SLAC-PUB-3008
November 1982
(I)

AN INTRODUCTION TO FASTBUS *

C. A. Logg
Electronics Department
Stanford Linear Accelerator Center
Stanford University, Stanford, California 94305

ABSTRACT

FASTBUS is a standardized modular 32-bit data-bus system for performing data acquisition, data processing, and control in high energy physics and other applications. It has been developed by the Fast System Design Group of the U. S. NIM Committee. Presented here is an overview of the FASTBUS hardware specification, the operation of the FASTBUS protocol, the implications that the use of FASTBUS has for software systems, and some of the computer to FASTBUS interfaces developed to date.

INTRODUCTION

FASTBUS (Ref. 1) is a standardized modular 32-bit data-bus system for use in high energy physics research and other applications where it may be necessary to handle very high data rates in a data acquisition, processing, and/or control situation. It has been designed by members of the high energy physics research community under the auspices of the U.S. NIM committee. Financial support has come from the Department of Energy. The primary design goal has been to create a standard that facilitates the implementation of very high speed data acquisition and data processing systems. This has been met by devising a system which provides for the parallel operation of many processors on independent bus segments, but which also allows the segments to link together to pass data between devices throughout the system.

Some other design goals have been:

+ extensibility - The standard should not preclude the use of new technologies; and for the future, established FASTBUS systems should be easily modified and extended.
+ flexibility - A system should be able to accomodate very high speed and very low speed devices, various network topologies, various addressing modes (the sending of commands to one device or many devices, with or without handshakes).
+ modularity - The standard should encourage the development of modular system components which can be used in many applications.
+ maintainability - The basic components of FASTBUS should be maintainable (that is, they should be built out of readily available and multiply sourced parts) and the problems of a system in operation should be diagnosable and repairable without requiring that the entire system be taken down.

A few of the features designed into FASTBUS as a result of considering these (and other) goals include: 32-bit device addresses, 32-bit internal device addresses, and 32-bit data units, all multiplexed on one set of 32 address/data lines; a diversified communications protocol with provisions for a variety of address and data transfer modes and synchronous and asynchronous communications; mechanical and electrical specifications; and various features to facilitate the development of software.

FASTBUS HARDWARE

The communications medium for a FASTBUS system is the set of signal lines known as the BUS. An electrically independent bus unit is known as a SEGMENT. FASTBUS devices can be connected together by a cable which contains the bus and the segment is then called a CABLE SEGMENT; or, grouped together into a crate with a backplane which contains the bus and the segment is then called a CRATE SEGMENT. A 19-inch FASTBUS crate (Figure 1) which can hold up to 26 modules is an example of a crate segment. The cable segment bus has just 60 signal lines, whereas the backplane segment bus has the basic 60 as well

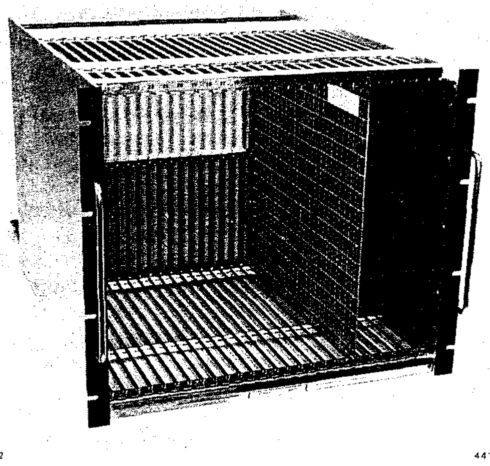

Figure 1: A 19-Inch FASTBUS Crate with a Kludge Card in Slot 6

* Work supported by the Department of Energy, contract DE-AC03-76SF00515.

(Contributed to the 1982 Fall DECUS U.S. Symposium, Anaheim, California, December 6-10, 1982)

as other signal and power lines. Figure 2 lists the signal lines and their identifiers. The backplane segment specification has also allowed for an optional auxiliary connector with up to 195 pins.

