SLAC-PUB-2738
May 1981
(I)

FASTBUS INTRODUCTION AND DEMONSTRATION*

C. A. Logg, L. Paffrath, B. Bertolucci, and D. Horelick
Stanford Linear Accelerator Center
Stanford University, Stanford, California 94305

This poster session paper presents a simplified explanation of the FASTBUS communication protocol and a brief description of the FASTBUS modules and the FASTBUS Operating System (FBOS) being used in the current prototyping efforts at SLAC. A sample session utilizing the FBOS to initialize and exercise a FASTBUS segment is also presented.

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 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 without handshake for transfer of data blocks at maximum speed.

This poster session paper presents a simplified explanation of the operation of the FASTBUS communication protocol. For a detailed description see Ref. 1. Also included is an overview of the FASTBUS Operating System (FBOS) and the three modules being used in the current prototyping efforts at SLAC.

FASTBUS Components

An example of a simple FASTBUS system is a single FASTBUS segment and a few modules. A 19 inch FASTBUS crate (Fig. 1), 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. The lowest 32 addresses allocated to any segment are used as the geographic addresses of the segment.

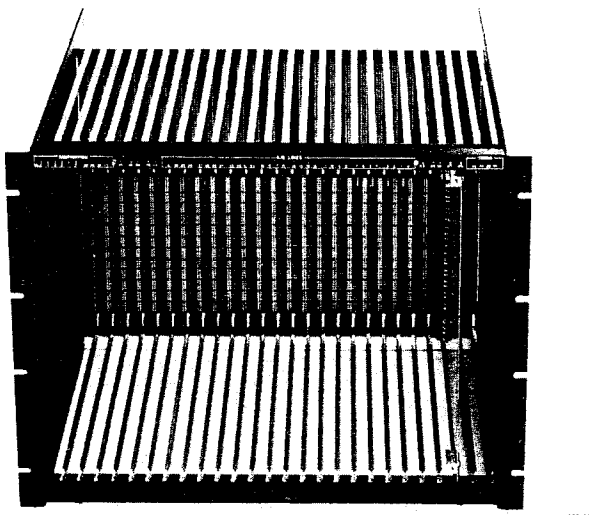

Fig. 1. A 19" FASTBUS Crate

There are two categories of FASTBUS modules: MASTERs and SLAVEs. A master module is one which can gain control (MASTERSHIP) of the segment and initiate operations on the 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 (Ref. 1) 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 geographic 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 geographic addressing can only select the module as a whole. Another advantage of logical addressing is that the module can be relocated within a crate (and possibly even the system) without any software changes in the masters (if the masters address modules by their logical addresses).

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

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 a master and slave(s).

Once a master has established the link, it can proceed to perform any data cycles necessary.

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 Arbitration Phase

The FASTBUS lines used in this discussion of Arbitration are:

AR - Arbitration Request
AG - Arbitration Grant
AL - Arbitration Level (6 lines)
GK - Grant Acknowledge.

The FASTBUS component used in the arbitration is the Arbitration Timing Controller (ATC).

Another FASTBUS line related to the arbitration, but ignored in this discussion, is the Arbitration Inhibit (AI) line.

The first step a master must perform when it is going to execute an operation on a FASTBUS segment is to arbitrate for mastership of the segment. The

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

(Contributed to the Topical Conference on Computerized Data Acquisition in Particle and Nuclear Physics, Oak Ridge, Tennessee, May 28-30, 1981.)

sequence (Fig. 2) for this is:

1. All masters, requiring masterhip on a segment, assert the AR line on the segment.
2. The ATC recognizes the arbitration requests and, at the appropriate time, generates AG which is an arbitration synchronization signal.
3. Each master which is arbitrating for FASTBUS mastership then asserts its arbitration vector on the 6 AL lines. A master's arbitration vector is its arbitration priority. 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.

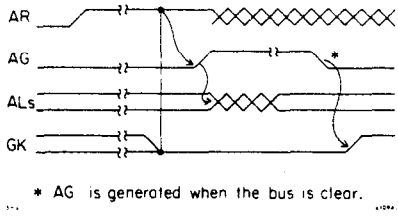

Fig. 2. Lines Asserted During Arbitration.

The Address Cycle Phase

The FASTBUS timing, control, and address/data lines used in this address cycle discussion are:

AS - Address Sync
AK - Address Acknowledge
CB - Control/(Block)
NH - No handshake
AD - Address/)Data) Lines (32 lines).

