SLAC-PUB-2835
October 1981
(E/A)

FASTBUS DEMONSTRATION SYSTEMS*

L. Paffrath, B. Bertolucci, S. Deiss, D. Gustavson, T. Holmes
D. Horelick, R. Larsen, C. Logg and H. Walz
Stanford Linear Accelerator Center
Stanford University, Stanford, California 94305

E. Barsotti, M. Larwill, T. Lagerlund, R. Pordes and L. Taff
Fermilab, P. O. Box 500
Batavia, Illinois 60510

R. Brown, R. Downing, M. Haney, B. Jackson, D. Lesny, K. Nater and J. Wray**
University of Illinois, Loomis Lab of Physics
1110 W. Green Street, Urbana, Illinois 61801

ABSTRACT

This paper will provide a demonstration of basic FASTBUS hardware and test software. The systems will include single crate segments, simple computer I/O, a fast sequencer and memory, some simple diagnostic and display devices and a UNIBUS to FASTBUS processor interface. The equipment will be set up to show the basic FASTBUS protocols and timing transactions, as well as some of the general initialization software features.

INTRODUCTION

FASTBUS is a standardized modular 32 bit data-bus system for data acquisition, data processing, and control. A FASTBUS system consists of multiple segments which can operate independently, but also link together for passing data. FASTBUS operates asynchronously to accommodate high and low speed devices, using handshake protocols for reliability. It can also operate synchronously for maximum data transfer speed. For a detailed description of FASTBUS, see Ref. 1.

As of October 20, 1981, the prototyping period will end, and the FASTBUS standard will be firm. This poster session presents both hardware and software FASTBUS systems. There are two hardware systems; one contains a simple computer interface, a fast sequencer and memory, the other a UNIBUS to FASTBUS interface together with a memory and simple display. A FASTBUS System Manager software system is presented as well as a brief summary of the FASTBUS Diagnostic System Status.

A FASTBUS BACKPLANE SEGMENT DEMONSTRATION

This demonstration utilizes three early prototype FASTBUS modules: a memory module, a sequencer, and an I/O Register to FASTBUS Interface. The software used to perform this demonstration utilizes the FASTBUS Diagnostic Operating System (FBDOS) software which is being developed for use in prototype development and hardware checkout. Because of recent FASTBUS protocol specification changes, the hardware used in the demonstration does not precisely match the specification, but the demonstration is still relevant since the basic concept of a FASTBUS operation remains the same.

Introduction

A 19 inch FASTBUS crate, which can hold up to 26 modules, is an example of a backplane segment. Each slot in the crate (and thus the module in that slot) can be uniquely accessed. The address of the slot in which a module resides is known as the module's GEOGRAPHIC ADDRESS.

A FASTBUS backplane segment has two attached ancillary logic boards. They are the Enable Geographic (EG line) Generator, and the Arbitration Timing Controller (ATC).

There are two categories of FASTBUS modules: MASTERs and SLAVEs. A master module is one which can gain control (MASTERSHIP) of a segment. A slave module cannot gain mastership of a segment. It can only assert information on the segment in response to a specific request by a master. Slave modules, however, can request servicing by asserting the Service Request (SR) line. All master modules must have slave capabilities.

Various recommended and mandatory module design features have been included in the specification to facilitate the creation of intelligent software for handling FASTBUS systems. One specification is the explicit definition of certain CONTROL and STATUS REGISTERS (CSRs). One of the mandatory CSRs is CSR 0. CSR 0, when read, must return the ID (type or model number) of the module. This mandatory feature makes it possible to identify the contents of each slot in a segment and hence generate a map of an entire FASTBUS system, segment by segment.

Another highly recommended feature is the implementation of a CSR to hold a software settable address. This address is known as the LOGICAL ADDRESS. Once this CSR is loaded and the logical address recognition enabled, the module can be addressed by asserting this address instead of the geographical address on the bus. The primary advantage of logical addressing is that it allows the allocation of as much address space as is needed by each module. The logical address can thus include internal address information which selects a part of a module, while geographical addressing can only select the module as a whole. Another advantage of logical addressing is that the module can be relocated within any software changes in the masters (if the masters address modules by their logical addresses).

