspend messages, when a suspended message is resumed, a count must be included in the headers to signify how many words had already been transferred. Since this field was necessary, it was noted that this field is really an offset and could also be used as such for the message when initially sent. HWC1 was arbitrarily chosen for this field.

B: What label means.

Rationale: The label definition is based on the various JIAWG DEM/VAL label definitions. A label is logically an extension of the Pi-Bus Slave ID. Thus, if the label is not available, then the Slave drops off the bus, without resulting in an error. The maximum buffer size is included to support software debug and security. Since a label is a way the Slave can specify where to place the incoming data without the Master having to know this, the location to place the data must be associated with the label.

4.3.4.2.2 Direct Addressing:

Description: Defines block message-short header direct addressing message headers.

Rationale: See 4.2.3.2.1.3.A.

4.3.4.3 Block Message - Extended Header Sequence:

Description: States that both label and direct block messages with extended headers are supported.

Rationale: See 4.2.3.2.1.3.A.

4.3.4.3.1 Label Addressing:

Description: Defines block-message extended header label addressing message headers.

Rationale: Same as rationale in 4.3.4.2.1.

4.3.4.3.2 Direct Addressing:

Description: Defines block-message extended header direct addressing message headers.

Rationale: See 4.2.3.2.1.3.A.

4.3.4.4 Bus Interface Message Sequence:

Description: Defines header words for Bus Interface Messages, states particulars about data link Bus Interface Messages, and states particulars about system time Bus Interface Messages.

Rationale:

A: The rationale for continuing after writes attempted to Reserved or write protected data link registers during a bus interface message and reporting an error to the Master is to lessen chip complexity. Allowing modification of the registers agrees with the way block messages are treated in that if an error occurs during a block message, memory may be modified even though the message was not completed successfully.

B: The Pi-Bus system timer was added to provide a backplane synchronized global time capability. Including this capability would save including additional signal lines (e.g., a discrete) to support such a capability. The rationale for a 48 bit instead of a 32 bit Pi-Bus system timer with a 1 usec resolution was based on a survey of real-time software experts. All surveyed said that the clock should be at least 48 bits to avoid complications in software having to deal with roll-over. A one microsecond tick was chosen since it was the same granularity as time being distributed over the JIAWG High Speed Data Bus.

4.3.4.4.1 Data Link Layer Registers:

Rationale: Provided to allow initialization and control over a portion of the Pi-Bus hardware. Refer to 4.3.7.

4.3.4.4.2 System Timer:

Rationale: Refer to 4.3.7.4.

4.3.4.5 Datagram Message - Short Header Sequence:

Rationale: See 4.2.3.2.1.3.E.

4.3.4.5.1 Nonacknowledge:

Rationale: This sequence eliminates both header and data acknowledge to provide a true datagram service. This is the most efficient message transfer sequence.

4.3.4.5.2 Acknowledge:

Rationale: This sequence eliminates head acknowledge while still providing an acknowledge of message completion. This is the most efficient block message transfer sequence when a message completion acknowledgment is required.

4.3.4.6 Datagram Message - Extended Header Message Sequence:

Rationale: Rationale not provided.

4.3.4.6.1 Nonacknowledge:

Rationale: Same as 4.3.4.5.1 for extended header messages.

4.3.4.6.2 Acknowledge:

Rationale: Same as 4.3.4.5.2 for extended header messages.

4.3.5 Exception Sequences:

Rationale: Not applicable, since no requirements are specified.

4.3.5.1 Suspend:

A: Support suspend as either a Master or Slave.

Rationale: The Pi-Bus is intended to be used in real time applications. The ability to establish and meet task deadlines require that messages are suspendable. In order to suspend a message, both the Master and Slave must support the suspend operation.

4.3.5.1 (Continued):

B: Modules may elect to specify messages as not suspendable as either a Master or Slave.

Rationale: There may be messages whose data must always be transmitted together. It may be the case that if part of the data is delayed, it must all be recalculated. These messages must be aborted instead of suspended. The Master or Slave must be capable of identifying these messages and preventing their suspension.

C: A suspend sequence shall not be started prior to the expiration of Vie Timer A.

Rationale: Several studies have shown that in real time avionic systems, the majority of the Pi-Bus traffic will be composed of short chains containing short messages. It is expected that most of these chains can complete their operation before the Vie Timer A expires. Allowing the chain to complete is the most efficient use of the Pi-Bus and, since the timer has not expired, there is no danger of missing a deadline.

The only requirement in real time operation is to meet the deadlines. There is no benefit in exceeding them. Since the scheduling must consider the worst case bus access condition of both Vie Interval Timers A and B timing out, the full bus time should be utilized in an attempt to complete the chain and save the time required by the suspend and resume operation.

D: Suspend if data remaining is equal or greater than 8 words.

Rationale: The suspend operation requires more than just the suspend sequence on the Pi-Bus. The Master or Slave must calculate the Resume Control Words and may be required to store information relative to the suspend operation in main store (for example, Slave label table updated). Both these and perhaps other operations require time, non-transfer cycles. If the overhead of the suspend operation is greater than the data remaining in the message, the Pi-Bus utilization would be better served by completing the message than suspending. While the actual number of cycles required will depend upon the actual implementation of the Master and Slave involved, it will be greater than three, and likely even greater than eight. Thus raising the remaining count requirement from three to eight increases the performance of the Pi-Bus.

E: Do not resume if error detected.

Rationale: When an error occurs, a typical error recovery procedure includes retransmission. The message sequence is allowed to continue to permit the Master to obtain the acknowledge word from the Slave to identify the error. The error type may be a factor in the error recovery procedure. The suspend sequence causes the Master to obtain the acknowledge word. The message can be ended at this stage and error recovery procedures initiated.

4.3.5.1.1 Single Slave Suspend:

A: Short header, short resume.

Rationale: A Block Message, Short Header, is a basic Pi-Bus message type. This message can be either multicast or singlecast. The Resume Control Words had to be defined for the multicast case since the Master must form them and all Slaves must use them. Once they were defined for the multicast case, is reasonable to use the same definition for the singlecast case.

B: Short header label Addressing Suspend.

Rationale: Specifying the original label and the number of words transferred from that label buffer in the Resume Control Words provides all of the necessary data to continue or resume a message that was suspended. The count of the number of words remaining to be transferred is calculated from the original transfer count and the actual transferred count from the Resume Control Words by the Master. This remaining count is then provided in the HWB of the resumed message to allow the remaining words to be transmitted. Since this is the data specified for a multicast operation, it is reasonable to also use it in a singlecast operation.

C: Short header direct Addressing Suspend.

Rationale: Specifying the next data address in the Resume Control Words provides all of the necessary data to continue or resume a message that was suspended. The count of number of words remaining to be transferred must be determined by the Master. This remaining count is then provided in the HWB of the resumed message to allow the remaining words to be transmitted. Since this is the data specified for a multicast operation, it is reasonable to also use it in a singlecast operation.

D: Extended header, extended resume.

Rationale: A Block Message, Extended Header, is a basic Pi-Bus message type. This message can be either multicast or singlecast. The Resume Control Words needed to be defined for the multicast case since the Master must form them and all Slaves must use them. Once they were defined for the multicast case, is reasonable to use the same definition for the singlecast case.

E: Extended header label Addressing Suspend.

4.3.5.1.1 (Continued):

Rationale: The last two Resume Control Words are defined the same as the short header Resume Control Words. This provides the required data for resuming a suspended message. The count of the number of words remaining to be transferred is calculated from the original transfer count and the actual transferred count from the Resume Control Words by the Master. This remaining count is then provided in the HWB of the resumed message to allow the remaining words to be transmitted. The first six Resume Control Words are the corresponding words from the message header. These six words will be provided to the Slave's host at the completion of the message regardless of whether or not the message was suspended. Since this is the data specified for a multicast operation, it is reasonable to also use it in a singlecast operation.

F: Extended header direct Addressing Suspend.

Rationale: The last two Resume Control Words are defined the same as the short header Resume Control Words. This provides the required data for resuming a suspended message. The count of number of words remaining to be transferred must be determined by the Master. This remaining count is then provided in the HWB of the resumed message to allow the remaining words to be transmitted. The first six Resume Control Words are the corresponding words from the message header. These six words will be provided to the Slave's host at the completion of the message regardless of whether or not the message was suspended. Since this is the data specified for a multicast operation, it is reasonable to also use it in a singlecast operation.

G: Extended header, short resume treated as unsuspendable.

The Resume Control Words for an extended header suspend have already been defined for the JIAWG platforms. A Slave must follow those definitions to be suspendable. The response in the header acknowledge word is allowed but if a Bus Request is asserted during this message, a suspension will not be attempted when Vie A Interval Timer expires. The message will be aborted if Vie B Interval Timer expires.

H: Short header, extended resume treated as unsuspendable.

The Resume Control Words for a short header suspend have already been defined for the JIAWG platforms. A Slave must follow those definitions to be suspendable. The response in the header acknowledge word is allowed but if a Bus Request is asserted during this message, a suspension will not be attempted when Vie A Interval Timer expires. The message will be aborted if Vie B Interval Timer expires.

4.3.5.1.2 Multiple Slave Suspend:

A: Short header, short resume.

4.3.5.1.2 (Continued):

Rationale: A Block Message, Short Header, is a basic Pi-Bus message type. The Resume Control Words must be defined for the multicast case since the Master must form them and all Slaves must use them. If they are not defined, various modules would not be interoperable.

B: Short header label Addressing Suspend.

Rationale: Specifying the original label and the number of words transferred from that label buffer in the Resume Control Words provides all of the necessary data to continue or resume a message that was suspended. The count of the number of words remaining to be transferred is calculated from the original transfer count and the actual transferred count from the Resume Control Words by the Master. This remaining count is then provided in the HWB of the resumed message to allow the remaining words to be transmitted.

C: Short header direct Addressing Suspend.

Rationale: Specifying the next data address in the Resume Control Words provides all of the necessary data to continue or resume a message that was suspended. The count of number of words remaining to be transferred must be determined by the Master. This remaining count is then provided in the HWB of the resumed message to allow the remaining words to be transmitted.

D: Extended header, extended resume.

Rationale: A Block Message, Extended Header, is a basic Pi-Bus message type. The Resume Control Words must be defined for the multicast case since the Master must form them and all Slaves must use them. If they are not defined, various modules would not be interoperable.