Signal Lines Common to Cable and Crate Segments

Mnemonic	Signal Name
AS	Address Sync
AK	Address Acknowledge
EG	Enable Geographic
MS	Mode Select (3 lines)
RD	Read
AD	Address/Data (32 lines)
PA	Parity
PE	Parity Enable
SS	Slave Status (3 lines)
DS	Data Sync
DK	Data Acknowledge
WT	Wait
SR	Service Request
RB	Reset Bus
BH	Bus Halted
AG	Arbitrtation Grant
AL	Arbitration Level (6 lines)
AR	Arbitration Request
AI	Arbitration Request Inhibit
GK	Grant Acknowledge

Additional Crate Segment Signal Lines

Mnemonic	Signal Name
TX	Serial Transmit
RX	Serial Receive
TP	T Pin
GA	Geographical Address (5 pins)
	Other voltage busses, digital & analog returns, daisy chain & returns, and reserved pins (62 pins)

Figure 2: FASTBUS Signals

Each segment is required to have ancillary logic, which contains the ARBITRATION TIMING CONTROL (ATC), GEOGRAPHIC ADDRESS CONTROL (GAC), terminators, system handshake logic, and bus RUN/HALT logic. The ATC circuitry resolves contention for use of the local bus segment. The GAC circuitry assists with the geographic addressing of devices on the segment.

A FASTBUS module is a FASTBUS device which is built on a FASTBUS MODULE CIRCUIT BOARD (MCB). When a module is plugged into any slot in an FASTBUS crate, it must connect to the crate segment and respond to the FASTBUS protocol. The module circuit board is approximately 367 millimeters high by 403 millimeters deep. Figure 1 shows a FASTBUS crate with a KLUDGE CARD (for wire-wrapping prototype modules) inserted in slot 6. The kludge card can accomodate approximately 300 16-pin IC equivalents.

There are two categories of FASTBUS modules: MASTERs and SLAVEs. A master module is one which can gain control (MASTERSHIP) of a segment and initiate operations on that segment. A slave module cannot gain mastership of any segment. It can only assert information on a segment in response to a specific request by a master. However slave modules can request servicing by asserting the SERVICE REQUEST (SR) line. The specification requires that all masters have some slave capabilities. These will be discussed below.

It is usually desireable to protect module control functions (such as internal enables, disables, and clears) so that it is not easy to unintentionally invoke them. To facilitate this, the standard has specified two separate internal device subdivisions. They are known as DATA SPACE and CONTROL SPACE. Each must be accessed via an explicit address cycle. In addition, several basic control, status, and information registers (contained in control space) have had their locations specified in the standard.

Various recommended and mandatory device features have been included in the specification to facilitate the creation of standardized software for handling FASTBUS systems. One requirement is that every module be accessable by its physical location on a segment. This is known as GEOGRAPHICAL ADDRESSING. On a crate segment there are 5 coded pins which enable each slot in the crate segment (and hence the module in that slot) to be uniquely addressed. On a cable segment, each device must have a set of switches which can be set to indicate the device´s geographical address.

Another specification is the explicit definition of certain CONTROL and STATUS REGISTERS (CSRs) in control space. One of the mandatory CSRs is CSR 0. CSR 0, when read, must return the ID (type or model number) of the device. CSR 0 together with the geographic addressing feature makes it possible to identify each device on a segment and hence generate a map of an entire FASTBUS system which can be used to verify a system´s configuration.

The implementation of CSR 3 is highly recommended. CSR 3 is defined to be a register which holds a 32-bit software settable address known as the LOGICAL ADDRESS. A logical address has three variable width fields (totaling 32-bits). The most significant field is known as the group field and is basically a segment address. The middle field is the module address field. The least significant field is the internal address field for specifying an address internal to the module. Once CSR 3 is loaded and the logical address recognition capability of the device is enabled, the device can be addressed by asserting this address instead of the geographic address. One advantage of logical addressing is that it allows the allocation of blocks of address space to modules. The logical address can then include internal address information which selects a part of a module. Geographic addressing can only select the module as a whole. Another advantage of logical addressing is that the module can be relocated within a segment (and possibly even the system) without any changes in the software applications programs, if the programs address slave devices via logical addresses.

Two segments may be connected together with a SEGMENT INTERCONNECT (SI) (Ref. 2). An SI monitors the activity on the two segments it connects. When an address which is recognized by the SI appears on one of the segments (called the SI´s NEAR SIDE because it is electrically nearest the master asserting the address), the SI responds by obtaining use of the bus on the other segment to which it is attached (the FAR SIDE), and then it asserts the address there. Device addresses may thus be propagated through many segments in this manner, until the segment on which the addressed device resides is reached. Note that each SI contains a look-up table which must be loaded with the addresses the SI is to recognize.