Other FASTBUS lines which are relevant to the address cycle but ignored in this discussion are:

EG - Enable Geographic; This line is generated by the EG Generator to indicate that an address is a geographic address.

NK - Negative Acknowledge, BK - Busy Acknowledge; These lines are used in multiple segment systems to return connection failure information to the master.

PE - Parity Enable, PA - Parity; These lines are used to indicate the parity of the data on the AD lines. Their use is optional.

After a master has obtained mastership, it can begin the address cycle.

Each module has two address spaces which are known as CONTROL space and DATA space. As a protection mechanism, the connection to these two address spaces is made via differing address cycles. The CB line is used to indicate which address space is being addressed. CB=1 indicates a control space address. CB=0 indicates a data space address. It is also possible to address more than one module at a time. This mode is known as MULTI-LISTENER (or BROADCAST) mode and is indicated by the NH line. NH=1 indicates multi-listener mode. NH=0 indicates SINGLE-LISTENER mode, the one that will be discussed here. The following table summarizes the CB and NH encodings during the address cycle:

CB	NH	ADDRESS CYCLE MEANING
0	0	single module data space address
1	0	single module control space address
0	1	broadcast to data space
1	1	broadcast to control space

The master initiates the address cycle (Fig. 3) by asserting the address of the slave on the AD lines, asserting the CB and NH lines, and finally asserting the AS line. The slave, upon recognizing its address, asserts the AK line to establish the AS-AK lock. Once the AS-AK lock is established, the master can proceed to the data cycle phase.

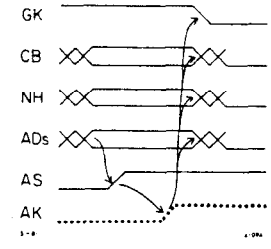

Fig. 3. Address Cycle.

The Data Cycle Phase

The FASTBUS Lines used in this data cycle discussion are:

DS - Data Sync
DK - Data Acknowledge
CB - (Control)/Block Transfer
NH - No Handshake
BK - Busy Acknowledge
NK - Negative Acknowledge
RD - Read
AD - (Address)/Data (32 lines)

Other FASTBUS lines used in the data cycle, but ignored in this discussion, are:
PE - Parity Enable, PA - Parity: The use of these lines is optional.

The data transfer during a data cycle can be either from the master to the slave (called a WRITE cycle), or, from a slave to a master (called a READ cycle). The direction of the transfer is controlled by the RD line. RD=1 for a read, RD=0 for a write.

There are 4 possible types of data cycles. These are called: a random data cycle, an extended address cycle, a handshake block transfer cycle, and a non-handshake block transfer cycle. The type of data cycle being performed is indicated by the CB and NH lines using the following encoding:

CB	NH	DATA CYCLE MEANING
0	0	random data cycle
1	0	handshake block transfer
0	1	extended address cycle
1	1	non-handshake block transfer

Once the AS-AK lock is established between a master and slave, any combination of data cycles (which a slave and master are equipped to handle) can be concatenated to perform the transfer of information between the modules.

A master initiates a data cycle by

1) asserting:
CB and NH to indicate the type of data cycle being performed,

RD to indicate the direction of 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 information transfer,
decoding the CB and NH 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 NK and BK lines.
3) The slave then completes the data cycle handshake by asserting DK.

An example of data cycle concatenation is shown in Figs. 4a and 4b. A normal address cycle, such as displayed in Fig. 3 must have been successfully completed. In Fig. 4a:
Section A shows a random data write cycle. The master asserts: CB=0, NH=0, the data on the AD lines, and then DS=1. The slave, after processing, returns DK=1. The master then drops DS, the slave drops DK, and the master then proceeds to the next cycle.
Section B shows an extended address write cycle. The extended address cycle is used to set a pointer internal to the slave module. This internal pointer, called the Internal Address (IA) is the location within a module where the next read or write data will come from or go to. It is often referred to as the next transfer address (NTA).

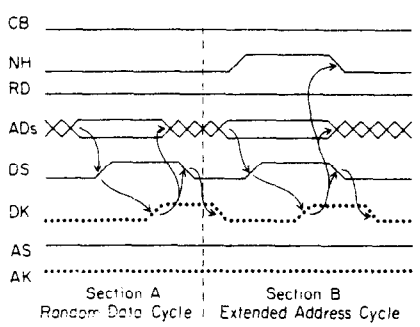

Fig. 4a. Examples of Write Cycles.