E: Extended header label Addressing Suspend.

Rationale: The last two Resume Control Words are defined the same as the short header Resume Control Words. This provides the required data for resuming a suspended message. The count of the number of words remaining to be transferred is calculated from the original transfer count and the actual transferred count by the Master. This remaining count is then provided in the HWB of the resumed message to allow the remaining words to be transmitted. The first six Resume Control Words are the corresponding words from the message header. These six words will be provided to the Slave's host at the completion of the message regardless of whether or not the message was suspended.

F: Extended header direct Addressing Suspend.

4.3.5.1.2 (Continued):

Rationale: The last two Resume Control Words are defined the same as the short header Resume Control Words. This provides the required data for resuming a suspended message. The count of number of words remaining to be transferred must be determined by the Master. This remaining count is then provided in the HWB of the resumed message to allow the remaining words to be transmitted. The first six Resume Control Words are the corresponding words from the message header. These six words will be provided to the Slave's host at the completion of the message regardless of whether or not the message was suspended.

G: Extended header, short resume not possible.

The Resume Control Words for an extended header have already been defined. It required the extended resume.

H: Short header, extended resume not possible.

The Resume Control Words for a short header have already been defined. It requires the short resume.

4.3.5.1.3 Resuming Suspended Messages:

A: BIU Master goes idle after suspend sequence.

Rationale: Going idle after a suspend sequence allows vieing for a new master.

B: Revie to resume a suspended message.

Rationale: A module must reacquire Mastership of the Pi-Bus before it can initiate or resume any messages. The only procedure permitted for this is the vie sequence.

4.3.5.1.4 Using Suspend with Vie Intervals A and B.

A: Suspend using Vie Interval Timers A and B.

Rationale: See rationale for 4.3.5.1.

4.3.5.2 Abort:

A: ACK response to an Abort.

Rationale: The Abort sequence, as described in AS4710 requires Slaves to assert ACK on the AB2 cycle of the Abort sequence. However if there are no Slaves at the time of the Abort, either no modules had been addressed or all addressed modules cease participation in the message before the Abort, no ACK will be asserted. There are occurrences where the Master will perform an abort and no Slaves were selected. This results in NS on the AS lines during the third cycle of abort. This is not considered an AS sequence error because this is normal operation.

4.3.6 Wait:

A. Wait allowed.

Rationale: Allows Master or Slave to control the data transfer rate on Pi-Bus to accommodate slow or temporarily busy devices.

4.3.6.1 Rules for Asserting Wait.

A1: Slave may assert Wait on last acknowledge cycle.

Rationale: The final status on the current message is not available to the bus interface unit of the Slave till two cycles after the last acknowledge cycle. This provides the Master time to issue an abort following the acknowledge. A Master may allow a message to complete following the detection of an error in order to obtain the acknowledge words for analysis. It will then issue the abort to notify the Slave to disregard the message.

A2: Wait rule one

Rationale: "Wait may normally be asserted by a Bus Interface only if that module is scheduled to be a bus Master or Slave on the next transfer cycle. However, the current bus Master may assert Wait if the bus will be placed in the Idle state following the Wait induced non-transfer cycle(s) provided the vie interval requirement in not violated."

This rule establishes that only those modules involved with a message are permitted to assert a Wait. The Master is permitted to assert a Wait on any cycle of the message. The following are typical times when a Master might assert a Wait:

HAn: While the initial data is being fetched for a receive message.
D: During the Data cycles if the memory interface cannot keep up with the message data rate.
DZ: While final data is being stored for a transmit message.
DAn: Allow for checking for a late NAK, storing an interrupt report and fetching the control block for the next message.

4.3.6.1 (Continued):

The Master is allowed to assert Wait on the last Acknowledge cycle of the message because it will either continue the Pi-Bus activity with another message or permit the Pi-Bus to go Idle on the next transfer cycle.

The Slave is also permitted to assert a Wait on any cycle of the message. The following are typical times when a Slave might assert a Wait:

HAZ: To permit a label entry fetch and check.
HAn: While the initial data is being fetched for a transmit message.
D: During the Data cycles if the memory interface cannot keep up with the message data rate.
DZ: While final data is being stored for a receive message.
DAn: Allow for checking for a late Abort, updating a label table entry and storing an interrupt report.

Note that normally a BIU might not be considered to be scheduled as a Slave on any cycle following the last Acknowledge cycle of the message. However, a Slave may be required to monitor the Pi-Bus for several cycles following the message to check for a late Abort sequence initiated by the Master. If such an Abort is detected, the previous message must be ignored. This means that the label table entry for that message cannot be changed and interrupt reports cannot be stored until a number of cycles after the end of the message. In this sense, the BIU is still a Slave and must assert Wait on the last Acknowledge cycle of a message to hold off the next Pi-Bus message until it is finished with the first message. Note that the Wait does not hold off or effect an Abort sequence.

A3: Wait rule two

Rationale: "A Bus Interface shall not assert Wait on H0 or H1 cycle."

This rule requires the Master to transmit the H0, H1 and HZ or the H0, H1 and H2 sequences without a nontransfer cycle. Since the HZ or H2 cycle is the earliest cycle that a Wait can be asserted, the following cycle (the HA or H3 cycle) is the first cycle that can be delayed.

A Bus Interface cannot be a Slave until it decodes its address in the HWA which it received on the H0 cycle. A module Bus Interface cannot assert a Wait on H0 or H1 to gain time to determine whether or not it is being selected by the HWA.

A4: Wait rule three

Rationale: "A Bus Interface shall not assert Wait on a particular cycle (N), if on the second previous cycle (N-2) the module did not assert Wait, but Wait was asserted on that cycle. If the Bus Interface does assert Wait on cycle N and Wait was not asserted by any other module on cycle N-1, the Bus Interface shall assert Wait for an even number of contiguous cycles."

4.3.6.1 (Continued):

Wait rule three makes it possible for a Bus Interface to assert a Wait on any desired Pi-Bus transfer cycle. The first part of the rule does not permit a Bus Interface to assert a Wait on cycle N if it did not previously assert a Wait on cycle N-2 and a Wait had been asserted by another Bus Interface. The "N-2" Wait rule is illustrated in Figure 18. The figure shows a Bus Interface which requires a Wait to be asserted on word D5 to delay the transfer of word D6. This time may be required to fill or empty a data buffer before continuing. If a second Bus Interface asserts two Waits starting on word D3 to delay word D4, then D5 will be delayed by two cycles. The "N-2" rule inhibits the first Bus Interface from asserting a Wait until cycle 6. This is exactly when it is needed to delay word D6. In summary, the "N-2" rule allows different Bus Interfaces to delay two words on the Pi-Bus which are separated by one word.

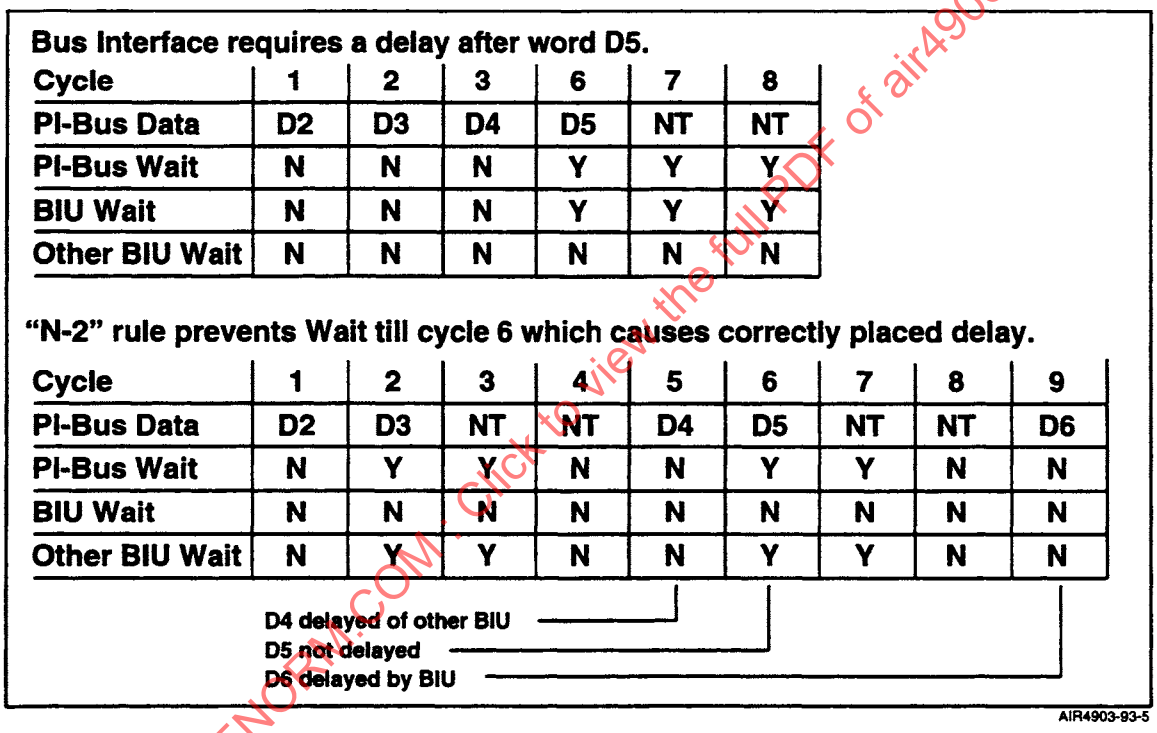

FIGURE 18 - "N-2" Wait Rule

The second part of Wait rule three requires the Bus Interface to check for a Wait on the cycle before it asserted a Wait. Because of the pipeline effect of the Pi-Bus, this check is too late to impact the assertion of the Wait. However, if a Wait was asserted on the previous cycle (and not by this Bus Interface) then the Wait must be de-asserted for one cycle. The Wait can be attempted again on the next cycle. The "N-1" rule is illustrated in Figure 19. The first Bus Interface needs to assert a Wait on word D4 to delay word D5 for two cycles. However, a second Bus Interface may assert Wait on word D3 (cycle 2) to delay word D4 for four cycles.