THE FASTBUS PROTOCOL

The FASTBUS protocol is the set of rules by which the devices utilize the bus signal lines for communication in the system. There are basically 3 parts to a FASTBUS operation: arbitration for bus mastership, master/slave address lock establishment, and the data transfer cycles.

Arbitration for Bus Mastership

Only one master can utilize the bus of a segment at any time. The Arbitration procedure (controlled by the ATC logic) resolves any contention for the use of a segment bus. The FASTBUS signal lines used in the arbitration are:

AR - arbitration request
AG - arbitration grant
AL - arbitration level (6 lines)
GK - grant acknowledge
AI - arbitration inhibit

A simplified explanation (the AI line is ignored here for simplicity) of the arbitration sequence (Figure 3) is as follows:

1. All masters requesting mastership of a segment assert the AR line.
2. The ATC recognizes the arbitration requests and, at the appropriate time, asserts AG which is an arbitration synchronization signal.
3. Each master which is participating in the arbitration then asserts its arbitration vector (CSR 8) on the 6 AL lines. A master's arbitration vector is its priority (and must have been initialized). During arbitration, each arbitrating master compares its internal arbitration level, bit by bit, with the level on the bus. If the bus level is higher, each master removes any lower order bits that it has asserted. When the ATC lowers the AG line, the master whose ALs match the ALs on the bus wins mastership. The winning master asserts GK to take mastership of the segment.

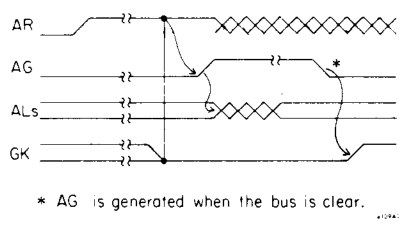

Figure 3: Lines Asserted During Arbitration

Establishing the Address Lock

After a master has obtained mastership, it must establish a connection to the device(s) with which it is going to communicate. This is known as establishing the AS-AK lock.

The FASTBUS timing, control, and address/data lines used in the establishment of the AS-AK lock are:

AS - address sync
AK - address acknowledge
MS - mode select (3 lines: MS<2:0>)
AD - address/data lines
SS - slave status (3 lines: SS<2:0>)
EG - enabled geographic
PA - Parity
PE - Parity Enable

Each device may have two separate address spaces (control and data space). The MS lines are used to indicate which address space is being accessed. It is also possible for a master to address more than one slave at a time. When more than one slave is begin addressed at a time, the master is said to be doing a BROADCAST. The SINGLE-LISTENER or broadcast state is also indicated via the MS lines. Figure 4 lists the MS codes and their meaning at address cycle time.

MS<2:0>	SIGNIFICANCE
0	data space - specific device
1	control space - specific device
2	data space - broadcast
3	control space - broadcast
4-5	reserved - specific device
6-7	reserved - broadcast

Figure 4: Address Type Specification

The master initiates the address cycle (Figure 5) by asserting the address of the slave on the AD lines, asserting the MS code for the kind of address cycle desired on the MS lines, and finally asserting the AS line. The slave, upon recognizing its address, asserts the AK line, and the AS-AK lock is then established. Note that the EG line is used to indicate whether an address is a geographic address (EG=1) or a logical address (EG=0). The GAC ancillary logic monitors the address cycles on a segment and generates EG accordingly. The SS lines are used to indicate various connection conditions. They are set by the slave before AK is generated if the connection was sucessful but troubled, or by the SI (who will generate AK) if there is a network failure and the address can not be propagated to the next segment. Figure 6 details the SS codes at address time.