After the extended address cycle, which has set the slave's internal address pointer, the master could perform, for example, a handshake block transfer sequence of data cycles to read a block of data from a consecutively addressed area within the slave.

Figure 4b (Sects. C, D, and E) shows an example of a handshake block transfer read. Note that in the extended address and random data cycles, DS must be dropped at the completion of the data cycle. In a handshake block transfer data cycle (CB=1), every change in DS is used to indicate another data cycle.

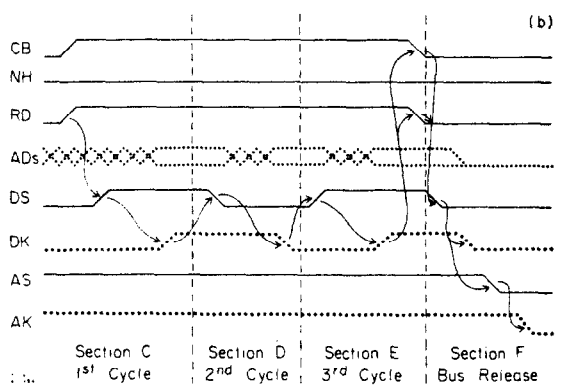

Fig. 4b. Cycles of a Handshake Block Transfer Read and Bus Release.

Bus Relase Phase

Once a master has obtained mastership it can perform any combination of address and data cycles needed to complete its operations as long as it retains the GK assertion. However, it is permissible to release the GK line as soon as the AS-AK lock is established if the master is not going to perform any more address cycles. The release of GK permits the next possible arbitration to occur. 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 (Sect. F of Fig. 4b).

Prototyping and Protocol Verification Efforts

Currently SLAC is performing some protocol verification tests and studying the implications of implementing the protocol. Three FASTBUS modules have been built for these tests. These are: a 256 32-bit word memory module (Ref. 2), a FASTBUS Sequencer (Ref. 3), and an I/O Register to FASTBUS Interface (IORFI) (Ref. 4). The IORFI is interfaced to an LSI-11 (/2 or /23) (Ref. 5) via 2 DRV11s (Ref. 6).

The FASTBUS Memory Module

The FASTBUS memory module has 256 words in data space and 4 CSR registers. CSR 0 is the ID register and bit 1 of CSR 0 is the logical address enable bit. 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. It is being used as a high speed slave module in the current tests.

The FASTBUS Sequencer

The FASTBUS sequencer is being used as a high speed master in the current tests. The sequencer has only control space addresses. The control space is divided into three sections. These are the status register memory, 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 cycle(s). The following functions can currently be encoded into an operation word to create an instruction for the sequencer:

Mnemonic	Function
AS	assert AS line
CB	assert CB line
NH	assert NH line
RD	assert RD line
DC	perform a data cycle (only extended address and random data cycles are currently implemented)
TERM	terminate sequencer operation and assert the Service Request (SR) line.

FASTBUS Operating System (FBOS)

SLAC FORTH 2.6 (Ref. 7) has been used to develop an operating system for FASTBUS prototype development and hardware checkout.

Currently the FBOS (Ref. 8) contains routines for performing various kinds of FASTBUS transfers, a Sequencer Program Assembler (Ref. 9), 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. That is, calling one word will perform the arbitration for mastership, address cycle, data cycle (or cycles in the case of a block transfer), and bus release. Some examples of words in the Complete FASTBUS Operations layer are:

DRD (DWR)
performs a data word read (write) from (to) data space
XDRD (XDWR)
performs an extended address cycle, followed by a data word read (write) from (to) data space
DBLKRD (DBLKWR)
performs a handshake block transfer of a block of data from (to) data space
XCRD (XCER)
performs a data word read (write) from (to) control space.

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 combination of FASTBUS cycle operations. They can also be called individually to single-step through a FASTBUS operation.

For example, ADD-CS, WR-EXT-ADD, and RD-CYCLE are words contained in the FCO layer.

XCRD is composed of calls to: ADD-CS which performs an address cycle to control space, WR-EXT-ADD which performs an extended address write cycle, and RD-CYCLE which performs a read data cycle.

For a complete description of FCO and CFO layer, as well as the other layers of the system, see Ref. 8.

