REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)* May 1990	**2. REPORT TYPE** Conference paper	**3. DATES COVERED** *(From - To)*

4. TITLE AND SUBTITLE See report.	**5a. CONTRACT NUMBER**
	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S) See report.	**5d. PROJECT NUMBER**
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) See report.	**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) See report.	**10. SPONSOR/MONITOR'S ACRONYM(S)**
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT

Distribution Statement A - Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES

Presented at the IEEE 1990 National Aerospace and Electronics Conference (NAECON 1990) held in Dayton, Ohio, on 21-25 May 1990.

14. ABSTRACT

See report.

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON**
a. REPORT Unclassified	**b. ABSTRACT** Unclassified	**c. THIS PAGE** Unclassified	UU		**19b. TELEPHONE NUMBER** *(include area code)*

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std. Z39.18

The JIAWG Input/Output System (JIOS)

John Newport, Ph.D.
Naval Avionic Center
Indianapolis, Indiana 46219-21B

Chuck Roark, Ph.D.
Texas Instruments Defense Systems & Electronics Group
Plano, Texas 75086

ABSTRACT: The JIAWG Input/Output System (JIOS) provides the software/hardware interface for Built-In-Test and I/O provided via the PI-Bus and TM-Bus for the JIAWG 16-bit Common Modules. This paper describes the need for JIOS, the functionality of the JIOS, concerns related to the use of the JIOS, and a planned JIOS demonstration.

1.0 Overview

The Joint Integrated Avionics Working Group (JIAWG) is a Congressionally mandated tri-service organization founded to establish design standards and specifications for the three service's next generation aircraft. A subset of these documents is concerned with the development of interchangeable hardware modules. Two hardware modules are considered interchangeable if

* the two modules will work electrically to specifications within the same slot
* operational software will execute on both modules without modification.

These common modules are to be usable by all three services and are to be built under the philosophy of form-fit-function-and-interface (as compared to built-to-print). The JIAWG produced documents will be referenced contractually in the next generation aircraft procurements.

Two JIAWG documents are responsible for defining the standard 16-bit processing module. Specifically, these documents are the 16-Bit Common Avionics Processor (CAP-16) specification and the Common Avionics Processor 16-Bit Instruction Set Architecture (ISA) specification. Together, these documents define the standard 16-bit processing module to be a MIL-STD-1750A, Notice 1 Instruction Set Architecture (ISA), together with the module hardware and software interfaces and functions. Two of the most difficult issues to solve regarding the ISA have been the Input/Output (I/O) implementation and a standard approach to module level Built-In-Test (BIT).

The type of I/O required for CAP-16 is associated with the backplane bus hardware. The primary intermodule communication channel is a dual redundant, single controller, Very High Speed Integrated Circuit (VHSIC), Phase II, PI-Bus, as specified in the JIAWG PI-Bus specification. A dual VHSIC Test and Maintenance (TM) Bus interface, as defined by the JIAWG TM-Bus

Table 1. Differences in CAP-16 Vendor Designs

ITEM	DIFFERENCES
Software interface to PI-Bus	- Different XIO assignments - Difference in PI-Bus data structures - Difference in interrupt structure - Even though there is much common functionality, there are some differences in functionality
Software interface to TM-Bus	- Different XIO assignments - Difference in TM-Bus data structures - Major differences in overall commands and functionality - difference in interrupt structure
Software interface to Built-In-Test	- Different BIT reporting methods, including amount of and format of information reported - Difference in BIT functionality supported and in manner BIT functionality initiated

specification, is also needed to provide intermodule support for fault tolerance, testability, and diagnostics. Finally, a Built-In-Test interface is required to provide software access to the module diagnostic hardware.

Four CAP-16 vendors support next generation aircraft Demonstration/Validation and Full Scale Development programs of the three services. However, each of the ongoing programs began prior to establishment of the JIAWG. As a consequence, the great flexibility that MIL-STD-1750 allows regarding I/O allowed the four JIAWG CAP-16 vendors' designs to be quite different. A summary of these differences is presented in Table I. This is a major commonality problem that required an unconventional solution, the JIAWG Input/Output System (JIOS). In order to understand the problem more fully, though, an examination of the I/O requirements of MIL-STD-1750A is needed.