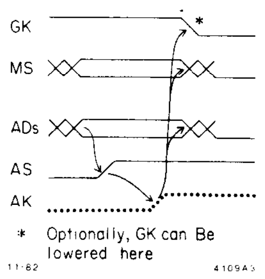

Figure 5: Address Cycle

SS<2:0>	SIGNIFICANCE
0	address recognized
1	network busy
2	network failure
3	network abort
4-5	reserved
6	invalid internal address (IA) in logical address - IA rejected
7	invalid internal address in logical address - IA accepted

Figure 6: Address Time SS Responses

Data Transfer Cycles

Once the AS-AK lock is established, the master can proceed with the data transfer cycles. The signal lines used in the data transfer cycles are:

DS - data sync
DK - data acknowledge
MS - mode select (3 lines: MS<2:0>)
AD - data lines
SS - slave status (3 lines: SS<2:0>)
PA - parity
PE - parity enable
RD - read (data transfer direction)

There are several different ways in which data can be transferred, as well as two different kinds of data which can be transferred. The MS lines are used to indicate the mode as well as the kind of data. Each module has an internal address register. This internal address can be written via a logical address cycle, and, read or written via a data cycle. Data transfer methods include handshake block transfers, random data cycle tranfers, and pipelined data transfers. The direction of transfer is controlled by the RD line. If RD=1, then the master is asking that the slave assert data for the master to read. If RD=0 then the master is asserting data for the slave to process. Figure 7 lists the MS codes and their meanings for the data cycle.

MS<0:2>	SIGNIFICANCE
0	random data cycle
1	handshake block transfer
2	secondary address (IA)
3	pipelined block transfer
4-6	reserved
7	reserved - **pipelined**

Figure 7: MS Codes for Data Cycles

Any combination of data cycles (which a slave and master are equipped to handle) can be concatenated together to perform the transfer of information between the two modules.

The master initiates the data cycle by:

1. asserting: MS to indicate the type of data transfer which is to be performed, RD to indicate the direction of the data transfer, and in the case of a write, the data on the AD lines;
2. and then asserting DS.

The slave responds to the initiated data cycle by

1. reading the RD line to determine the direction of the information transfer, decoding the MS lines to ascertain the type of data cycle,
2. then executing internally the indicated function, and, for a read, asserting the data on the AD lines. If the slave detects an error, it asserts an error code on the SS lines.
3. The slave then completes the data cycle handshake by asserting DK.

An example of data cycle concatenation is shown in Figures 8 and 9. A normal address cycle, such as displayed in Figure 5, must have been sucessfully completed.

Figure 8, section A shows a secondary address cycle which is used to set the internal address register in the slave module. The address contained in the IA register is the address of the location in the slave module where the next data written to the module will be placed, or where the data for the next read will be taken from.

Figure 8, section B, shows a random data write cycle. The master asserts MS=0, the data on the AD lines, and DS=1. The slave, after processing, returns DK=1. The master then drops DS, the slave drops DK, and the master proceeds to the next cycle.

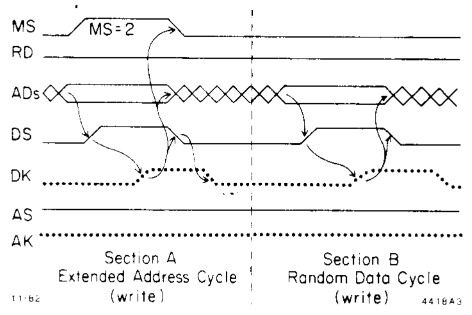

Figure 8: Write Cycle Examples

Figure 9, sections A,B, and C, show an example of a handshake block transfer read. Note that in the secondary address and random data cycles, DS must be dropped at the completion of each cycle. In a handshake block transfer (MS=1) every change in DS is used to indicate another data cycle.

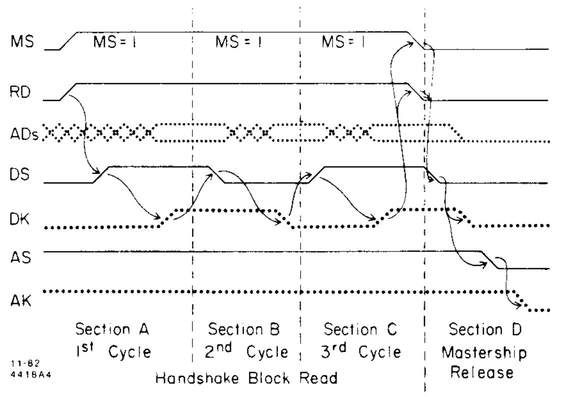

Figure 9: Handshake Block Read and AS-AK Lock Termination

Mastership Release

Once a master has obtained mastership, it can perform any combination of address and data cycles needed to complete its task, as long as it retains the GK assertion. However, it is permissable to release the GK line (as in Figure 5) as soon as the AS-AK lock is established, if the master is not going to perform any more address cycles. Mastership of the bus is released when the master has set AS=0 and GK=0. When the slave sees AS=0, it removes AK (Figure 9, section D).