Phases in a FASTBUS Operation

There are basically 4 phases in a FASTBUS operation. These are the ARBITRATION, the ADDRESS cycle, the DATA cycle, and the BUS RELEASE phases.

Arbitration is the first phase in which a master must participate. Only one master can utilize the bus of a segment at any time. The arbitration resolves any contention which there may be for the use of the bus.

Once mastership is gained, the master addresses the module(s) with which it is going to communicate. The address cycle results in the establishment of the link between the master and slave(s). There are four kinds of address cycles: single-listener data space, single-listener control space, multiple-listener (broadcast) data space, and multiple-listener control space.

Once a master has established the link, it can proceed to perform any data cycles necessary. There are four kinds of data cycles: random [used to

* Work supported by the Department of Energy, contract DE-AC03-76SF00515.
** Work supported by the Department of Energy, contract DE-AC02-76ER01195.

(Invited paper presented at the Nuclear Science Symposium, San Francisco, California, October 21-23, 1981.)

transfer one 32 bit word to (write) from (read) a module], extended address (used to read or write a module's internal next transfer address), handshake block transfer, and non-handshake block transfer.

When an operation is complete, the master may either proceed with another address and data cycle sequence, or release mastership of the segment so other masters can have access to it.

The I/O Register to FASTBUS Interface (IORFI)

The IORFI provides a means of interfacing a computer to FASTBUS. It is connected to a processor via two 16 bit output registers (OR1,OR2) and two input registers (IR1,IR2). One of the output registers (OR1) is used to specify the interface function that is to be performed when the interface is accessed via the Data-in Register (IR2) or the Data-out Register (OR2). The other input register (IR1) is used to read the direct status of some of the FASTBUS lines independently of OR1 (see Ref. 2 for a detailed description of the IORFI).

The Memory Module

The memory module being used in this demonstration has 256 words in data space and 4 CSR registers. CSR 0 is the ID register. The other CSRs are used as the logical address register, a run options register, and an error counter register. The module can execute random, extended address, and handshake block transfer data cycles. See Ref. 3.

The Sequencer

The sequencer module is being used as a master which can perform high speed FASTBUS operations. The sequencer has only control space addresses. The control space is divided into three sections. These are the status registers, the control memory, and the data memory. The sequencer is operated by loading encoded operation words (the sequencer program) into the control memory. The data memory is used as the source of the 32 bit AD line values to be used for address and data write cycles, and as the destination of data read during FASTBUS read cycles. See Ref. 3.

The FASTBUS Diagnostic Operation System (FBDOS)

This operating system, written in FORTH, is being developed for use in FASTBUS system and module check-out. Currently the FBDOS (Ref. 4) contains routines for performing various kinds of FASTBUS transfers, a Sequencer Program Assembler (Ref. 5), and facilities for monitoring FASTBUS operations.

A layered approach has been used in the design and implementation of the system software. The top layer, called the Complete FASTBUS Operations (CFO) layer is composed of words which perform complete FASTBUS operations. The next layer is composed of FASTBUS Cycle Operations and is known as the FCO layer. The FCO words are used to create the CFO words. However, they are available to the user who wishes to create his own combinations of FASTBUS Cycle Operations. They can also be called individually to single step through a FASTBUS operation. For a complete description of the FCO and CFO layers, as well as the other layers of the system, see Ref. 4.

The FBDOS contains several debugging facilities. The most widely used is the FB command. This command prints symbolically on the terminal the state of the bus. The IDRFI WT generation logic facilitates the development of FASTBUS instruction tracing software. Thus, via the IDRFI, the operator can single step any FASTBUS operation.

This demonstration shows many FASTBUS features: geographical addressing to locate and identify devices, logical addressing which allows devices to be position-independent, use of the WT line to monitor the state of the bus, the various data-cycles including read-modify-write and the multimaster arbitration capability. Many of these features were designed into FASTBUS to facilitate the development of diagnostic and system software for multiprocessor environments.