The Sequencer Program Assembler

The sequencer program assembler encodes symbolic sequencer operations into a 32-bit instruction word and stores them in a program array. The AD line values are loaded in a data array. Figure 5 is a sample of a sequencer program.

```
0 DATA1 0 PGM1 STARTPGM   initialize the assembler (pass
                          it data and program arrays
                          which are to be used for the
                          assembly)
{AS DC #1000. AD}         address cycle instruction
{AS DC RD DMEM}           read instruction
{AS DC NH 1. AD}          extended address cycle
{AS DC RD DMEM}           read cycle
{ }                       empty instruction to drop AS
{TERM}                    terminate sequencer operation
ENDPGM                    indication to the assembler
                          that this is the program end
```

Fig. 5. Sequencer Program

FASTBUS Operation Monitoring Facilities

The FBOS contains several debugging facilities. The simpliest, and most widely used, is the FB command. This command reads, decodes, and prints symbolically on the terminal everything that can be read through the IORFI about the bus. FB is heavily used by the other debugging facilities.

The TRACE facility has the commands TRACE-ON and TRACE-OFF. The trace facility allows the user to examine (display on the terminal or lineprinter via the FB command) the FASTBUS bus status at each step in a FASTBUS operation. This can be used whether the IORFI is acting as a master or a slave.

The VERIFY facility has the commands VERIFY-ON and VERIFY-OFF. When the verify facility is enabled, each extended address write cycle and/or random data write cycle is followed by an extended address read cycle and/or random data read cycle, respectively. The readback is compared to what was written, and if they do not match, the message VERIFY FAILURE is printed on the terminal. This facility is useful for finding problems such as stuck bits. It must be used with some discretion since some verify failures will not necessarily indicate an error. For example, in the case of writing a bit to CSR 0, the readback will not match the written data, since the ID of the module will be returned in the read.

The DIAGNOSTIC facility is controlled by the commands DIAG-ON and DIAG-OFF. When the facility is enabled, a message is printed for each incomplete FASTBUS cycle. For example, if slot 5 is empty, and an address cycle to that slot is performed (i.e., 5. ADD-CS) the message NO AK RESPONSE will be printed on the terminal.

The IORFI has WT line (Ref. 1) generation logic which enables it to generate WT on any transition of AS, AK, DS, or DK. Routines written to facilitate using this feature include: TT-WT-ENB which enables the timing transition WT line generation, TT-WT-CLR which clears a timing transition WT condition and re-enables WT generation logic, and TT-WT-DIS which disables the WT generation completely. Thus, the sequencer can be loaded, started, and single-stepped through its sequence to the memory module. At each step, a complete display of the FASTBUS bus status (using FB.) can be seen.

Sample Session

This is a sample session wherein: the FASTBUS segment is initialized and mapped, the memory module is loaded and its logical addressing is enabled, the sequencer is initialized, loaded with a program and started, and single stepped through its operations. After the sequencer has finished, its contents are read out and the results are displayed on the terminal. The contents of the memory module are also read out (using a handshake block transfer) and displayed on the terminal.

1. Initialize the interface and issue a RESET BUS to the segment.

2. Map the segment:

SLOT	ID	MODULE TYPE
0	0000 0000	EMPTY SLOT
1	0000 0000	EMPTY SLOT
2	0000 0000	EMPTY SLOT
3	0000 0000	EMPTY SLOT
4	0001 0100	MEMORY MODULE
5	0000 0000	EMPTY SLOT
6	0000 0000	EMPTY SLOT
7	0000 0000	EMPTY SLOT
8	0000 0000	EMPTY SLOT
9	0000 0000	EMPTY SLOT
10	0000 0000	EMPTY SLOT
11	FFFF FF00	NON-STANDARD
12	0000 0000	EMPTY SLOT
13	0000 0000	EMPTY SLOT
14	0000 0000	EMPTY SLOT
15	0000 0000	EMPTY SLOT
16	0000 0000	EMPTY SLOT
17	0000 0000	EMPTY SLOT
18	0000 0000	EMPTY SLOT
19	0001 0200	SEQUENCER
20	0000 0000	EMPTY SLOT
21	0000 0000	EMPTY SLOT
22	0000 0000	EMPTY SLOT
23	0000 0000	EMPTY SLOT
24	0000 0000	EMPTY SLOT
25	0000 0000	EMPTY SLOT