SOFTWARE IMPLICATIONS OF FASTBUS

From early on it was clear that sophisticated software tools were going to be needed to handle complex FASTBUS systems. In early 1979, the software arm of the Fast System Design Group, called the Software Working Group (SWG) was formed. Its main function has been to define the issues inherent in handling FASTBUS systems, tackle SOME of those issues, and to work closely with the hardware group in defining hardware features which can aid in the development of software.

Early in 1979, the SWG tackled the subject of initializing a FASTBUS system. As indicated earlier in this paper, there are several options, registers (e.g., logical address, arbitration priorities), memories and other control features (including the SI route maps) which must be initialized before a FASTBUS system is operational. The system initialization subject will be examined here by looking at a hypothetical example.

Figure 10 shows a hypothetical FASTBUS system. The left side of the figure illustrates the topology of the data acquisition portion of the system. The right side illustrates a control system portion. Imagine the following:

+ The host computer reads data out of module S(1,1) when it is notified by S(1,1) that data is ready. (S(1,1) asserts the SR line to notify the host). The host computer also reads various environmental parameters out of S(3,1) at regular timed intervals, or when notified via a interrupt message from master M(3,1) that it should. S(1,1) and S(3,1) could be memory modules.
+ M(2,1) reads the data from the data acquisition portion of the system (segments S4, S5, and S6), reads environmental parameters from S(3,1), processes that information, and writes its results into S(1,1) which is connected to M(2,1) via a non-FASTBUS bus.
+ M(3,1) is responsible for collecting the environmental parameters from the control portion of the system and storing them in S(3,1).
+ M(7,1) monitors and controls the part of the control system attached to segment S7.
+ M(8,1) and M(8,2) work together handling the S8 segment.
+ All masters are programmable and can be reprogrammed if necessary to meet changing conditions.