2.0 MIL-STD-1750 INPUT/OUTPUT

Within the MIL-STD-1750A Instruction Set Architecture all I/O is required to occur through eXecute Input/Output (XIO) instructions. There are three types of XIO instructions: mandatory, optional, and user defined. The user defined XIO instructions are used to allow MIL-STD-1750A processors to interface to I/O hardware which is not defined in MIL-STD-1750A. These user defined XIO instructions are associated with "spare" channel codes. The standard reserves 21,140 read channel codes and an equal number of write codes for user defined XIOs.

Once a hardware interface controller, such as a PI-Bus Interface Unit (PBIU) is defined, XIOs can be defined to permit communication between the processor and the controller. As a consequence, the XIOs reflect the underlying bus interface logic, especially with respect to register and memory control. Thus, different I/O hardware implementations may have not only different XIO mnemonics and channel code assignments, but different XIO functions.

Three of the four vendors associated with the CAP-16 directly control input/output with user defined XIO commands. That is, XIO commands are used to communicate with the I/O controller register structure, point to transfer blocks in memory, and update channel control data structures in memory. Unfortunately, a thorough investigation of these XIO commands showed that, while there is some commonality in overall PBIU functionality, there is almost no commonality among XIO implementations in the designs. This is a consequence of the hardware differences among the I/O controllers. Therefore, it became clear that either one particular design must be selected or a new XIO design developed, in order to have common CAP-16 XIOs. The cost and schedule impact of the redesigns was so large that the JIAWG began to investigate alternatives.

The fourth JIAWG vendor uses memory mapped I/O. This involves providing hardware within the memory management unit which controls reads and writes to dedicated I/O memory locations. Unfortunately, there are no predefined MIL-STD-1750A ISA structures which lend themselves easily to such I/O. As a consequence, this implementation has no XIO commonality with the designs of the other CAP-16 vendors and, as such, selection of this design suffers the same drawbacks as selecting one of the three other XIO designs.

The CAP-16 dilemma, then is to define some common ISA mechanism which will not render all existing designs obsolete, will support the form-fit-function-and-interface common module acquisition approach, yet can be used as as a baseline to grow to a pure hardware, assembly language interface. The JIOS concept was formulated to answer this need.

3.0 JIOS STRUCTURE

The JIOS is intended to be a generic software interface for BIT as well as PI and TM buses. The hardware and software which comprise the JIOS are required to be an integral part of the module architecture so that no operational software downloads are needed.

The JIOS features are specified by the JIAWG Input/Output Built-In-Test Interface Definition Specification (IOBIDS). IOBIDS contains a complete definition of the user interface including functions provided, calling sequences, parameter type definitions, and special user requirements. These definitions are provided as an Ada package specification with no package body, for each functional area. The package body is not provided since this would be vendor specific and, as such, following Ada package body visibility rules, is not visible to operational software or the Runtime System (RTS). Linkage conventions are also established so that operational software and Runtime Systems can be linked to the JIOS facilities.

Figure 1 depicts the relationship between the JIOS, CAP-16 module hardware, and operational and RTS software. The JIOS provides a software layer which interfaces with operational software or RTS. This layer then implements the desired functionality using a combination of software and calls to module hardware.

Four I/O designs exist upon which to base the JIOS designs. Hence, the development of the IOBIDS has been, in effect, a top-down functional design