A DEMONSTRATION OF A FASTBUS SYSTEM USING A UNIBUS TO FASTBUS INTERFACE

This demonstration shows data acquisition using a UNIBUS Processor Interface (UPI). See Ref. 5. The UPI allows a processor on the UNIBUS to execute any FASTBUS operation (except non-handshake block transfers), to transfer data between UNIBUS and FASTBUS, and to detect errors. The UPI will also respond to FASTBUS interrupts and Service Requests, interrupting the processor on the UNIBUS, and enabling the processor to determine the source of the interrupt and thus to respond to it.

The UPI has two functional components:

1) Two FASTBUS Segment Drivers (FSD) which allow the processor to execute any FASTBUS operation and inspect the results. Each FSD is capable of list processing. Several processor words are required to start each operation or list of operations, but once started each FSD will operate on its own and signal the completion by setting a Ready bit in a register. This bit can optionally cause an interrupt. List elements are stored in processor memory and fetched as they are executed via DMA transfers. Results of the execution of each list element are returned to a separate status block in memory. FSD Block Transfer operations are performed by hardware logic capable of multi-word transfers, either between UNIBUS memory and FASTBUS devices, or from one FASTBUS slave to another (via a hardware-controlled read cycle followed by a write cycle for each word transferred, or via a "burst" mode involving internal buffering of more than one word. The FSD's are controlled by microcode stored in a prom in the FSD.
2) A "FASTBUS Interrupt Receiver" (FIR). This responds to interrupt messages from FASTBUS devices and to FASTBUS Service Requests (SRs) by interrupting the processor (if the interrupt is enabled). There are two FIR ports and one SR port, each with a separate interrupt vector and interrupt-enable bit, and each jumperable to any UNIBUS vector level.

The UPI consists of two FASTBUS modules (or one double-width module) and one relatively simple UNIBUS module. One FASTBUS module, the FSD, consists of the FSD hardware and is interfaced by microcode. The second module, the FASTBUS Master Interface (FMI), contains the FIR and the FASTBUS and UNIBUS interfaces. There is a data-plus-control bus between the FASTBUS modules and the UNIBUS module. The UNIBUS module is called the UNIBUS Master Interface (UMI).

The demonstration shows data acquisition from a FASTBUS system using the FSD list processor interface between a PDP-11 UNIBUS and FASTBUS. The system demonstrates that the FASTBUS speed can be used in real time systems to gather data. The efficiency of block transfers, and the use of list processing devices on FASTBUS host interfaces, can reduce UNIBUS overhead. This relieves the high level processor of the time-consuming task of data gathering.

In the demonstration, lists of FASTBUS operations are performed by the UPI after being initiated by the PDP-11. Once the list is started, there need be no more intervention by the PDP-11 processor until its completion.

The FASTBUS crate contains the two-module UPI, a FASTBUS memory module with 240 data locations, and a simple display module. The simple display module is used to monitor the state of the FASTBUS lines to show

that the FASTBUS segment is active and data is being transferred.

The memory module is addressed using a logical address. This address is written into control register 1 of the memory module, at initialization time, by the PDP-11.

In the demonstration, data are written and read from the memory module in block transfer mode. A program in the PDP-11 first downloads the microcode for the UPI into a read only memory, and resets the UPI.

A modified version of the data analysis program MULTI is used to read and write data to the memory module. The data read is displayed in graphical form. The data transfer routine of MULTI uses the standard routines for FASTBUS to construct the FASTBUS operation lists to be executed by the UPI, and to instruct the UPI when to begin its operations.

Parameters in MULTI may be set during the demonstration to change the data written to the memory module, and to change the mode of the FASTBUS reads being done. The data from FASTBUS to UNIBUS memory may be transferred in 32 bit or 16 bit mode. In this latter mode only the low order 16 bits of each data word are transferred to the UNIBUS. The UPI may be instructed to ignore a particular FASTBUS operation or to change the burst size of each block transfer.

Parameters may also be set in MULTI to instruct the data transfer routine to give a user defined list of FASTBUS operations to the UPI for execution.