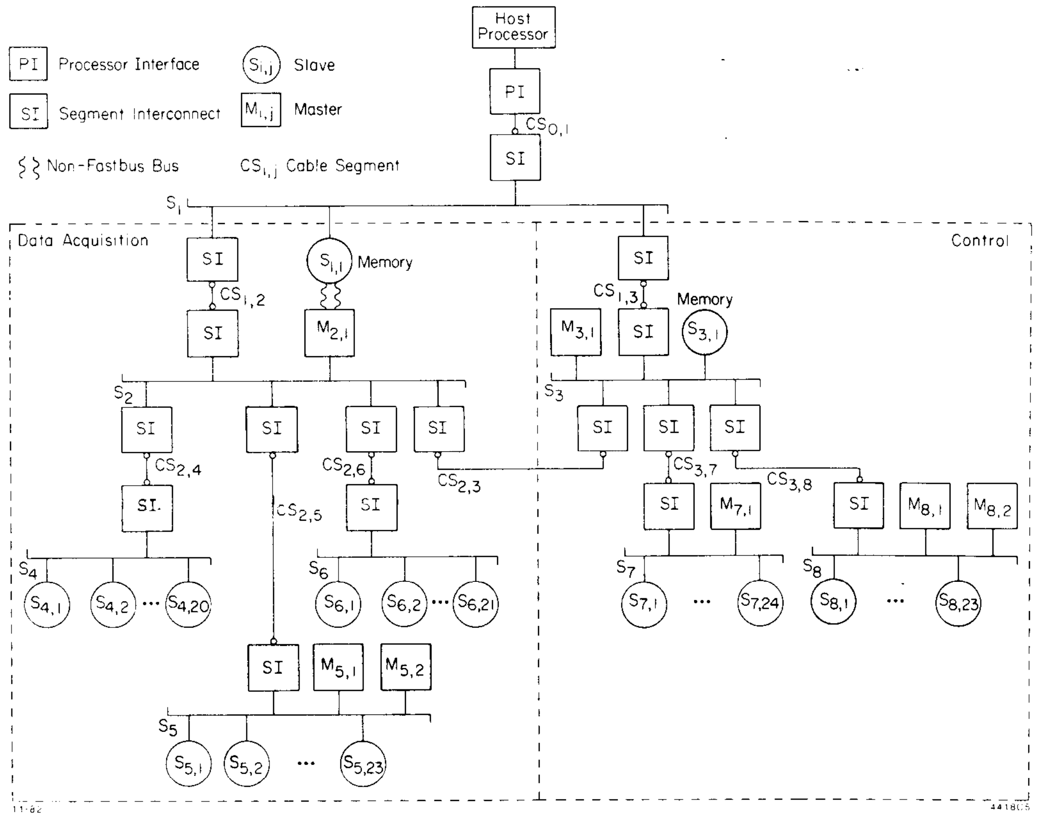

Figure 10: A Hypothetical FASTBUS Data Acquisition and Control System

In this system, the following are some of the items which would need initialization:

+ SIs must be initialized and loaded with route maps.
+ Modules must have their logical address registers (CSR 3) loaded and the logical address recognition enabled.
+ Masters must have their arbitration priorities (CSR 8) loaded.
+ Masters must have their programs downloaded.
+ The entire system must be started and synchronized.

The recommendations developed by the SWG for handling the initialization entail the following:

+ There should be one computer attached to a FASTBUS system which is identified as the HOST computer. The host is responsible for managing the system. This entails initializing the system and perhaps performing other run-time jobs such as maintaining an error log and verifying that the system is completely operational. Of course, the host can also perform other functions such as data taking and analysis!
+ A data base should be created on the host. It should contain the information needed to manage the system, such as: types and locations of all modules in the system, address space requirements, which modules need to have programs downloaded (and maybe the pointers to the programs), internal module registers that need to be initialized, information on system topology, and desired message and broadcast routes. It may also contain certain traffic projections such as the amount of activity expected on a segment for each event.
+ There should be access, display, and editing methods for the data base.
+ There should be available a program which performs an analysis of the information in the data base. This program determines the optimal message routes, allocates address space to the segments and the modules on them, and creates the tables (route maps) which are downloaded into the SIs.
+ In general, it is recommended that programs (which are downloaded into masters) address slave modules in a symbolic address independent fashion. Once all modules are allocated addresses, it is then necessary to resolve these symbolic addresses. This is referred to as the linking problem. This problem is yet to be effectively solved although the SWG has several ideas.
+ Finally, there should be a program which downloads the system, SI by SI, segment by segment, module by module.

A lot of effort has been put into studying the system initialization subject by the SWG. Several algorithms have been developed to handle it. These include a routing algorithm, an address space allocation algorithm, and an algorithm for systematically loading all the initialization information into a system. A feasibility implementation (Ref. 3) was done in 1979, and currently Fermi National Accelerator Laboratory (FNAL) is designing a system management software package (Ref. 4) and implementing the algorithms in FORTRAN.

One thing that became clear when the SWG first started studying the various FASTBUS software topics, was that a common communications language was needed. When coding for the operating system and bus interface level, it may be difficult to write general purpose code which can be widely shared, as that code must often be done in assembly language and is very computer specific. However at the higher coding levels (such as implementing a system loading algorithm and in many instances the applications program level) it is feasible and desireable to write transportable and sharable code. Over the past year work has been done on defining a set of Standard Subroutines for FASTBUS (Ref. 5) and they will soon be published.

In any FASTBUS system there are several control, status, and information elements contained in the control space of modules which are common to many modules. The SWG has studied these and developed recommendations for standard assignments for many of them. It is hoped that these efforts will facilitate the development of code modules which can be shared among many FASTBUS systems.

The SWG meetings have also served as a forum for the discussion of many hardware and software ideas and developments. These discussions have proved very valuable, and a FASTBUS Software Resource Guide (Ref. 6) is currently being developed. It will include an overview of the various FASTBUS software packages available, of the issues of managing FASTBUS systems, of various computer/FASTBUS interfaces, mandatory and recommended module features, and other miscellaneous information such as good practices and conventions.

There are still other topics to be studied. The data base, its creation, display, access and editing has not been solved. Currently some members of the community are exploring commercially available data base packages (including DEC's Datatrieve) for a solution. FASTBUS system simulation has only been briefly mentioned. It would certainly be handy to be able to create a data base for a system under design and run a simulation of that system to check for traffic loading and bus contention problems. The linking problem mentioned earlier, serial network protocols, buffered interconnect (an asynchronous method of passing messages between devices) message formats, and many others still need to be solved, defined, or resolved. It is envisioned that the FASTBUS Software Resource Guide will evolve and be periodically updated to pass along the information on these and other topics related to FASTBUS.