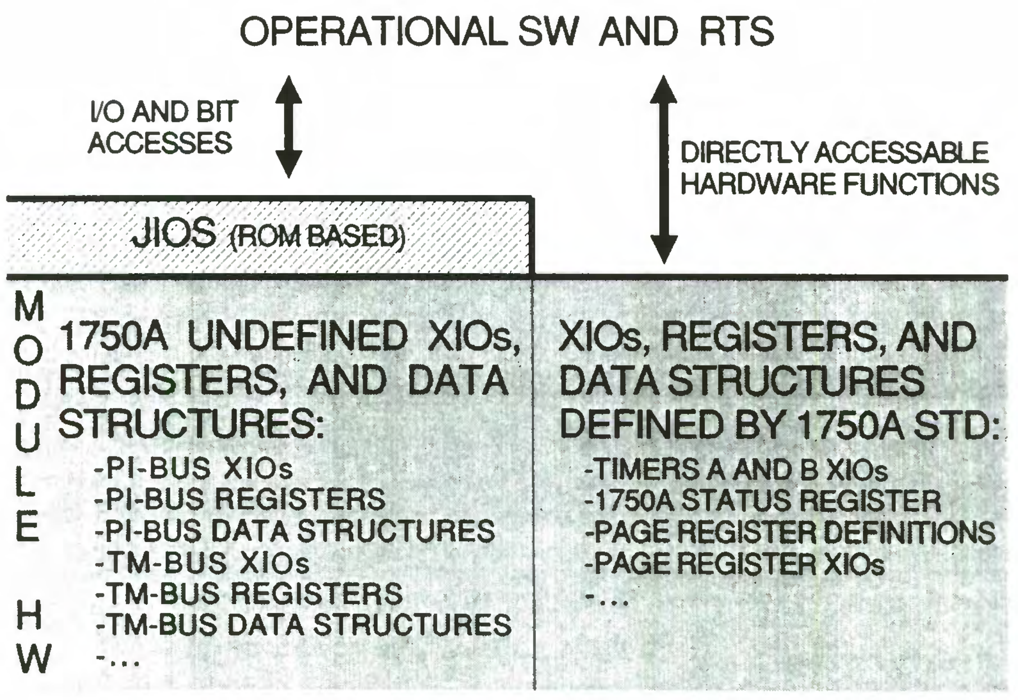

Figure 1. JIOS, Hardware, and Software Relationships

instead of more traditional XIO definitions which are targeted to specific hardware controllers. However, the linkage conventions established in IOBIDS are intended to offer a natural path to a complete hardware implementation. It is noted, though, that if there are to be no modifications to existing operational or RTS software, then a small part of the JIOS will always be present to provide a translation from the assembly language subprogram call to the hardware/XIO interface.

The linkage conventions will be used to establish vector tables and other data structures needed for a pure hardware implementation. The final step of the JIOS definition, then, is to define, in effect, an entirely new set of user defined XIOs which can be implemented conventionally. The flexibility that IOBIDS offers, though, allows existing designs to be compliant with JIAWG interface standards by providing ISA patches for non-compliant hardware.

At the time this paper was written, the PI-Bus and BIT portions of the JIOS were defined in the IOBIDS. However, the JIAWG TM-Bus Backplane Specification was not well-defined, so the TM-Bus interface will not be described in this paper. The remainder of this section gives an overview of the BIT and PI-Bus requirements of the JIOS as documented in the IOBIDS.

3.1 BIT Structure

The CAP-16 BIT implementations are as diverse as the backplane bus designs, and deriving common functions from this diversity is difficult. Therefore, the JIOS provides a functionally simple interface, with relatively uncomplicated reporting mechanisms. Only memory systems, CPU, and backplane systems are included in the JIOS BIT interface. Other vendor unique hardware cannot invoked from the JIOS.

CAP-16 systems offer power-up, initiated, and periodic BIT. The JIOS provides an interface only for periodic BIT, as that is the primary resource needed by the application software. However, a primitive initiated BIT can be constructed from the application software using JIOS procedures. These procedures allow the software to start and stop periodic BIT, recall the periodic test results, and test the CPU, memory, or backplane system individually.

Three types of tests can be called through the JIOS: CPU, memory, and backplane. Each of these procedures has an input parameter whose value is either destructive or non-destructive, which allows the application program to save the current data or context prior to invocation. Each procedure can be called individually, thus providing a simple initiated BIT. Periodic BIT executes all three types of tests continuously. Results from periodic BIT can be sampled by calling the procedure ACCESS_BIT_STATUS_WORDS.

The test results include a summary word, individual words for the separate procedures, and interfaces to the MIL-STD-1750 Fault Register (FT) and Memory Fault Status Register (MFSR). The summary word simply identifies what type of hardware failed, such as the CPU, PI-Bus A or B, discretes, memory, or others. The individual words for the procedures offer more detail as to the failure, such as the page of the memory which failed. The FT and MFSR interfaces are needed because the spare bits allowed by MIL-STD-1750 for these registers have been implemented differently by the various vendors.