3. Initialize the memory module and enable its logical addressing:

```
AS                  CB     GKI GKB     0000 0004
AS AK               CB     GKI GKB     0000 0004
AS AK DS DK     NH         GKI GKB     0000 0001
AS AK                      GKI GKB     0000 0000
AS AK DS DK                GKI GKB     0000 1000
AS AK                      GKI GKB     0000 0000
```

```
                                                0000 0000
AS                      CB      GKI GKB         0000 0004
AS AK                   CB      GKI GKB         0000 0004
AS AK DS DK         NH          GKI GKB         0000 0002
AS AK                           GKI GKB         0000 0000
AS AK DS DK                     GKI GKB         0000 0000
AS AK                           GKI GKB         0000 0000
                                                0000 0000
AS                      CB      GKI GKB         0000 0004
AS AK                   CB      GKI GKB         0000 0004
AS AK DS DK         NH          GKI GKB         0000 0003
AS AK                           GKI GKB         0000 0000
AS AK DS DK                     GKI GKB         0000 0000
AS AK                           GKI GKB         0000 0000
                                                0000 0000
AS                      CB      GKI GKB         0000 0004
AS AK                   CB      GKI GKB         0000 0004
AS AK DS DK         NH          GKI GKB         0000 0000
AS AK                           GKI GKB         0000 0000
AS AK DS DK                     GKI GKB         0000 0002
AS AK                           GKI GKB         0000 0000
                                                0000 0000
```

4. Load 10 zeros into the memory module data space:

```
AS AK                           GKI GKB         0000 1000
AS AK DS DK             CB      GKI GKB         0000 0000
AS AK                   CB      GKI GKB         0000 0000
AS AK DS DK             CB      GKI GKB         0000 0000
AS AK                   CB      GKI GKB         0000 0000
AS AK DS DK             CB      GKI GKB         0000 0000
AS AK                   CB      GKI GKB         0000 0000
AS AK DS DK             CB      GKI GKB         0000 0000
AS AK                   CB      GKI GKB         0000 0000
AS AK DS DK             CB      GKI GKB         0000 0000
AS AK                   CB      GKI GKB         0000 0000
                                                0000 0000
```

5. Initialize the sequencer:

```
AS                      CB      GKI GKB         0000 0013
AS AK                   CB      GKI GKB         0000 0013
                                                0000 0000
AS                      CB      GKI GKB         0000 0013
AS AK                   CB      GKI GKB         0000 0013
AS AK DS DK         NH          GKI GKB         0000 0001
AS AK                           GKI GKB         0000 0000
AS AK DS DK                     GKI GKB         0000 0000
AS AK                           GKI GKB         0000 0000
                                                0000 0000
AS                      CB      GKI GKB         0000 0013
AS AK                   CB      GKI GKB         0000 0013
AS AK DS DK         NH          GKI GKB         0000 0002
AS AK                           GKI GKB         0000 0000
AS AK DS DK                     GKI GKB         0000 0000
AS AK                           GKI GKB         0000 0000
                                                0000 0000
AS                      CB      GKI GKB         0000 0013
AS AK                   CB      GKI GKB         0000 0013
AS AK DS DK         NH          GKI GKB         0000 0003
AS AK                           GKI GKB         0000 0000
AS AK DS DK                     GKI GKB         0000 0013
AS AK                           GKI GKB         0000 0000
                                                0000 0000
```

6. Load the sequencer program:

```
AS MEM (0) = 0000 1001
AS DC MEM (1) = F1F1 F1F1
AS DC NH MEM (2) = 0000 0002
AS DC MEM (3) = F2F2 F2F2
AS DC RD MEM (4) = 8181 8181
AS DC NH MEM (5) = 0000 0001
AS DC RD MEM (6) = 8181 8181
EMPTY INSTRUCTION
TERM
```

7. Start the sequencer:

```
AS                      CB      GKI GKB         0000 0013
AS AK                   CB      GKI GKB         0000 0013
AS AK DS DK         NH          GKI GKB         0000 0000
AS AK                           GKI GKB         0000 0000
AS AK DS DK                 AR  GKI GKB         0000 0008
AS AK                       AR  GKI GKB         0000 0000
                            AR  GKI GKB         0000 0000
```