4.3.6.1 (Continued):

The first Bus Interface will assert a Wait on cycle 3 while it is checking the Wait lines from cycle 2. It will determine that a second Bus Interface asserted a Wait on that cycle and will remove its Wait on cycle 4. Since it still needs the Wait on word D4 and the "N-2" rule is met, the first Bus Interface can again assert Wait on cycle 5. The results of checking Wait for cycle 4 again requires that the first Bus Interface removes the Wait on cycle 6. The assertion of Wait on cycle 7 is finally successful and word D5 is delayed by two cycles. Thus the "N-1" Wait rule allows different Bus Interfaces to delay consecutive words on the Pi-Bus.

The third part of Wait rule three, an even number of Waits must be asserted, is required for the "N-1" rule to be successful in permitting consecutive words to be delayed. If an odd number of Waits had been asserted by the second Bus Interface in the above example (no Wait on cycle 5), the first Bus Interface would not have been able to assert a Wait on word D4. Instead, the Wait would have been asserted on D5 and D6 would have been delayed, not D5 as required.

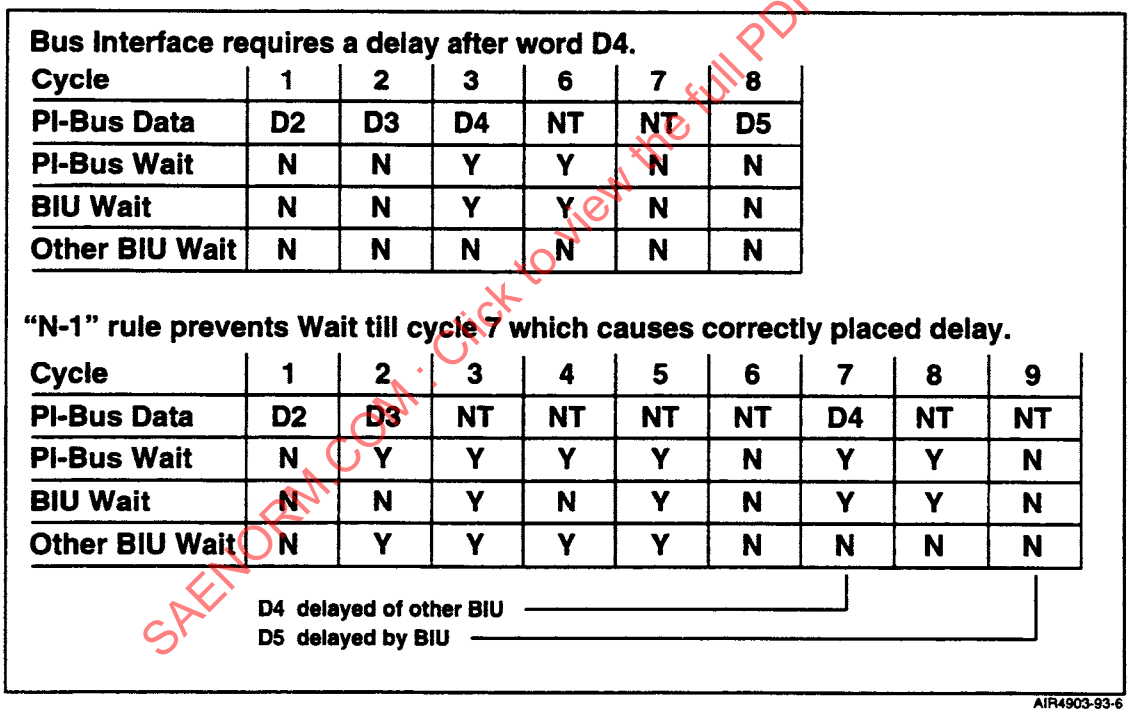

FIGURE 19 - "N-1" Wait Rule

4.3.6.2 Effects of Wait:

A. Abort during Wait.

Rationale: The Abort sequence must be altered for Abort to keep Master and Slave(s) in sync and to end a possibly hung message. Any cycle on which the Abort cycle type appears can be interpreted as a transfer cycle regardless of the state of Wait.

4.3.6.2.1 Line Groups During Nontransfer Cycles:.

4.3.6.2.1.1 Data line group during non-transfer cycles.

A: Only valid symbols shall be posted on the Data line group.

Rationale: A valid symbol is any value with correct parity or EC code. The Data lines will be checked for valid parity or EC code during the non-transfer cycles. An invalid symbol causes an uncorrectable or correctable Data line error to be detected. This will result in the termination of the message with an Abort.

4.3.6.2.1.2 Cycle Type Group During Nontransfer Cycles:

A: Only valid symbols shall be posted on the Cycle Type group.

Rationale: A valid symbol is any value with correct parity or EC code. The Cycle Type lines will be checked for valid parity or EC code during the non-transfer cycles. An invalid symbol causes an uncorrectable or correctable Cycle Type line error to be detected. This will result in the termination of the message with an Abort.

B. Master asserts valid symbols during NT cycles.

Rationale: The Master module must source valid (no line errors) symbols on the CT lines during NT cycles since line error checking is still active.

4.3.6.2.1.3 AS Group During Nontransfer Cycles:

A. Assert valid symbols during NT cycles.

Rationale: The Slave module(s) must source valid (no line errors) symbols on the AS lines during NT cycles since line error checking is still active.

4.3.6.2.1.4 Wait Lines During Nontransfer Cycles:

A. Asserting Wait during NT cycles is done the same as during Transfer cycles.

Rationale: Waits cause NT cycles. Therefore, waits during NT cycles are treated no differently than waits during Transfer cycles.

4.3.6.2.1.5 Bus Request Lines During Nontransfer Cycles.

B. Master asserts valid symbols during NT cycles.

Rationale: The Master module must source valid (no line errors) symbols on the BR lines during NT cycles since line error checking is still active.

4.3.7 Data Link and System Time Facilities:

A: A Master and Slave shall implement all Data Link Facilities.

Rationale: Interoperability is the goal of the standard. Requiring all modules to implement all Data Link Facilities forces interoperability at this level.

4.3.7.1 Data link register address space.

Rationale: Rationale not provided.

4.3.7.2 Register Protection:

A: The BIU shall support write protection.

Rationale: A Pi-Bus system can be configured in one of two basic ways. The first utilizes a Master module approach. This Master module controls the general task utilization of the other modules. In this approach, the Master module would need access to the Data Link Facilities of all of the modules to control the system.

The second approach is a distributed control. In this approach, each module would manage its own Data Link Facilities and write protect them so that another module could not change them.

4.3.7.3 Registers:

A: Data with line error not transferred.

Rationale: The Data Link Layer registers are implemented in hardware and their contents immediately effect the operation of the Bus Interface Unit. While a Bus Interface Message which writes these registers does intentionally change them, the change should only take place if the data is received by the BIU without error.

4.3.7.3.1 Reserved Register - Address 0:

A: Why this register was changed from Multicast Acknowledge Register in VHSIC Pi-Bus Specification to a reserved register in AS4710.

Rationale: This register was called the Multicast Acknowledge Register in the VHSIC Pi-Bus Specification. The Multicast Acknowledge Register stores error information on multicast messages for which this module was a Slave. In order for the current bus Master to access this information, it must send a Bus Interface message to the module requesting it. This may require host intervention to set up the message. However, this register will be overwritten by the next multicast message addressed to the module. Therefore this register is only useful under very controlled conditions. Because of this, this register was not considered useful and was not included in AS4710.

4.3.7.3.2 Control Register - Address 1:

A: Permanently write protected.

Rationale: It is not considered safe to allow reset of a Pi-Bus controller through the backplane communication bus (i.e., Pi-Bus). A module should perform the reset of the Pi-Bus controller(s) itself or through a reset command via a test and maintenance bus (e.g., TMBus). Also, there is no way to report results through the Pi-Bus if reset is unsuccessful since the bus controller would be off line in this case. Therefore, this register was permanently write protected, which basically made the register Reserved.

4.3.7.3.3 Module Capabilities Register - Address 2:

Rationale: Provides standard location to query capabilities of a module.

4.3.7.3.4 Vie Interval A Register - Address 3:

A. Use of this register discussed in 4.3.3.3 and 4.3.5.1.4

4.3.7.3.5 Vie Interval B Register - Address 4:

A. Use of this register discussed in 4.3.3.3 and 4.3.5.1.4

4.3.7.3.6 Vie Priority Register - Address 5.

A. Use of this register discussed in 4.3.3.3.

4.3.7.3.7 Reserved Registers - Addresses 6 to 32:

Rationale: Reserved registers provided for future growth and placement chosen to keep contiguous with other control registers.

4.3.7.3.8 Logical Slave ID Registers - Addresses 33 to 255.

A: All Logical Slave ID registers shall be provided.

Rationale: Interoperability is the goal of the standard. Requiring all modules to implement all of the Logical Slave ID registers forces interoperability at this level. This is already a small number of logical addresses for many systems, where a minimum of 223 logical IDs are required.

4.3.7.4 System Time:

A: 48-bit, 1 MHz timer.

Rationale: Many systems have a need to distribute time accurately. Time can be distributed between racks or boxes with a fiber optic high speed data bus. However, all of the modules within a rack or box also need access to the same reference time. By placing a timer within the Data Link Layer of the Pi-Bus and providing a Bus Interface message for distributing it, this time is available to all modules. The overflow from a 48-bit, 1 MHz timer only occurs every 8.9 years. This should be sufficient for most missions. Limiting the timer to 32-bits would reduce the time between overflows to 1.19 hours. With this short of a time, the device would need to maintain a count of the overflows.

B: AT code = 100.

Rationale: AS4710 defines AT codes of 100 through 110 as available for implementation defined registers. The System Timer is an implementation defined register whose implementation is now required.

C: Synchronized within 4 microseconds for a 12.5 MHz Pi-Bus.

Rationale: The Pi-Bus system timer was included for deterministic global time distribution. Specifying the synchronization between the System Timers to within 4 microseconds (for a 12.5 MHz Pi-Bus) provides all modules with the same time with a reasonable amount of accuracy. The 4 microseconds allows time to synchronize the source System Timer with the Pi-Bus clocks, transmit it, and resynchronize to the destination System Timer for loading. This time does force the design to read the source System Timer when the Pi-Bus message is active on the bus and ready for data.

4.3.8 Initialization:

A: Capable of Built-In-Test:

Rationale: All modules must be capable of determining their health after powering up and before going on line. Part of the checking is the testing of the Pi-Bus interface logic. Since the module can test itself upon power up, it should be capable of performing self-test upon command.

4.3.8 (Continued):

B: Pi-Bus Transceivers Disabled While in BIT:

Rationale: Several modules may be performing test at the same time or while the Pi-Bus is active. The transceivers of the modules being tested must be disabled to prevent interference with other bus activity.