The JIOS BIT package is intended to provide common test functions for access by application software. Given the diversity of implementations, the JIOS defines a conceptual set of primitives for which detailed software interfaces are described. Only this approach allows application software to be transportable and capable of rendering consistent results.

3.2 PI-Bus Structure

The JIOS categorizes the PI-Bus routines as follows:

* Master PI-Bus Interface - routines and data structure definitions used for transmitting messages over the PI-Bus.
* Slave PI-Bus Interface - routines and data structure definitions used for receiving messages from the PI-Bus.
* PI-Bus Interrupt Interface - routines which describes routines, data structures, and protocol for servicing PI-Bus interrupts.
* Error PI-Bus Interface - routines and data structure definitions for PI-Bus related error handling.
* Configuration and Control Interface - routines and data structure definitions for PI-Bus related configuration and control processing.

The remainder of this paragraph gives a further description of each category.

3.2.1 Master PI-Bus Interface.

The basic data structure for transmitting messages is the Communication Control Block (CCB). The CCB is a concept used by all the JIAWG PI-Bus vendors. A CCB contains the PI-Bus header words for the message to be sent, identifies the data to be sent, denotes whether an interrupt should be generated upon transmission of the message associated with the CCB, and denotes if there is another CCB "chained" to it. PI-Bus messages with/without extended headers are supported.

The interface also defines a data structure called the logical priority. The logical priority is be used for the interface when a message is transmitted over the PI-Bus.

This category includes routines to build CCBs, modify CCBs, execute CCBs, abort a CCB, allow a CCB chain to supercede a currently executing CCB chain, and set the logical priority to be used for PI-Bus transmissions.

3.2.2 Slave PI-Bus Interface

There are two basic data structures defined in the Slave PI-Bus Interface: label table and message report.

The label table is a concept used by all the JIAWG PI-Bus vendors - even though the label table entry definitions are not identical. A label table entry denotes whether the label is active, whether an interrupt should occur upon successful reception of a message to the label, where the message being received should be written to in memory, whether the buffer is busy, whether double buffering is enabled and if so, the address of the alternate buffer, and the maximum size message allowed to be received by the label.

The message report is returned to the PI-Bus interrupt handler when a received message is "popped" from the received message queue. Most JIAWG vendors refer to the received message queue as the Slave Receive List. The message report contains the PI-Bus header words (including extended header words for messages with extended headers) and a copy of the associated label table entry for label messages.

This category contains routines to set the maximum label value for the label table, to initialize the label table, and to read and write label table entries.

3.2.3 PI-Bus Interrupt Interface

The basic data structure defined by the PI-Bus Interrupt Interface is the interrupt queue. The interrupt queue logically contains a record for each PI-Bus interrupt that has not been "popped" by application code. Each interrupt queue record contains the cause of the interrupt: no service required, master error, slave error, message received, transfer complete, or self-test complete. Based on the cause, the record contains pertinent information needed by the application software to service the interrupt:

* no service - ignore
* master error - master error report
* slave error - slave error report
* message received - message report
* transfer complete - pointer to CCB processed
* self-test complete - status of self-test.

This category contains routines to determine the current PI-Bus status (as defined in the Configuration and Control Interface category), and to pop entries from the interrupt queue. In addition, there is a routine which is always called by the PI-Bus interrupt handler to allow the JIOS to perform any special activities that it may need to perform.

3.2.4 Error PI-Bus Interface

The Error PI-Bus Interface defines the Master and Slave Error Report data structures and includes a routine to invoke JIOS self-test of its PI-Bus features.

3.2.5 Configuration and Control Interface

The basic data structure defined by the Configuration and Control Interface category is the PI-Bus status word.

This category contains routines for reading the PI-Bus status, denoting whether the module will be suspendable when it is a PI-Bus Slave, setting and resetting the module as busy with respect to the PI-Bus, enabling and disabling the PI-Bus logical identifiers, reading the configuration image, reading the multicast acknowledge registers, writing vie interval A and B timers, writing the vie priority register, enabling and disabling the PI-Bus transceivers, resetting the module's PI-Bus Interface, switching the active PI-Bus channel, and reading and setting the PI-Bus system time.

4.0 The IOBIDS DEMONSTRATION

Despite the path to commonality that JIOS offers, this approach is both unconventional and, admittedly for reasons that will follow, not the best technical path. As a consequence, at least three important questions remain about the feasibility of the JIOS.