FASTBUS SYSTEM MANAGER

The FASTBUS System Manager is a software system which assigns each device an address or a range of addresses and specifies the communication paths over which any two modules in the system can communicate. To do this the system manager must maintain a data base that describes system topology as well as details of each component module in the system. In the long run the system manager will evolve to include facilities for error recovery, module diagnostics, module initialization and system verification procedures to ensure that the actual physical configuration agrees with its description in the data base. Through the system manager the experimenter will then be able to bootstrap the system, configure around faulty modules, and enable/disable experiment data collection runs.

The System Manager presently consists of the following major components:

(1) Virtual Memory Data Base Access Mechanisms

The System Manager data base is partitioned into two files. The Network file describes system topology, and the Name file gives infrequently used detailed information about each system component. Each file is a random file of fixed length records with variant record types. These files can be very large, perhaps several hundred kilobytes. To simplify access to these files a small number of record buffers are maintained in memory. Records are paged into these buffers on a demand basis. This allows the access to file records as if they were present in a large memory resident array of records. The number of the record requested is first passed through a function which takes care of paging, replacement, and buffer management. Then the function returns the number of the buffer in which the requested record was found or placed.

(2) Data Base Linked List Processor

The records in the files are linked together into linked lists of several types such as a list of segments, a list of modules on each segment, etc. A simple linked list processor was implemented to provide a uniform access mechanism. The functions provided are: (a) get the head of the list, (b) get the next record in the list, (c) insert a record in middle of the list, (d) remove a record from a list, and (e) find a record in a list. Each of these routines takes the list type as an argument in order to know how to find and manipulate the lists' linkage pointers.

(3) Route Map Generator

The route map generator uses three square matrices and an iterative approach to find the shortest reversible unique route between every possible pair of segments. Where there is a choice it takes the route that has the smallest window, i.e., has the largest minimum size SI in the path. This conserves address space for reasons beyond explanation here. The three matrices are as follows. The PATH matrix shows at each row/column intersection which segment to go through first to get from the segment represented by the row to the segment represented by the column. Initially PATH contains nonzero entries only where there is a direct connection between segments via an SI. PATH also shows what size the smallest SI is on that path (i.e., an 8 bit SI is smaller than a 12 bit). After each iteration the PATH matrix is updated, and another matrix called NEWS shows what new paths were found. The NEWS is OR'd with a running total matrix (RTM) to see if any routes remain to be found. The algorithm continues until the RTM is all ones, success, or RTM is not all ones and NEWS is all zeros, failure.

(4) Address Space Allocator

Address space is allocated for modules within segments and for segments within systems using the same procedures. After initializing data structures we have a list of fixed segments/modules and their sizes chained together in order of ascending address. Also, we have a list of mobile segments/modules that can be plugged in anywhere they fit. The mobile ones are chained together in order of descending size. The algorithm then takes the largest mobile one and plugs it into the first hole between fixed ones that it will fit. The process is repeated until all mobile objects have been merged. The algorithm is not optimal, but it will serve until larger system issues regarding multiple SI sizes are solved.

(5) Data Base Editor

The data base editor parses command lines into keyword strings (three characters each) after removing noise words and ignorable characters. After recognizing a keyword it branches immediately to the processor for that keyword and looks for additional qualifier keywords. If something is missing or a parameter is out of range, input file processing is stopped with an operator error message. By observing a small set of vocabulary and syntax rules the user can issue editor commands in a format approaching 'FASTBUS English'. Commands may be concatenated on one line or they can span multiple lines because the command stream is considered as a long text string terminated by an EOF.

(6) System Loader/Initializer Simulation

The system loader simulates the overall sequence of events that take place when a system is initialized. That is, it 'resets' segments, 'loads' their modules, then 'loads' their SI route map tables in the order they would be handled in a real system initialization. This is a very specific order that represents a depth-first scan through the system. This approach is required to ensure that all segments get loaded, and so that we always build our bridges to the next segment as we go without burning any behind us. When real SI hardware is available and other software issues have been resolved, these simulated actions can be replaced by real FASTBUS operation sequences.