4.3.9 Error Detection, Recovery and Diagnostics:

Detection and Correction of Multiple Errors: The error correction codes chosen for the SAE Pi-Bus have the capability of correcting at least one error and will detect any double bit error. The code however, is designed to detect all triple bit errors. The code is also capable of detecting some percentage of triple bit errors as uncorrectable errors. Below is a list of each code studied and its ability to detect triple errors.

a. There are a total of 9880 possible triple errors with the SAE error correction code. This code will detect 6776 of these errors for a detection percentage of 68.6%.

In an effort to maintain the advantages of a single-error-correct, double-error-detect and package-error-detect code, while keeping to an eight transceiver design, a new EC code was developed specifically for the Pi-Bus applications by C.L. Chen and J.R. Burns of IBM. It has been modified so that all 40 bits of the code may be wired on eight transceivers, without any loss in the ability to detect a transceiver fail.

The original Pi-Bus EC code, as developed by C.L. Chen, was to put four data bits and one check bit on each transceiver. However, due to the requirements of allowing message transfer between 16-bit and 32-bit modules on the Pi-bus, two problems emerged:

a. Only three extra lines are left available on transceivers 5-8 for error correction; an EC code with one check bit per transceiver could not be implemented.

b. At least 29 lines need to be used on transceivers 1-4; without a modification to the EC code, only 27 lines would be used.

Therefore, Modified Pi-Bus EC code was designed to correct all single bit errors and detect all double bit errors. In addition, Modified Pi-Bus EC code will detect any error combination on a single transceiver - up to six errors on any of the eight transceivers. All of the transceivers will protect four data bits; four of the transceivers will protect one check bit, two will protect two check bits and two will protect no check bits.

4.3.9.1 Correctable Line Errors:

Rationale: Rationale not provided.

4.3.9.2 Uncorrectable Line Errors:

A: Adding HA0-1 to the bottom entry of table 5-35.

Rationale: An error on cycle HA0-1 will not be detected in time to be reported in the Header Acknowledge Word. It will be reported in the Data Acknowledge Word if there is one in the sequence.

B: Wait Line Error.

Rationale: The Wait function is in integral part of the message sequence. If it is not detected properly by all modules in the message, the modules will not be in sync. Therefore a NAK is asserted on a Wait Line Error. Interpreting the first Wait Line Error as a Wait would most likely cause a violation in the even number of Wait cycle rule and result in a Wait Sequence error. Therefore, a Wait Line Error causes a NAK and is interpreted as no Wait being asserted. A Master detecting a Wait Line Error will abort the message.

C: AS Sequence error during Vie.

An AS sequence error on vie is either a Recognize or an ACK on V0 or VZ0 or an ACK following VZ0 The most likely cause of this is the detecting module started the vie sequence one cycle too late or the only contending module posted a ACK instead of a Recognize. In either case, the sequence is incorrect and the Slaves set Master unknown to prevent participating in a message with the winning module.

D: Parity checking during Vie sequence.

Rationale: The vie sequence is unique in that while several modules may start asserting bits on the data bus, only those bit asserted by the final vie winner are meaningful. Line errors and invalid signal pairs on lines not from the winner of each vie cycle are irrelevant and are considered correctable. Likewise, the parity bit can be ignored.

4.3.9.3 Sequence Errors:

A: Becoming not selected.

Rationale: There are conditions under which a Slave cannot participate in a message. The Slave is then required to become not selected. If the occurs in a message in which the only addressed Slave becomes not selected, the Master will detect an AS sequence error.

4.3.9.3.1 Cycle Type Sequence Errors:

A: Check bus sequence against expected sequence.

Rationale: By checking that the bus Cycle Types stay within the allowed sequence (as defined by HWA), synchronization between Master and Slave modules is maintained.

4.3.9.3.2 Acknowledge Set, Wait and Bus Request Sequence Errors:

A: Odd number of Waits.

Rationale: An even number of Wait cycles is specified in AS4710 and is required for correct operation. The detection of a odd number of Wait cycles, except when caused by the start of an Abort sequence, is a serious error which should cause the assertion of a NAK.

4.3.9.4 Semantic Errors:

Rationale: Not applicable since no requirements are specified.

4.3.9.4.1 Header Semantic Errors:

A: Cease being Slaves.

Rationale: AS4710 allows modules to cease being Slaves in response to unspecified conditions. The standard defines those conditions.

4.3.9.4.2 Header and Data Acknowledge Semantic Errors:

Rationale: Rationale not provided.

4.3.9.5 Diagnostics:

Rationale: Not applicable since no requirements are specified.

4.3.9.5.1 On-Line Testing.

Rationale: Rationale not provided.

4.3.9.5.2 Off-Line Testing:

A: Parameter Write message in internal loopback.

Rationale: Internal loopback is a powerful technique for testing the Pi-Bus interface without requiring signals to be asserted onto the Pi-Bus. A minimum requirement for a module is to have the capability of transmitting and receiving the simplest message type (Parameter Write) in this mode.

4.3.9.6 Error Handling:

A: Error Table Defined Responses:

Rationale: Interoperability is the goal of the standard. Modules must have a common approach to handling error condition in order to achieve interoperability and interchangeability.

4.3.9.6 (Continued):

B: Programmable control for Master abort.

Rationale: A Master module needs to continue the message sequence following a Slave detected error (indicated by a NAK) to obtain the Acknowledge Word. The Acknowledge Word defines the error and may be used to determine the correct error handling procedure. After the Acknowledge Word is received, the Master can due one of two things. First, it could initiate an Abort sequence to indicate to all Slaves that the preceding message must be discarded. This approach is taken if the system design requires all Slaves to have the same data from this message. Second, it could proceed with the next message. This would allow those Slaves which correctly received the message to use it. Those Slaves which did not correctly receive the message would not be able to use it. The correct operation for the Master must be specified by its host.

4.3.9.6.1 Deviations from Scheduled Sequence:

A: Error during header or header acknowledge.

Rationale: The basic philosophy of the standards is that if the message header and header acknowledge are not correct, the message should be aborted. The Acknowledge Word will provide any available data for determining the error handling procedure. Since the message cannot be completed correctly, Pi-Bus bandwidth is not wasted on transferring data for a message which cannot be successful.

B: Line or sequence error.

Rationale: A line or sequence error is a catastrophic failure. The message cannot be completed successfully. The Master has all of the data necessary for an error handling procedure. The Pi-Bus is best served by aborting this message and allow the next message to proceed.

C: Local condition

Rationale: The design of any module will have local conditions which will not permit the successful completion of a Bus message. Examples of these conditions would be memory parity errors, memory protect violations, memory accessing errors, etc. Under these conditions, cleanly terminating the message with a Abort sequence both notifies the Slave of the error and minimizes the loss of Pi-Bus utilization.

D: Abort after message.

4.3.9.6.1 (Continued):

Rationale: An Abort sequence is utilized by the Master to specify that the current (previous) message must not be used. The abort may not be issued till after the Acknowledge Word is received. The Slave must wait for this late abort before completing its processing of the message. A specific time must be defined. The standard requires that if a late abort is issued, the first cycle of the abort must be on the Pi-Bus on or before five cycles following the last acknowledge cycle. The Slave must assert six cycles of Wait, starting with the last acknowledge cycle, to allow time to detect the abort.

E: Tenure Pass or undefined message type.

Rationale: The standard does not permit the use of Tenure Pass or undefined message types. A Master must not initiate these sequences. Instead, an error is reported to the host and the Pi-Bus is released.

4.3.9.6.2 Slave Response to Exception Conditions:

A: Error in header.

Rationale: The basic philosophy of standards is that if the message header is not correct, the Slave should not continue in the message sequence. The error will be reported in the Acknowledge word and cause the assertion of a NAK. The header Acknowledge Word will provide any available data for determining the error handling procedure. Since the message cannot be completed correctly, the Slave module assets are not wasted on a message which cannot be successful.

B: Error other than cycle sequence error.

Rationale: Errors detected during the data cycles are posted in the Acknowledge Word and cause the assertion of a NAK. The Slave continues with the message sequence to allow the Master to obtain the Acknowledge Word.

C: Cycle type sequence error.

Rationale: The cycle type signals are used to insure that the message sequence and Slave sequence are in sync. A cycle type sequence error indicates that the Slave no longer is in sync with the bus and cannot correctly continue with the message. Therefore, it asserts a NAK and ceases participation as a Slave.

4.3.9.6.3 Slave Cease Sequence Participation:

A: Cease participation without signaling an error.

Rationale: The logical address space of the Pi-Bus (223 logical addresses) is not sufficient for most systems. A label mode of Block Messages was developed to overcome this short coming. The label, in the header, is treated as an extension of the logical Slave ID. If a module is not enabled for a specific label, it should not participate in the message. Signaling an error would not allow the message to successfully complete with those modules which were enabled for the label.

4.3.9.6.4 BIU Response to Errors:

Rationale: Rationale not provided.

5. ERROR TABLE RATIONALE:

5.1 Table 1 Message Type Format Errors (Slave Mode Operation):

Error 1.1 HWA Received, bus priority unknown.

BIU Action:

* Indicate Recognize during Header
* If Datagram Message
  NAK on or before HZ+2
  Indicate Not Selected by HZ+2
* Else
  NAK on or before HA0
  Report Command Error in Header Acknowledge Word
  Indicate Not Selected after Header Acknowledge

Rationale: Indicates Slave is out of sync with protocol and cannot be trusted for proper operation.

Error 1.2 BIU "Busy".

BIU Action:

* Indicate Recognize during Header
* If Datagram Message
  NAK on or before HZ+2
  Indicate Not Selected by HZ+2
* Else
  NAK on or before HA0
  Report Busy in Acknowledge Word
  Indicate Not Selected after Header Acknowledge

5.1 (Continued):

Rationale: Slave will drop off since it cannot participate due to being busy.

Error 1.3 Broadcast and single Slave message type.

BIU Action:

* Indicate Recognize during Header
* If Datagram Message
  NAK on or before HZ+2
  Indicate Not Selected by HZ+2
* Else
  NAK on or before HA0
  Report Command Error in Acknowledge Word use Single Slave Acknowledge
  Indicate Not Selected after Header Acknowledge

Rationale: Protocol error detected.

Error 1.4 Format Bit Error.