COMPUTER TO FASTBUS INTERFACES

Over the past two years various computer to FASTBUS interfaces have been developed.

I/O Register to FASTBUS Interface (IORFI)

The Input/Output Register to FASTBUS Interface (IORFI) (Ref. 7) is a computer-non-specific interface. It can be used to connect any computer (which has parallel I/O ports and associated strobes) to a FASTBUS backplane segment. The FASTBUS backplane interface is built on a single module circuit board. It is connected to a computer by two 16-bit parallel input registers and two 16-bit parallel output registers. IORFIs have been used extensively in the checkout of other interfaces, SIs, and FASTBUS modules in Canada, Europe, and the United States.

UNIBUS Processor Interface (UPI)

The UNIBUS Processor Interface (UPI) (Ref. 8) is a list driven, microcoded interface for connecting a UNIBUS machine to a FASTBUS system. The UPI allows a processor on the UNIBUS to execute any FASTBUS operation (except pipelined transfers), to transfer data between the UNIBUS and a FASTBUS system, and to detect errors. The UPI can also respond to FASTBUS interrupts and service requests. UPIs have been used (Ref. 9) in an experiment at Fermi National Accelerator Laboratory (FNAL) to take data, and at the University of Illinois in the checkout of the segment interconnects. Currently there is also a project in progress to attach a VAX to FASTBUS via a UPI.

VAX-FASTBUS Interface

A FASTBUS host interface (Refs. 10,11) for VAXs has also been developed. It is a list processing microprocessor controlled interface, and has been constructed by a connection to the VAX DR-32 Device Interconnect (DDI) implemented via the DEC DR-780 channel. It is described in detail in another paper in these proceedings.

Other Microprocessor-FASTBUS Interfaces

There are several other FASTBUS interfaces which have been developed (or are under development). The KEK National Laboratory for High Energy Physics at Hiroshima University in Japan (Ref. 12) and the European Organization for Nuclear Research (CERN) in Geneva, Switzerland (Ref. 13) are two research centers which are very active in FASTBUS development.

SOFTWARE

Over the past two years during the prototype development phase, various software packages have been developed for diagnostic and test-bench situations. The FASTBUS Diagnostic Language (FDL) (Ref. 14) was developed at the University of Illinois. It is a high-level, conversational language designed to provide quick software access to FASTBUS hardware for diagnostic purposes. It has been heavily used in the development and checkout of the segment interconnects. It also contains extensive slave simulation code for the IORFI.

At SLAC, FORTH (Ref. 15) has been chosen as the test-bench language. The FASTBUS Diagnostic Operating System (FBDOS) (Ref. 16) is a package of FORTH words which provides a set of versatile commands for accessing FASTBUS. It is intended primarily for use in prototype module checkout, production module testing, and module diagnostics.

There have been at least four implementations of the Standard Subroutines for FASTBUS. The first one, implemented at FNAL for the PDP-11 UNIBUS Processor Interface has actually been used for a data acquisition applications program in an experiment there (Ref. 17). Currently FNAL is adapting their standard subroutine package to work with the VAX-UPI connection.

CERN has implemented a set of FORTRAN routines (Ref. 18) based on the Standard Subroutines for FASTBUS and a computer connection to the IORFIs via CAMAC. This package utilizes the NIM/ESONE Standard CAMAC Subroutines and is currently running on the NORSK-DATA computers under BASIC/FORTRAN and on PDP-11s under CATY/FORTRAN.

An implementation of the Standard Subroutines was done at Brookhaven early in 1982 for the VAX-FASTBUS Interface.

THE FASTBUS SNOOP

The FASTBUS SNOOP (Ref. 19) is a special diagnostic module which is currently under development. It has a very fast front end "silo" which allows it to make a record of the FASTBUS operations on a segment and thus serve as a logic analyzer. The Snoop will also be able act as a master on a system. A series of Snoops in system will be connected via a serial network and thus be able to operate independently of FASTBUS. At least one of the Snoops in a system will have a sophisticated workstation (Ref. 20) attached to it. A user sitting at that workstation will be able to communicate with the other Snoops and thus diagnose trouble throughout the system.