The system manager software requires several enhancements before use in a real FASTBUS system. As mentioned, real FASTBUS drivers must be added. The work on specification of standard FASTBUS subroutine packages will guide the FASTBUS driver effort. The system manager data base is open-ended at this time pending further study of how to include diagnostic, error recovery, and initialization procedures with a general solution. Work remains to be done to specify how the device addresses are communicated to linkage editors that service various intelligent modules within a system. Then we need an integrated linkage editor approach that produces quasi-device-independent load modules for use by a generalized device initializer/ loader. Perhaps most important is the need to generalize the address space assignment and route map generator algorithms to handle systems which have many different sizes of segment interconnects. Such systems present subtle constraints on the allocation of address space which in turn affects route map generation. Finally, some effort is needed to find ways to make all such algorithm implementations transportable to the many research labs.

FASTBUS DIAGNOSTIC SYSTEM STATUS

The FASTBUS diagnostic system is currently under development. The prototype hardware has been delayed from the original expectations for many reasons.

Now that the specification has stabilized, the circuit board layout is being digitized for producing the printed circuit artwork on a computerized photo-plotting system. Meanwhile, wire-wrapped partial implementations are being fabricated which will allow testing of the microprocessor and network-related features, using the actual MC68000 processor and serial network hardware.

While awaiting completing of the real hardware, software is being developed on substitute hardware which simulates parts of the final system.

Low-speed implementations of the serial network hardware were added to existing Z80 microprocessor systems, and the basic network control algorithms were developed and debugged.

The system is being written in FORTH, an interactive language which combines compiling, interpreting, assembling, editing and operating system features in a very compact package. The needed interrupt-driven multitasking system was implemented in FORTH, using a version called FIG (FORTH Interest Group) FORTH, which is widely available and is in the public domain.

This multitasking system was first implemented on the Heath H89 intelligent terminal/microprocessor system, which will be the console and floppy disk storage device for the diagnostic system. It was then ported to the Z80 network test machines, and then the FORTH nucleus was ported to the Motorola KDM MC68000 development board.

We recognized a need for a standard version of FORTH which could work on a wide range of machines, which would make portability of programs and pro grammers easier. In particular, a different version of FORTH from an earlier branch of FORTH's evolution was being used widely on LSI/11's at SLAC, for CAMAC based systems and miscellaneous test-bench work.

The FORTH community has been working toward a standard to solve these same problems of portability. The current version, called FORTH 79, seems to us to be the best base to use for a standardized FORTH system.

FORTH 79 has now been implemented on the Z80 (H89 stand-alone and under the CPM operating system), the LSI/11, and the MC68000.

The multitasking system is now being ported to the MC68000 and documentation is being prepared. This work should be complete in the next month or two, at which time the wire-wrapped Snoop prototype should be available for testing. This will provide a more useful model of the final Snoop hardware for our software development than we have had to date. For detailed descriptions of the Snoop and diagnostic system (see Refs. 6 and 7).

SUMMARY

This demonstration has shown some of the major features of FASTBUS and the current state of the prototyping effort, which will be completed in FY '82.

ACKNOWLEDGEMENTS

I would like to thank all the co-authors for their contributions, and especially R. Larsen and C. Logg for their many helpful suggestions.

REFERENCES

1. Working Group Document - Tentative Specification, U.S. NIM Committee, "FASTBUS Modular High Speed Data Acquisition System for High Energy Physics and Other Applications," available from L. Costrell, NBS, Washington, D.C., August 20, 1981.
2. C. Logg and L. Paffrath, "I/O Register to FASTBUS Interface," SLAC-Internal Note, August 1981.
3. B. Bertolucci and D. Horelick, "Design of a FASTBUS Programmable Sequencer Module and Memory Module," Paper 2J3, this conference.
4. C. Logg, "FASTBUS Diagnostic Operating System," SLAC Internal Note, August 1981.
5. "Unibus Processor Interface," Fermilab Document FBN008.
6. R. Downing and H. Walz, "FASTBUS Snoop Diagnostic Module," IEEE Trans. Nucl. Sci. NS-28, No. 1 (February 1981), pp. 380-384.
7. D. Gustavson, T. Holmes, L. Paffrath and J. Steffani, "A 'Front Panel' Human Interface for FASTBUS," ibid, pp. 343-345.