BIU Action:

* NAK on H0+2
* Indicate Not Selected

Rationale: Slave cannot respond to message with improper format.

Error 1.5 AT/MT Not Implemented.

BIU Action:

For Datagram Message see Error Table 3.1 and 3A.1 else
* NAK on or Before HA0
* Report Resource Not Present in Acknowledge Word
* Indicate Recognize during Header
* Indicate Not Selected after Header Acknowledge

Rationale: Slave cannot respond to message with improper AT/MT.

Error 1.6 Abort during Header.

BIU Action:

* Respond with Acknowledge on cycle AB2
* Respond Not Selected on cycle AB3

5.1 (Continued):

Rationale: Allows Slave to gracefully drop off when Master aborts.

5.2 Table 2 Block Message Format and Device Errors (Slave Mode Operation):

Error 2.1 AT = 4,5,6,7 or AT=0,1,2,3 (if module does not support).

BIU Action:

* NAK on or before HA0
* Report Resource Not Present in Acknowledge Word
* Indicate Not Recognized during Header
* Indicate Not Selected after Header Acknowledge

Rationale: See rationale for error 1.5.

Error 2.2 Device Error On Data Source.

BIU Action:

* Assert NAK as soon as possible, but prior to Data Acknowledge
* Report Device Error in Acknowledge word
* Indicate Recognize during message
* Notify device of error

Rationale: A message is only successful if data is transferred between two or more modules. If a Slave cannot correctly store data in the device memory, the message is not successful. The assertion of a NAK notifies the Master that the message cannot be successfully completed. There is no exact timing relationship between the time data is on the Pi-Bus and when a memory store is attempted. Therefore there is no exact timing restrains on when the NAK must be asserted other than before the Data Acknowledge word. This restrain also allows the error to be reported as a Device Error to the Master in the acknowledge word. Indicating Recognized after the error allows the message to continue and allows for the reading of the Data Acknowledge word by the Master. Since this error is the result of either a device hardware failure or device software control failure, the error must be reported to the device.

Error 2.3 Device Error On Data Fetch.

BIU Action:

* Assert NAK after Header Acknowledge sequence
* Report Device Error in Acknowledge word
* Indicate Recognize during message
* Notify device of error

5.2 (Continued):

Rationale: If a Slave cannot correctly fetch data from the device memory, the message cannot be successful. Any time erroneous data is placed on the bus by a Slave, a NAK must be asserted to indicate an error in the message. The memory error may have been encountered before the Data cycles due to early data fetching during the header or header acknowledge cycles. However the NAK must be asserted after the header acknowledge and the error reported in the data acknowledge word. Indicating Recognized after the error allows the message to continue and allows for the reading of the Data Acknowledge word by the Master. Since this error is the result of either a device hardware failure or device software control failure, the error must be reported to the device.

Error 2.4 Suspend Sequence When Not Suspendable.

BIU Action:

* Assert NAK in two cycles
* Report Sequence Error in Acknowledge word
* Return indeterminate resume control words
* Notify device of error

Rationale: Even if a Slave cannot correctly suspend the message, the proper protocol (sequence) can be followed to allow the Master access the Data Acknowledge word. The NAK will indicate an error and cause the Master to disregard the resume control words. Since the resume control words will be disregarded, they can be indeterminate data but must be valid symbols. The Sequence Error in the Data Acknowledge will explain the reason of the NAK to the Master. Since this error occurs during the data transfer, the device is notified.

Error 2.5 Abort During Data.

BIU Action:

* Respond with Ack on cycle AB2
* Respond with Not Selected on cycle AB3
* Notify device of error

Rationale: A Master can initiate an Abort sequence at any time due to conditions it detects on the Pi-Bus or due to its own internal conditions. The Slave must always correctly follow the Abort sequence. Since this error occurs during the data transfer, the device is notified.

5.3 Table 2A Block Message Header Format and Device Errors (Slave Mode Operation for Label Block Messages):

Error 2A.1 Label Out Of Range.

BIU Action:

* Discontinue bus sequence
* Indicate Not Selected

Rationale: Allows Slave to gracefully exit sequence.

Error 2A.2 Device Error on Label Table Access.

BIU Action:

* NAK on or before HA0
* Report Device Error in Header Acknowledge Word
* Indicate Recognize during Header
* Indicate Not Selected after Header Acknowledge
* Notify Device of Error

Rationale: Allows Slave to gracefully exit sequence and to notify Slave device of Slave error.

Error 2A.3 Label Not Active.

BIU Action:

* Discontinue Bus Sequence
* Indicate Not Selected

Rationale: Allows Slave to gracefully exit sequence.

Error 2A.4 Resume and Label not suspended.

BIU Action:

* NAK on or before HA0
* Report Command Error in Acknowledge Word
* Indicate Recognize during Header
* Indicate Not Selected after Header Acknowledge

Rationale: Allows Slave to gracefully exit sequence.

Error 2A.5 HWB + HWC1 > buffer size.

BIU Action:

5.3 (Continued):

* NAK on or before HA0
* Report Resource Not Present in Acknowledge Word
* Indicate Recognize during Header
* Indicate Not Selected after Header Acknowledge

Rationale: Allows Slave to gracefully exit sequence.

Error 2A.6 Device or Label Busy is Active.

BIU Action:

* NAK on or before HA0
* Report Busy in Acknowledge Word
* Indicate Recognize during Header
* Indicate Not Selected after Header Acknowledge
* Stop memory accesses

Rationale: Allows Slave to gracefully exit sequence.

5.4 Table 3 Acknowledge Datagram Message (AT = 4-7) Format and Device Error (Slave Mode Operation):

Error 3.1 AT = 6,7 or AT = 4,5 (if the module does not support).

BIU Action:

* NAK on or before D0
* Indicate Recognized
* Inhibit memory accesses
* Report Resource Not Present in Data Acknowledge Word

Rationale: Allows Slave to gracefully exit sequence.

Error 3.2 Label Out of Range.

BIU Action:

* Discontinue Bus Sequence
* Indicate Not Selected

Rationale: Allows Slave to gracefully exit sequence.

Error 3.3 Device Error on Label Table Access.

BIU Action:

5.4 (Continued):

* NAK
* Indicate Recognized
* Inhibit memory accesses
* Report Device Error in Data Acknowledge Word
* Notify Device of Error

Rationale: Allows Slave to gracefully exit sequence and to notify Slave device of Slave error.

Error 3.4 Label Not Active.

BIU Action:

* Discontinue Bus Sequence
* Indicate Not Selected

Rationale: Allows Slave to gracefully exit sequence.

Error 3.5 Resume and Label not Suspended Access.

BIU Action:

* NAK
* Indicate Recognized
* Inhibit memory accesses
* Report Command Error in Data Acknowledge Word

Rationale: Allows Slave to gracefully exit sequence.

Error 3.6 HWB + HWC1 > Buffer Size.

BIU Action:

* Discontinue Bus Sequence
* Indicate Not Selected

Rationale: Allows Slave to gracefully exit sequence.

Error 3.7 Device or Label Busy.

BIU Action:

* NAK
* Indicate Recognized
* Inhibit memory accesses
* Report Busy in Data Acknowledge Word

5.4 (Continued):

Rationale: Allows Slave to gracefully exit sequence.

Error 3.8 Device Error on Data Store.

BIU Action:

* NAK before DA0
* Indicate Recognized during message
* Inhibit memory accesses
* Report Device Error in Data Acknowledge Word
* Notify Device of Error

Rationale: Allows Slave to gracefully exit sequence and to notify Slave device of Slave error.

Error 3.9 Suspend Sequence When Message Had an Error.

BIU Action:

* NAK in two cycles
* Report Sequence Error in Data Acknowledge Word
* Flag label as being "unsuspended"

Rationale: Allows Slave to gracefully exit sequence.

Error 3.10 Abort During Data.

BIU Action:

* Respond with ACK on cycle AB2
* Respond Not Selected on cycle AB3
* Notify Device of Error

Rationale: Allows Slave to gracefully exit sequence and to notify Slave device of incomplete data received.

5.5 Table 3A Non-Acknowledge Datagram Message (AT = 0-3) Format and Device Error (Slave Mode Operation):

Error 3A.1 AT = 2,3 or AT = 0,1 (if the module does not support).

BIU Action:

* NAK in two cycles
* Indicate Not Selected

5.5 (Continued):

Rationale: Allows Slave to gracefully exit sequence.

Error 3A.2 Label Out of Range.

BIU Action:

* Discontinue Bus Sequence
* Indicate Not Selected

Rationale: Allows Slave to gracefully exit sequence.

Error 3A.3 Device Error on Label Table Access.

BIU Action:

* NAK
* Indicate Not Selected
* Notify Device of Error

Rationale: Allows Slave to gracefully exit sequence and to notify Slave device of Slave error.

Error 3A.4 Label Not Active.

BIU Action:

* Discontinue Bus Sequence
* Indicate Not Selected

Rationale: Allows Slave to gracefully exit sequence.

Error 3A.5 Resume and Label not Suspended Access.

BIU Action:

* NAK
* Indicate Not Selected

Rationale: Allows Slave to gracefully exit sequence.

Error 3A.6 HWB + HWC1 > Buffer Size.

BIU Action:

* NAK
* Indicate Not Selected

5.5 (Continued):

Rationale: Allows Slave to gracefully exit sequence.

Error 3A.7 Device or Label Busy.

BIU Action:

* NAK
* Indicate Not Selected

Rationale: Allows Slave to gracefully exit sequence.

Error 3A.8 Device Error on Data Store.

BIU Action:

* NAK if before Dn+2
* Indicate Recognized during message
* Notify Device of Error

Rationale: Allows Slave to gracefully exit sequence and to notify Slave device of Slave error.

Error 3A.9 Suspend Sequence when message had an error which requires notification to the device.

BIU Action:

* NAK in two cycles
* Flag label as being "unsuspended"

Rationale: Allows Slave to gracefully exit sequence.

Error 3A.10 Abort During Data.

BIU Action:

* Respond with ACK on cycle AB2
* Respond Not Selected on cycle AB3
* Notify Device of Error

Rationale: Allows Slave to gracefully exit sequence and to notify Slave device of incomplete data.

5.6 Table 4 Parameter Write Message Format Errors (Slave Mode Operation):

Error 4.1 AT = 2 through 6 or AT = 0 (if the module does not support).