8. Sequencer in operation (single stepping):

```
AS                          WTB GKB WTT         0000 1001
AS AK                       WTB GKB WTT         F1F1 F1F1
AS AK DS                    WTB GKB WTT         F1F1 F1F1
AS AK DS DK                 WTB GKB WTT         F1F1 F1F1
AS AK    DK                 WTB GKB WTT         F1F1 F1F1
AS AK              NH       WTB GKB WTT         0000 0002
AS AK DS           NH       WTB GKB WTT         0000 0002
AS AK DS DK        NH       WTB GKB WTT         0000 0002
AS AK    DK        NH       WTB GKB WTT         0000 0002
AS AK                       WTB GKB WTT         F2F2 F2F2
AS AK DS                    WTB GKB WTT         F2F2 F2F2
AS AK DS DK                 WTB GKB WTT         F2F2 F2F2
AS AK    DK                 WTB GKB WTT         F2F2 F2F2
AS AK         RD            WTB GKB WTT         0000 0000
AS AK DS      RD            WTB GKB WTT         F2F2 F2F2
AS AK DS DK RD              WTB GKB WTT         F2F2 F2F2
AS AK    DK RD              WTB GKB WTT         F2F2 F2F2
AS AK              NH       WTB GKB WTT         0000 0001
AS AK DS           NH       WTB GKB WTT         0000 0001
AS AK DS DK        NH       WTB GKB WTT         0000 0001
AS AK    DK        NH       WTB GKB WTT         0000 0001
AS AK         RD            WTB GKB WTT         F2F2 F2F2
AS AK DS      RD            WTB GKB WTT         F1F1 F1F1
AS AK DS DK RD              WTB GKB WTT         F1F1 F1F1
AS AK    DK RD              WTB GKB WTT         F1F1 F1F1
AS AK                       WTB GKB WTT         0000 1001
```

9. The sequence now contains:

```
AS MEM (0) = 0000 1001
AS DC MEM (1) = F1F1 F1F1
AS DC NH MEM (2) = 0000 0002
AS DC MEM (3) = F2F2 F2F2
AS DC RD MEM (4) = F2F2 F2F2
AS DC NH MEM (5) = 0000 0001
AS DC RD MEM (6) = F1F1 F1F1
EMPTY INSTRUCTION
TERM
```

10. Read the memory module memory:

```
AS AK                       SR      GKI GKB     0000 1000
AS AK DS DK RD CB           SR      GKI GKB     0000 0000
AS AK         RD CB         SR      GKI GKB     F1F1 F1F1
AS AK DS DK RD CB           SR      GKI GKB     F2F2 F2F2
AS AK         RD CB         SR      GKI GKB     0000 0000
AS AK DS DK RD CB           SR      GKI GKB     0000 0000
AS AK         RD CB         SR      GKI GKB     0000 0000
AS AK DS DK RD CB           SR      GKI GKB     0000 0000
AS AK         RD CB         SR      GKI GKB     0000 0000
AS AK DS DK RD CB           SR      GKI GKB     0000 0000
AS AK         RD CB         SR      GKI GKB     0000 0000
                            SR                  0000 0000
```

11. List of the memory module contents:

0	0000 0000
1	F1F1 F1F1
2	F2F2 F2F2
3	0000 0000
4	0000 0000
5	0000 0000
6	0000 0000
7	0000 0000
8	0000 0000
9	0000 0000

Summary

A brief explanation of the FASTBUS protocol has been given along with a short description of the 3 FASTBUS modules and operating system which are being used in the initial prototyping and verification efforts.

REFERENCES

1. FASTBUS Modular High Speed Data Acquisition System for High Energy Physics and other Applications, U.S. NIM Committee.
2. Memory Module - Specifications, Boris Bertolucci, 8/1/80, SLAC.
3. FASTBUS Sequencer - Interim Description, Dale Horelick, 10/10/80, SLAC.
4. I/O Register to FASTBUS Interface, Connie Logg and Leo Paffrath, (SLAC EIN Note), March 1981.
5. Microcomputers and Memories, Digital Equipment Corporation, 1981.
6. Microcomputer Interfaces Handbook, Digital Equipment Corporation, 1980.
7. SLAC FORTH Programmer's Guide, M. Stoddard, J. Kieffer, and S. Deiss (SLAC EIN software note), Aug. 1980.
8. Sequencer Program Assembler and Driver Routines, C. Logg (SLAC EIN software note), March 1980.
9. Software for the FASTBUS Prototype Development, C. Logg (SLAC EIN software note), March 1981.