The first area of concern is performance. Adding an additional software layer will cost execution speed, but there is no simple way to estimate this loss. Some argue, in support of JIOS, that since current bus driver software is written in Ada and provides similar functionality as the JIOS, the loss will be minimal.

Another area of concern is program size. The requirement that the JIOS be an integral part of the module architecture means, in practice, that any software the vendor must use for JIOS will be stored in on-the-module's Start-Up Read-Only Memory (SUROM). Since data processor module real estate is very limited, a small SUROM is preferred. CAP-16 currently requires a 32 thousand 16-bit word SUROM, but this must contain the bootloader, parts of the executive, diagnostic code, and perhaps other vendor code, as well as the JIOS.

A third area of concern is information security. The interaction of the input/output software with the operational software raises security issues.

A demonstration program has been established to answer these questions. The demonstration will use the CAP-16 gate level simulation models produced as part of the ZyCad-Air Force program, Demonstration of Avionic Module Exchangeability via Simulation (DAMES). The CAP-16 subcontractors involved are IBM, Texas Instruments, and Unisys.

As part of the demonstration program, a subset of PI-Bus functionality of the JIOS will be implemented, as a prototype, for the IBM and TI Cap-16 models simulated in the DAMES program. An IOBIDS executive will be developed which makes calls to PI-Bus support routines. Two versions of the PI-Bus routines will be implemented. One makes calls to the JIOS. The other uses a conventional approach based on existing PI-Bus driver software. Based on the JIOS prototypes, both performance and program size estimates will be made. By comparing the timings of the JIOS and conventional implementation, any performance degradation incurred by using the JIOS will be estimated. The sizes for the demonstration JIOS will be used to estimate the memory size for a complete JIOS implementation. The security question will not be addressed in the demonstration. Based on these estimates a decision will be made as to whether the JIOS is an acceptable approach for the JIAWG.

5.0 JIOS DRAWBACKS

As alluded to above, one of the original goals of JIOS was to provide a software interface that could eventually be implemented entirely in hardware. However, as already noted, to avoid software changes for systems already using JIOS, a small part of JIOS must always be present. This part will be the entry point to JIOS calls, which translates a call to the appropriate XIO call. There are two other areas which have been discovered in defining the JIOS which should not be considered candidates for hardware implementation. First, there are many data structures assumed by the JIOS (e.g., PI-Bus Label Table) which must be allocated from the JIOS user memory, but must be initialized through JIOS calls. Thus, these calls do not lend themselves to a direct hardware implementation. This is required since the same logical data structure has different bit-by-bit definitions for the various vendor implementations. Clearly, a hardware implementation would allow user direct access to these data structures.

Another JIOS drawback is that the details of the vendor interrupt structures had to be hidden because of grossly dissimilar implementations. It is not obvious that the current JIOS design, a logical queue, leads in a natural way to a hardware implementation. For the PI-Bus, for example, there are several reasons an interrupt can occur (e.g., message received, message transferred, self-test complete, error). In order to accommodate the existing designs, the JIOS interface logically queues all unserviced interrupts. When the JIOS user "pops" an unserviced interrupt, the user has no control over which interrupt is popped. A hardware implementation would allow this capability. Beyond, these two concerns, the JIOS interface appears to be a good baseline for future XIO definitions.

6.0 SUMMARY AND CONCLUSIONS

The IOBIDS, and the JIOS it defines, is the product of three Government services, five prime contractors, and four computer subcontractors. The goal of the JIOS is to offer current MIL-STD-1750A data processor designer, despite I/O policies inconsistent with CAP-16 requirements, a method of compliance which will not cause substantial hardware redesigns. The JIOS standard will eventually mature so that most of the JIOS will become a pure ISA-level hardware standard, effectively defining a new set of user defined XIOs.

The JIOS is a unique approach which carries with it technical risk associated with performance and program size. The demonstration will provide data on these areas before the IOBIDS approach is imposed contractually.

Tri-service hardware commonality has proven to be a technically challenging goal to pursue. The cost and schedule benefits of achieving this goal, though, are so potentially large that the challenge should prove to be worth the time and dollars spent in meeting it. IOBIDS is one of many components needed to achieve this commonality.