BIU Action:

* NAK on or before HA0
* Report Resource Not Present in Header Acknowledge word.
* Indicate Recognized during header.

Rationale: A JIAWG common module Slave is required to support an AT code of 1. Since the other codes are not required, they are not allowed. The NAK on the third header cycle notifies the bus Master of an error which will not permit the message to complete successfully. The Slave indicates Recognized during the header to permit the Master access to the Header Acknowledge word containing the Resource Not Present error. Since this error was a format problem by the Master and data was not transferred, the Slave device is not notified.

Error 4.2 AT Code = 7.

BIU Action:

* Assert NAK on or before HA0
* Report Command Error in Header Acknowledge word
* Indicate Recognized during header

Rationale: AS4710 reserves the use of an AT code of 7 for future needs. It requires Slaves to respond to it with a Command Error. The NAK on the third header cycle notifies the bus Master of an error which will not permit the message to complete successfully. The Slave indicates Recognized during the header to permit the Master access to the Header Acknowledge word containing the Command Error. Since this error was a format problem by the Master and data was not transferred, the Slave device is not notified.

5.7 Table 5 Bus Interface Message Format Errors (Slave Mode Operation):

Error 5.1A AT = 4 and Address Not 0 or Count Not 3.

BIU Action:

* Assert NAK on or before HA0
* Report Resource Not Present in Header Acknowledge word.
* Indicate Recognized during Header.
* Indicate Not Selected after Header Acknowledge.

5.7 (Continued):

Rationale: A Bus Interface message with an AT Code of 4 can only access the System Timer. The System Timer is the only register in the address space specified by an AT Code of 4. The register address of the timer is zero. The timer contains three words. The message must either read or write all three words. A partial access of the System Timer contents is not permitted. Hence a message accessing the System which does not contain a register address of zero or a word count of three is rejected by the Slave with a NAK on or before the header acknowledge word. The Slave indicates Recognized during the header so the Master can obtain the header acknowledge word containing the Resource Not Present error Message Complete indication. The Slave drops off of the message sequence following the header acknowledge by indicating Not Selected. Since this error was a format problem by the Master and data was not transferred, the Slave device is not notified.

Error 5.1B AT = 5 or 6, or AT = 4 (if the module does not support).

BIU Action:

* Assert NAK on or before HA0
* Report Resource Not Present Header Acknowledge word
* Indicate Recognized during Header
* Indicate Not Selected after Header Acknowledge

Rationale: Data Link Layer address spaces which would be accessed by an AT Code of 5 or 6 are not defined for JIAWG applications. Hence a message accessing these address spaces is rejected by the Slave with a NAK on or before the header acknowledge word. The Slave indicates Recognized during the header so the Master can obtain the header acknowledge word containing the Resource Not Present error and Message Complete indication. The Slave drops off of the message sequence following the header acknowledge by indicating Not Selected. Since this error was a format problem by the Master and data was not transferred, the Slave device is not notified.

Error 5.2 AT = 1, 2, 3 or 7.

BIU Action:

* NAK on or before HA0
* Report Command Error in Header Acknowledge word
* Indicate Recognized during Header
* Indicate Not Selected after Header Acknowledge

Rationale: Bus Interface Message AT Codes of 1, 2, 3 or 7 are reserved by AS4710. A message attempted to access these AT codes is rejected by the Slave with a NAK on the third cycle of the message. The Slave indicates Recognized during the header so the Master can obtain the header acknowledge word containing the Command Error and Message Complete indication. The Slave drops off of the message sequence following the header acknowledge by indicating Not Selected. Since this error was a format problem by the Master and data was not transferred, the Slave device is not notified.

5.7 (Continued):

Error 5.3 Addressed Register Write Protected.

BIU Action:

* NAK on or before DA0
* Report Protect Error in Data Acknowledge Word.
* Continue Bus Sequence.

Rationale: A NAK is asserted two cycles after data addressed to a write protected register is on the bus. Non-transfer cycles may cause the NAK to be asserted early or for more than one cycle. This indicates to the Master that the message cannot be successfully completed. The contents of the addressed write protected register is unchanged. This error does not effect the remaining portion of the message. Any writes to non-protected registers are carried out. The Slave continues to indicates Recognized during the message so the Master can finish the message and obtain the header acknowledge word containing the Protect Error.

Error 5.4 Reserved Addressed Register.

BIU Action:

* NAK on or before DA0
* Report Resource Not Present in Data Acknowledge word.
* Continue the bus sequence.

Rationale: A Slave asserts a NAK in response to a message attempting to access a non-implemented register. During a write message, the NAK is asserted two cycles after write data addressed to a non-implemented register is on the bus. During a read message, the NAK is asserted on the same cycle that the data from the non-implemented register should be on the bus. Non-transfer cycles may cause the NAK to be asserted early or for more than one cycle. The data on the bus from a non-implemented register is indeterminate but consists of valid symbols. The NAK indicates to the Master that the message cannot be successfully completed. This error does not effect the remaining portion of the message. Any further transfers with implemented registers are carried out. The Slave continues to indicates Recognized during the message so the Master can finish the message and obtain the header acknowledge word containing the Resource Not Present error.

Error 5.5 Abort During Data.

BIU Action:

* Respond with Ack during AB2.
* Respond with Not Selected on cycle AB3.

Rationale: A Master can initiate an Abort sequence at any time due to conditions it detects on the Pi-Bus or due to its own internal conditions. The Slave must always correctly follow the Abort sequence.

5.8 Table 6 Reserved Message Type Format Errors (MT = 0,2,10) (Slave Mode Operation):

Error 6A.1 All AT Codes.

BIU Action:

* Assert NAK on H0+2
* Indicate Not Selected

Rationale: Allows Slave to gracefully exit sequence.

5.9 Table 7 Vie Sequence Pi-Bus Errors (All Modules):

Error 7.1 Data Line Error.

BIU Action:

* If an uncorrectable line error
  Assert NAK in two cycles
  Set bus priority unknown
  If contender, notify device of error
  If eventual vie winner, release bus
* Else Ignore

Rationale: Non-Master participants of a bus sequence always assert a NAK in two cycles in response to a Data Line error. Since the Data Lines are used to establish the Bus Priority and there was an error with those lines, the Bus Priority is set to unknown. If this module is the eventual winner of the vie sequence, it must release the bus following the vie or after an abort following the first H0 and report the error to allowed the failing module to be identified and isolated.

Error 7.2 Uncorrectable Cycle Type Line Error.

BIU Action:

* Assert NAK in two cycles
  Set Bus Priority unknown
  If eventual vie winner, notify device of error and
  release bus

Rationale: Non-Master participants of a bus sequence always assert a NAK in two cycles in response to a Cycle Type Line error. Since the Cycle Type Lines are used to control the Vie sequence and there was an error with those lines, the Bus Priority is set to unknown. If this module is the eventual winner of the vie sequence, it must release the bus following the vie or after an abort following the first H0 and report the error to allowed the failing module to be identified and isolated.

5.9 (Continued):

Error 7.3 Uncorrectable Acknowledge Set Line Error.

BIU Action:

* Set Bus Priority unknown
  If eventual vie winner, notify device of error and release bus

Rationale: Since the Acknowledge Set lines are not directly used to control the Vie sequence, a NAK is not asserted. However these lines are used by the modules participating in the Vie sequence. Therefore an error in these signals causes the modules to set the Bus Priority unknown. If this module is the eventual winner of the vie sequence, it must release the bus following the vie or after an abort following the first H0.

Error 7.4 Uncorrectable Wait Line Error.

BIU Action:

* If eventual vie winner, notify device of error and release bus
* Otherwise ignore.

Rationale: Since the Wait Lines are not used to control the Vie sequence or to set the Bus Priority, an error with them during the Vie sequence does not cause a NAK. If this module is the eventual winner of the vie sequence, it must release the bus following the vie or after an abort following the first H0.

Error 7.5 Uncorrectable Bus Request Line Error.

BIU Action:

* If eventual vie winner, notify device of error and release bus
* Otherwise ignore.

Rationale: Since the Bus Request Lines are not used to control the Vie sequence or to set the Bus Priority, an error with them during the Vie sequence does not cause a NAK. If this module is the eventual winner of the vie sequence, it must release the bus following the vie or after an abort following the first H0.

Error 7.6 Cycle Type Sequence Error.

BIU Action:

* Assert NAK in two cycles.
* Set Bus Priority unknown
* If eventual vie winner, notify device of error and release bus

5.9 (Continued):

Rationale: Non-Master participants of a bus sequence always assert a NAK in two cycles in response to a Cycle Type sequence error. Since the Cycle Type Lines are used to control the Vie sequence and there was an error with those lines, the Bus Priority is set to unknown. If this module is the eventual winner of the vie sequence, it must release the bus following the vie or after an abort following the first H0.

Error 7.7 Acknowledge Set Sequence Error.

BIU Action:

* Set Bus Priority unknown
* If eventual vie winner, notify device of error and release bus

Rationale: Since the Acknowledge Set lines are not directly used to control the Vie sequence, a NAK is not asserted. However these lines are used by the modules participating in the Vie sequence. Therefore an error in these signals causes the modules to set the Bus Priority unknown. If this module is the eventual winner of the vie sequence, it must release the bus following the vie or after an abort following the first H0.

Error 7.8 Wait Line Sequence.

BIU Action:

* If eventual vie winner, notify device of error and release bus
* Otherwise Ignore

Rationale: Since the Wait Lines are not used to control the Vie sequence or to set the Bus Priority, an error with them during the Vie sequence does not cause a NAK. If this module is the eventual winner of the vie sequence, it must release the bus following the vie or after an abort following the first H0.

Error 7.9 Bus Request Sequence Error.

BIU Action:

* If eventual vie winner, notify device of error and release bus
* Otherwise Ignore

Rationale: Since the Bus Request Lines are not used to control the Vie sequence or to set the Bus Priority, an error with them during the Vie sequence does not cause a NAK. If this module is the eventual winner of the vie sequence, it must release the bus following the vie or after an abort following the first H0.

5.9 (Continued):

Error 7.10 NAK Detection.

BIU Action:

* Ignore.

Rationale: Since only a Master uses the reception of a NAK symbol to control bus operation and there is no Master during a Vie sequence, the NAK is ignored. However, the NAK will be useful for bus monitors.

Error 7.11 Correctable CT Wait, AS, or BR Line Error.

BIU Action:

* Ignore.