CONCLUSION

The FASTBUS standard has been under development for several years. The specification has now been finalized. During the development of FASTBUS, considerable thought has also been given to the software implications. The desires of the software people have occasionally conflicted with the desire of the hardware people for a simple and low-cost hardware specification. However, in most instances, acceptable compromises have been found which hopefully will keep the real overall system implementation cost down.

ACKNOWLEDGMENTS

The information presented here is condensed from many sources. Where practical, references have been given. The work described in this paper was done by a very large team of people from Canada, Europe, and the United States. It is impractical to list all the team members here. In particular, I wish to thank Dave Gustavson, Ray Larsen, and Leo Paffrath for their encouragement and support of my interest in FASTBUS.

REFERENCES

1. FASTBUS Modular High Speed Data Acquisition System for High Energy Physics and other Applications, Tentative Specification, U.S. NIM Committee, June 1982.

2. The FASTBUS Segment Interconnect, R. Downing and M. Haney, IEEE Transactions on Nuclear Science, NS-29, No. 1 (1982), page 94.

3. Software for Managing Multicrate FASTBUS Systems, S. Deiss and D.B. Gustavson; to be published in the IEEE Transactions on Nuclear Science, February 1983.

4. FASTBUS Software Progress, D. Gustavson, SLAC-PUB-2996, P.O. Box 4349, Stanford, CA 94305.

5. Standard Subroutines for FASTBUS, FASTBUS Software Working Group, Ruth Pordes (Editor), FNAL, P. O. Box 500, Batavia, Illinois 60510.

6. FASTBUS Software Resource Guide, Richard Brown (Editor), Department of Physics, University of Illinois, Urbana, Illinois 61801.

7. Input/Output Register to FASTBUS Interface, C.A. Logg and L. Paffrath, SLAC-PUB-2972, to be published in the IEEE Transactions on Nuclear Science, February 1983.

8. A UNIBUS Processor Interface for a FASTBUS Data Acquisition System, M. Larwill, E. Barsotti, T.D. Lagerlund, L.M. Taff, and J. Franzen, IEEE Transactions on Nuclear Science, Volume NS-28, No. 1 (1981), page 385.

9. Data Collection form FASTBUS to a PDP-11 through the UNIBUS Processor Interface, M. Larwill, et.al., to be published in the IEEE Transactions on Nuclear Science, February 1983.

10. FASTBUS Host Interface for VAX/VMS, E. J. Siskind, to be published in the IEEE Transactions on Nuclear Science, February 1983.

11. FASTBUS VAX Host Interface for VMS, E. J. Siskind, published in these proceedings.

12. A Status Report of FASTBUS at KEK, Y. Arai, KEK, National Laboratory for High Energy Physics, Hiroshima 730, Japan; to be published in the IEEE Transactions on Nuclear Science, February 1983.

13. Status of FASTBUS in Europe, H. Verweij, to be published in the IEEE Transactions on Nuclear Science, February 1983.

14. FASTBUS Diagnostic Language (FDL) Reference Manual for the UNIBUS Processor Interface, Sept. 1982, Dave Lesny, Seth Abraham, Steve Coffman, Keith Nater, Loomis Laboratory of Physics, University of Illinois, Urbana, Illinois 61801.

15. SLAC ELD LSI-11 FORTH User's Guide, August 1982, Connie Logg, SLAC, P.O. Box 4349, Stanford, California 94305.

16. FASTBUS Diagnostic Operating System (FBDOS), August 1982, Connie Logg, SLAC, P.O. Box 4349, Stanford, California, 94305.

17. Standard Subroutines for FASTBUS and their Implementation for a PDP-11 RT-11 System, using the UNIBUS Processor Interface, R. Pordes, to be published in the IEEE Transactions on Nuclear Science, February 1983.

18. CERN Software for FASTBUS, E. M. Rimmer, DD Division, CERN, 1211 Geneve 23, Switzerland.

19. FASTBUS Snoop Diagnostic Module, H.W. Walz, IEEE Transactions on Nuclear Science, Volume NS-28, No. 1 (1981), page 380.

20. A "Front Panel" Human Interface for FASTBUS, D. Gustavson, T.L. Holmes, L. Paffrath, and J. Steffani, IEEE Transactions on Nuclear Science, Volume NS-28, No. 1 (1981), page 343.