Rationale: Ignore since error is corrected.

5.10 Table 8 H0 Pi-Bus Errors (Slave Mode Operation):

Error 8.1 Uncorrectable Data Line Error.

BIU Action:

* Remain a non-participant

Rationale: A module must receive HWA correctly before becoming a Slave. A HWA cannot be received correctly in the presence of a Data Line error. Since the module did not become a Slave, the error is not reported to the device.

Error 8.2 Uncorrectable Cycle Type Line Error.

BIU Action:

* Remain a non-participant

Rationale: A module must receive HWA correctly before becoming a Slave. A HWA cannot be received correctly in the presence of a Cycle Type Line error. Since the module did not become a Slave, the error is not reported to the device.

Error 8.3 Uncorrectable Acknowledge Set Line Error.

BIU Action:

5.10 (Continued):

* Continue bus sequence
* Report Uncorrectable Line Error in Header Acknowledge Word

Rationale: A module can receive HWA correctly and become a Slave in the presence of an Acknowledge Set line error. However, the error must be reported in the Header Acknowledge word. Since the message can be completed correctly, a NAK is not asserted and the device is not notified of the error.

Error 8.4 Uncorrectable Wait Line Error.

BIU Action:

* Continue bus sequence
* Report Uncorrectable Line Error in Header Acknowledge Word

Rationale: Since a Wait is not accepted on an H0 cycle, a module can receive HWA correctly and become a Slave in the presence of a Wait line error. However, the error must be reported in the Header Acknowledge word. Since the message can be completed correctly, a NAK is not asserted and the device is not notified of the error.

Error 8.5 Uncorrectable Bus Request Line Error.

BIU Action:

* Continue bus sequence
* Report Uncorrectable Line Error in Header Acknowledge Word

Rationale: A module can receive HWA correctly and become a Slave in the presence of an Bus Request line error. However, the error must be reported in the Header Acknowledge word. Since the message can be completed correctly, a NAK is not asserted and the device is not notified of the error.

Error 8.6 Cycle Type Sequence Error.

BIU Action:

* Remain a non-participant

Rationale: A module must receive HWA correctly before becoming a Slave. A HWA cannot be received correctly in the presence of a Cycle Type sequence error. Since the module did not become a Slave, the error is not reported to the device.

Error 8.7 Acknowledge Set Sequence Error.

BIU Action:

5.10 (Continued):

* Continue bus sequence
* Report Sequence Error in Header Acknowledge Word

Rationale: A module can receive HWA correctly and become a Slave in the presence of an Acknowledge Set sequence error. However, the error must be reported in the Header Acknowledge word. Since the message can be completed correctly, a NAK is not asserted and the device is not notified of the error.

Error 8.8 Wait Line Sequence.

BIU Action:

* Continue bus sequence
* Report Sequence Error in Header Acknowledge Word

Rationale: Since a Wait is not accepted on an H0 cycle, a module can receive HWA correctly and become a Slave in the presence of a Wait sequence error. However, the error must be reported in the Header Acknowledge word. Since the message can be completed correctly, a NAK is not asserted and the device is not notified of the error.

Error 8.9 NAK Detection.

BIU Action:

* Ignore

Rationale: Only a Master uses the reception of a NAK symbol to control bus operation. The Slaves ignore the NAK.

Error 8.10 Correctable Line Error.

BIU Action:

* Continue sequence
* Report Correctable Line Error in Header Acknowledge Word

Rationale: Continue since error corrected and report to support diagnostics.

5.11 Table 9 H1 through HAn Pi-Bus Errors (Slave Mode Operation):

Error 9.1A Uncorrectable Data Line Error, H1 through Hn.

BIU Action:

5.11 (Continued):

* If Datagram Message
  Assert NAK in two cycles
* Else
  Assert NAK in two cycles
  Report Uncorrectable Line Error in Header Acknowledge Word
* Indicate Not Selected after Header Acknowledge.

Rationale: The NAK is asserted to notify the Master of an error which prevents the message from completing correctly. The error is reported in the Header Acknowledge word. Since the Slave could not correctly receive the header, it drops off the bus (indicates Not Selected) after the header acknowledge. Since the message has not transferred data, the device is not notified.

Error 9.1B Uncorrectable Data Line Error, HZ through HAn.

BIU Action:

* Assert NAK in two cycles
* If Non-Acknowledge Datagram
  Ignore
* Else
  Report Uncorrectable Line Error in Acknowledge Word

Rationale: Allows Slave to gracefully exit sequence.

Error 9.2 Uncorrectable Cycle Type Line Error.

BIU Action:

* Assert NAK in two cycles.
* If Datagram
  Indicate Not Selected after HZ
* Else
  Report Uncorrectable Line Error in Acknowledge Word
* If prior to HZ: become not selected after Header Acknowledge

Rationale: The NAK is asserted to notify the Master of an error which prevents the message from completing correctly. The error is reported in the Header Acknowledge word. Since the Slave could not correctly receive the header, it drops off the bus (indicates Not Selected) after the header acknowledge. Since the message has not transferred data, the device is not notified.

Error 9.3 Uncorrectable Acknowledge Set Line Error.

BIU Action:

5.11 (Continued):

* If Non-Acknowledge Datagram
  Ignore
* Else
  Report Uncorrectable Line Error in Acknowledge Word.

Rationale: Since an Acknowledge Set line error does not prevent the Slave from successfully completing the message, a NAK is not asserted and the device is not notified. The error is reported in the Header Acknowledge word.

Error 9.4 Uncorrectable Wait Line Error.

BIU Action:

* Assert NAK in two cycles
* If Non-Acknowledge Datagram
  Ignore
* Else
  Report Uncorrectable Line Error in Acknowledge Word.

Rationale: The NAK is asserted to notify the Master of an error which prevents the message from completing correctly. The error is reported in the Header Acknowledge word. Since the Slave could not correctly receive the header, it drops off the bus (indicates Not Selected) after the header acknowledge. Since the message has not transferred data, the device is not notified.

Error 9.5 Uncorrectable Bus Request Line Error.

BIU Action:

* If Non-Acknowledge Datagram
  Ignore
* Else
  Report Uncorrectable Line Error in Acknowledge Word.

Rationale: Since a Bus Request line error does not prevent the Slave from successfully completing the message, a NAK is not asserted and the device is not notified. The error is reported in the Header Acknowledge word.

Error 9.6 Cycle Type Sequence Error.

BIU Action:

* Assert NAK in two cycles
* Discontinue bus sequence

5.11 (Continued):

Rationale: The NAK is asserted to notify the Master of an error which prevents the message from completing correctly. The Slave cannot continue with a Cycle Type sequence error. It discontinues the bus sequence by indicating Not Selected. Since the message has not transferred data, the device is not notified.

Error 9.7 Acknowledge Set Sequence Error.

BIU Action:

* If Non-Acknowledge Datagram
  Ignore
* Else
  Report Sequence Error in Acknowledge Word

Rationale: Since an Acknowledge Set sequence error does not prevent the Slave from successfully completing the message, a NAK is not asserted and the device is not notified. The error is reported in the Header Acknowledge word.

Error 9.8 Wait Sequence Error.

BIU Action:

* Assert NAK in two cycles
* If Non-Acknowledged Datagram
  Ignore
* Else
  Report Sequence Error in Acknowledge Word.

Rationale: Receiving an odd number of Wait cycles is a Wait sequence Error. The NAK is asserted to notify the Master of an error which prevents the message from completing correctly. The error is reported in the Header Acknowledge word. Since the Slave could not correctly receive the header, it drops off the bus (indicates Not Selected) after the header acknowledge. Since the message has not transferred data, the device is not notified.

Error 9.9 NAK Detection.

BIU Action:

* Ignore.

Rationale Only a Master uses the reception of a NAK symbol to control bus operation. The Slaves ignore the NAK.

Error 9.10 Correctable Line Error.

5.11 (Continued):

BIU Action:

* Continue Sequence
* If Non-Acknowledge Datagram Message
  Ignore
* Else
  Report Correctable Line Error in Acknowledge Word

Rationale: Continue since corrected and report in support of diagnostics.

5.12 Table 10 D0 through DAn Pi-Bus Errors (Slave Mode Operation):

Error 10.1 Uncorrectable Data Line Error.

BIU Action:

* Assert NAK in two cycles
* If not Non-Acknowledge Datagram
  Report Uncorrectable Line Error in Acknowledge Word
* Continue Data transfer if before DZ
* Notify device of error

Rationale: The NAK is asserted to notify the Master of an error which prevents the message from completing correctly. A Data line error does not prevent the Slave from continuing with the message. The error is reported in the Data Acknowledge word. Since the error effects the data transfer, the error is reported to the device.

Error 10.2 Uncorrectable Cycle Type Line Error.

BIU Action:

* Assert NAK in two cycles
* If not Non-Acknowledge Datagram
  Report Uncorrectable Line Error in Acknowledge Word
* Continue Data transfer if before DZ
* Notify device of error

Rationale: The NAK is asserted to notify the Master of an error which prevents the message from completing correctly. A Cycle type line error does not prevent the Slave from continuing with the message. The error is reported in the Data Acknowledge word. Since the error effects the data transfer, the error is reported to the device.

Error 10.3 Uncorrectable Acknowledge Set Line Error.

5.12 (Continued):

BIU Action:

* If not Non-Acknowledge Datagram
  Report Uncorrectable Line Error in Acknowledge Word.
* Continue Data transfer if before DZ

Rationale: Since an Acknowledge Set line error does not prevent the Slave from successfully completing the message, a NAK is not asserted and the device is not notified. The error is reported in the Data Acknowledge word.

Error 10.4 Uncorrectable Wait Line Error.

BIU Action:

* Assert NAK in two cycles
* If not Non-Acknowledge Datagram
  Report Uncorrectable Line Error in Acknowledge Word
* Continue Data transfer if before DZ
* Notify device of error

Rationale: The NAK is asserted to notify the Master of an error which prevents the message from completing correctly. The error is reported in the Data Acknowledge word. The Slave continues participating in the bus sequence. Since the error effects the data transfer, the error is reported to the device.

Error 10.5 Uncorrectable Bus Request Line Error.

BIU Action:

* If not Non-Acknowledge Datagram
  Report Uncorrectable Line Error in Acknowledge Word
* Continue data transfer

Rationale: Since a Bus Request line error does not prevent the Slave from successfully completing the message, a NAK is not asserted and the device is not notified. The error is reported in the Data Acknowledge word.

Error 10.6 Cycle Type Sequence Error.

BIU Action:

* Assert NAK in two cycles
* Indicate Not Selected after NAK
* Discontinue data transfer
* Notify device of error

5.12 (Continued):

Rationale: The NAK is asserted to notify the Master of an error which prevents the message from completing correctly. The Slave cannot continue with a Cycle Type sequence error. It discontinues the bus sequence by indicating Not Selected. Since the error effects the data transfer, the error is reported to the device.

Error 10.7 Acknowledge Set Sequence Error.

BIU Action:

* Report Sequence Error in Acknowledge Word

Rationale: Since an Acknowledge Set sequence error does not prevent the Slave from successfully completing the message, a NAK is not asserted and the device is not notified. The error is reported in the Data Acknowledge word.

Error 10.8 Wait Sequence Error.

BIU Action:

* Assert NAK in two cycles
* If not Non-Acknowledge Datagram
  Report Sequence Error in Acknowledge Word
* Continue data transfer in before DZ
* Notify device of error

Rationale: Receiving an odd number of Wait cycles is a Wait sequence Error. The NAK is asserted to notify the Master of an error which prevents the message from completing correctly. The error is reported in the Data Acknowledge word. The Slave continues to participate in the bus sequence. Since the error effects the data transfer, the error is reported to the device.

Error 10.9 NAK Detection.

BIU Action:

* Ignore

Rationale: Only a Master uses the reception of a NAK symbol to control bus operation. The Slaves ignore the NAK.

Error 10.10 Correctable Line Error.

BIU Action:

5.12 (Continued):

* Continue Sequence
* If Non-Acknowledge Datagram Message
  Ignore
* Else
  Report Correctable Line Error in Acknowledge Word

Rationale: Continue since error corrected and report error in support of diagnostics.

5.13 Error Table 11 H0 through HZ Errors (Master Mode):

Error 11.1A Uncorrectable Data or Acknowledge Line Error.

BIU Action:

* Terminate message with an abort by HAn+5
* Notify device of error

Rationale: Master aborts potentially bad transfer and notifies Master device in support of diagnostics.

Error 11.1B Uncorrectable Wait or Bus Request Line Error.

BIU Action:

* Terminate message with an abort by HAn+5
* Notify device of error

Rationale: Master aborts potentially bad transfer and notifies Master device in support of diagnostics.

Error 11.1C Uncorrectable Cycle Type Line Error.

BIU Action:

* Terminate message with an abort by HAn+5
* Notify device of error

Rationale: Master aborts potentially bad transfer and notifies Master device in support of diagnostics.

Error 11.2 Cycle Type Sequence Error.

BIU Action:

5.13 (Continued):

* Terminate message with an abort in two cycles
* Notify device of error

Rationale: Master aborts potentially bad transfer and notifies Master device in support of diagnostics.

Error 11.3A Acknowledge Set Sequence Error.

BIU Action:

* Terminate message with an abort by HAn+5
* Notify device of error

Rationale: Master aborts potentially bad transfer and notifies Master device in support of diagnostics.

Error 11.3B Wait Sequence Error.

BIU Action:

* Terminate message with an abort by HAn+5
* Notify device of error

Rationale: Master aborts potentially bad transfer and notifies Master device in support of diagnostics.

Error 11.4A NAK detected Single Slave.

BIU Action:

* Terminate message with an abort by HAn+5
* Notify device of error

Rationale: Master aborts since there are no other listeners.

Error 11.4B NAK detected Multicast or Broadcast.

BIU Action:

5.13 (Continued):

* If not Abort_On_Error_Mode:
  Notify Device of error after message sequence
* Else
  If a Non-Acknowledge Datagram
  Ignore
  Else
  Terminate message with an abort by HAn+5
  Notify device of error

Rationale: Not applicable for non-acknowledge datagram since NAK cannot be generated.. Otherwise always notify Master device in support of diagnostics and abort only if Abort_On_Error_Mode is set.

Error 11.5 Vie Interval B Exceeded.

BIU Action:

* Terminate message with an abort in two cycles
* Notify device of error
* Enter Idle State

Rationale: Defined Vie Interval B operation.

Error 11.6 Absolute Tenure Time-out.

BIU Action:

* Terminate message with an abort so that the fourth cycle of the abort shall occur on or before the 2**24+8 cycle of the Master's tenure
* Notify device of error
* Enter Idle State

Rationale: Defined Absolute Tenure Time-out operation.

Error 11.7 Correctable Line Error.

BIU Action:

* Continue Sequence

Rationale: Continue since error is corrected.

Error Table 12HA0 through HAn Errors or HA0 through HAn+2 for Parameter Writes (Master Mode).

5.14 Error 12.1A Uncorrectable Data or Acknowledge Line Error:

BIU Action:

* Terminate message with an abort by HAn+5
* Notify device of error

Rationale: Master aborts potentially bad transfer and notifies Master device in support of diagnostics.

Error 12.1B Uncorrectable Wait or Bus Request Line Error.

BIU Action:

* Terminate message with an abort by HAn+5
* Notify device of error

Rationale: Master aborts potentially bad transfer and notifies Master device in support of diagnostics.

Error 12.1C Uncorrectable Cycle Type Line Error.

BIU Action:

* Terminate message with an abort by HAn+5
* Notify device of error

Rationale: Master aborts potentially bad transfer and notifies Master device in support of diagnostics.

Error 12.2 Cycle Type Sequence Error.

BIU Action:

* Terminate message with an abort in two cycles
* Notify device of error

Rationale: Master aborts potentially bad transfer and notifies Master device in support of diagnostics.

Error 12.3A Acknowledge Set Sequence Error.

BIU Action:

* Terminate message with an abort by HAn+5
* Notify device of error

5.14 (Continued):

Rationale: Master aborts potentially bad transfer and notifies Master device in support of diagnostics.

Error 12.3B Wait Sequence Error.

BIU Action:

* Terminate message with an abort by HAn+5
* Notify device of error

Rationale: Master aborts potentially bad transfer and notifies Master device in support of diagnostics.

Error 12.4A NAK Detected Single Slave.

BIU Action:

* Terminate message with an abort by HAn+5
* Notify device of error

Rationale: Master aborts since there are no other listeners.

Error 12.4B NAK Detected Multicast or Broadcast.

BIU Action:

* If not Abort_On_Error_Mode:
  Notify Device of error after message sequence
* If Abort_On_Error_Mode:
  Terminate message with an abort by HAn+5
  Notify device of error

Rationale: Not applicable for non-acknowledge datagram since NAK cannot be generated.. Otherwise always notify Master device in support of diagnostics and abort only if Abort_On_Error_Mode is set.

Error 12.5A Header Acknowledge Semantic Error, Single Slave.

BIU Action:

* Terminate message with an abort by HAn+5
* Notify device of error

Rationale: Master aborts since there are no other listeners.

5.14 (Continued):

Error 12.5B Header Acknowledge Semantic Error, Multicast or Broadcast.

BIU Action:

* If not Abort_On_Error_Mode:
  Notify Device of error after message sequence
* If Abort_On_Error_Mode:
  Terminate message with an abort by HAn+5
  Notify device of error

Rationale: Notify Master device of error in support of diagnostics and if Abort_On_Error_Mode is set, abort transfer.

Error 12.6A Header Acknowledge Error or Busy, Single Slave.

BIU Action:

* Terminate message with an abort by HAn+5
* Notify device of error

Rationale: Master aborts since there are no other listeners.

Error 12.6B Header Acknowledge Error or Busy, Multicast or Broadcast.

BIU Action:

* If not Abort_On_Error_Mode:
  Notify Device of error after message sequence
* If Abort_On_Error_Mode:
  Terminate message with an abort by HAn+5
  Notify device of error

Rationale: Notify Master device of error in support of diagnostics and if Abort_On_Error_Mode is set, abort transfer.

Error 12.7 Vie Interval B Exceeded.

BIU Action:

* Terminate message with an abort in two cycles
* Notify device of error
* Enter Idle State

Rationale: Defined Vie Interval B operation.

5.14 (Continued):

Error 12.8 Absolute Tenure Time-out.

BIU Action:

* Terminate message with an abort so that the fourth cycle of the abort shall occur on or before the 2**24+8 cycle of the Master's tenure
* Notify device of error
* Enter Idle State

Rationale: Defined Absolute Tenure Time-out operation.

Error 12.9 Correctable Line Error.

BIU Action:

* Continue Sequence

Rationale: Continued since error corrected.

5.15 Error Table 13 Do through Dz Errors (Master Mode):

Error 13.1A Uncorrectable Data, Wait, Acknowledge Set or Bus Request, Line Error.

BIU Action:

* Terminate message with an abort by DAn=5
* Notify device of error

Rationale: Master aborts potentially bad transfer and notifies Master device in support of diagnostics.

Error 13.1B Uncorrectable Cycle Type Line Error.

BIU Action:

* Terminate message with an abort by DAn=5
* Notify device of error

Rationale: Master aborts potentially bad transfer and notifies Master device in support of diagnostics.

Error 13.2 Cycle Type Sequence Error.

BIU Action:

5.15 (Continued):

* Terminate message with an abort by DAn=5
* Notify device of error

Rationale: Master aborts potentially bad transfer and notifies Master device in support of diagnostics.

Error 13.3 Acknowledge Set Sequence Error (NS or ACK).

BIU Action:

* Terminate message with an abort by DAn=5
* Notify device of error

Rationale: Master aborts potentially bad transfer and notifies Master device in support of diagnostics.

Error 13.4 Wait Sequence Error.

BIU Action:

* Terminate message with an abort by DAn+5
* Notify device of error

Rationale: Master aborts potentially bad transfer and notifies Master device in support of diagnostics.

Error 13.5 NAK Detected During Data.

BIU Action:

* If not a Non-Acknowledged Datagram and
* If Abort_On_Error_Mode or Single Cast:
  Terminate message with an abort by DAn+5
  Notify device of error
* If not Abort_On_Error_Mode:
  Notify device of error after message sequence

Rationale: Not applicable for non-acknowledge datagram since NAK cannot be generated.. Otherwise always notify Master device in support of diagnostics and abort only if Abort_On_Error_Mode or if a single cast (since there would be no remaining listener) is set.

Error 13.6 Vie Interval B Exceeded.

BIU Action: