SLAC-PUB-2994
October 1982
(M)

A FASTBUS CONTROLLER MODULE USING A MULTIBUS MPU*

S. R. Deiss
Stanford Linear Accelerator Center
Stanford University, Stanford, California 94305

Abstract

The architecture, adaptability and performance of SLAC's single board controller module will be detailed. Example uses with an 8086 MPU and with a 68000 MPU will be given. Details of circuit design and software interface will be provided.

Introduction

In FASTBUS systems there are many tasks that could benefit from the availability of a single card microcomputer. All experiments involve a large element of slow-real-time equipment monitoring and control tasks to be contrasted with high speed data acquisition. Such tasks are ideal for an MPU. A single card MPU has the added advantage of being totally self contained with no need for power or chassis mounting beyond that provided by the FASTBUS segment itself.

At SLAC we have developed a working prototype of this kind of device called the SLAC FASTBUS CONTROLLER, or simply SFC (Fig. 1). The SFC is a 'universal MPU controller' in the sense that it is designed to accept any IEEE 796 (MULTIBUS) MPU card into onboard P1 and P2 connectors.[3] Therefore, any microprocessor which has a single card computer implementation on MULTIBUS can be plugged into the SFC to make a FASTBUS controller.[2] This list includes Motorola's 68000, Intel's 8086, and National's 16032 as well as several 8 bit microprocessors. The SFC board provides all functions of a MULTIBUS motherboard. To the 796 BUS MPU the SFC interface logic responds as a standard IEEE 796 I/O slave (D16 I16 VO L).

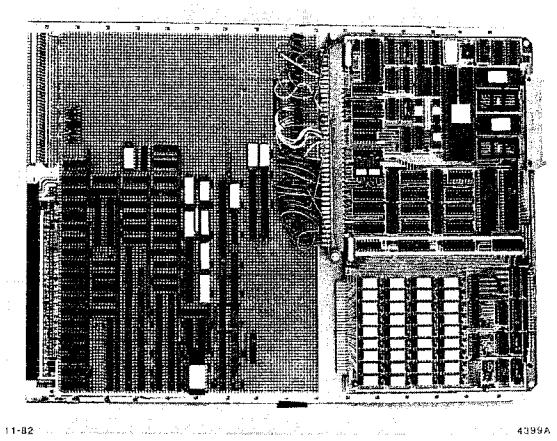

Fig. 1. SFC Wirewrap Prototype with 68000.

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

Furthermore, the 796 bus signals can be routed via the FASTBUS auxilliary connector to an adjacent FASTBUS slot in order to allow use of some of the many MULTIBUS peripheral controller cards. One must only ensure that maximum MULTIBUS length is not exceeded.

FASTBUS Master Capability

The SFC performs FASTBUS address or data cycles on command from the MPU. All defined FASTBUS operating modes are supported. The MPU expresses its command as a read or write to MULTIBUS I/O space. 256 bytes of I/O space are required for the SFC. The FASTBUS command is memory mapped within this 8 bit space. That is, the 8 bit I/O address is decoded into AS, DS, MS0, and MS1. The MULTIBUS data bus simultaneously carries 8/16 bits of FASTBUS AD lines. Therefore, in two MULTIBUS cycles a 16 bit MPU can completely read or write the 32 FASTBUS AD lines plus specify any type of FASTBUS address or data cycle to be performed. MPU's like the 68000 and the 16032 can move a 32 bit longword with one MPU instruction. Thus with one instruction the MPU can perform a complete 'primitive' cycle on the FASTBUS.

The SFC does away with much of the software overhead usually associated with performing a FASTBUS cycle under MPU software control. MULTIBUS and FASTBUS are both asynchronous. Hence, it is natural for the MPU to wait for the FASTBUS AK or DK acknowledge signal to return from the slave or ancilliary logic before it is given the MULTIBUS XACK* signal to terminate the MULTIBUS cycle. The MPU does not have to assert the strobe, move the data, and then read, test and branch to see if the operation was successful. This is done automatically by the SFC for the MPU.

The MPU supplies the command and the data for the FASTBUS cycle. The SFC issues the strobes, waits for acknowledges, manages a timeout counter, checks for parity errors and non-zero slave SS response codes, etc. while the MPU waits for the XACK*. When the FASTBUS cycle finishes or times out the SFC sends back the XACK*. The XACK* is accompanied by a bus error (BERR) interrupt if anything unusual happened that requires MPU intervention (parity error, SS non-zero, timeout). Such events are relatively rare. By handling their occurence with an interrupt instead of a status check after each operation, the MPU's FASTBUS utilization increases dramatically while the code efficiency in terms of FASTBUS operations per byte of code also improves.

As an example implementation, the SFC interface logic was tested using an early prototype of the SUN 68000 from STANFORD University.[5] This is an 8MHz MPU with 256K of parity RAM, 32K ROM, 2 level memory protection and address translation unit, serial I/O, parallel input port, 5 timers and the MULTIBUS interface. Each of its MULTIBUS cycles included about 300 ns of overhead for address translation and

(Presented at the Nuclear Science Symposium, Washington, D.C., October 20-22, 1982.)

MULTIBUS access plus 300 ns more of SFC address decoding and handshaking between MULTIBUS and FASTBUS plus slave response time (approx. 50ns). In this case the SUN MPU could do a FASTBUS address or data cycle in 2 μs if using registers, or in 3-5 μs if using on board RAM. In some modes data transfer burst rates approaching 1MHz are possible. However, 5 μs per 32 bit data transfer is a good number to use for comparisons which includes some time for loop overhead, error recovery overhead, and operand effective address calculation.

The SFC has begun undergoing tests with an INTEL iSBC 86/12A (8086) MPU board.[4] Detailed timing can not be given as yet. However, the performance seems to be on the same order of magnitude as the SUN 68000 processor board described above. There is some speed loss due to 8086 instruction set and register architecture. The 86/12A will be combined with the SFC to provide a single board backup computer for the liquid argon system in the MARK II detector at SLAC. The software development will include an implementation of the FASTBUS Standard Subroutine Package for use by the application software.[8]

Following is a sample of the assembly coding that would be required for a 68000 to do the following: arbitrate, check for mastership, address cycle, read-modify-write, drop AS, drop GK.

```
;LOAD BASE ADDRESS OF SFC
      MOVE.L #$1F0000,A0
;SET ARB. REQ. BIT IN CTL. REG.
 BEG  BSET   7,7(A0)
;SEE IF BUS MINE IN MASTER STS. REG.
 CHEK_BTST   5,4(A0)
;LOOP HERE UNTIL BUS MINE
      BEQ.S  CHEK
;ADDRESS SLOT $11 SLAVE
      MOVE.L #$11,$E0(A0)
;READ 1 WORD FROM DATA SPACE
      MOVE.L $F0(A0),D0
;DOUBLE THE WORD
      ADD.L  D0,D0
;WRITE THE WORD BACK TO THE SLAVE
      MOVE.L D0,$F0(A0)
;TAKE AS DOWN
      MOVE.B #$0,$40(A0)
;TAKE GK DOWN
 END  BCLR   7,7(A0)
```

The I/O address to FASTBUS command map is given in detail elsewhere along with explanation of all control and status register bits.[1] But as can be seen everything is done as an address displacement off of the SFC base address. In a real programming environment these address constants would be replaced by predefined mnemonic assembler constants. All of the above operations might be redefined as macros of one or two instructions. For example, the three instructions starting at 'BEG' might make a macro called 'GETBUS' while the last instruction at END might be a macro called 'DROPBUS'. If that were done, the above code would read as follows in macroese:

```
      START
BEG   GETBUS
      GEOADR $11
      READDATA D0
      ADD.L D0 D0
      WRITEDATA D0
      ASDOWN
END   DROPBUS
```

This entire BEG..END sequence takes place in approximately 19 μs as long as no BERR interrupts occur (Fig. 2). The status register used for error recovery is arranged for a mask-shift-indexed branch through a table of error routine entry points. Thus, recovery should be quick and table management overhead should be low.

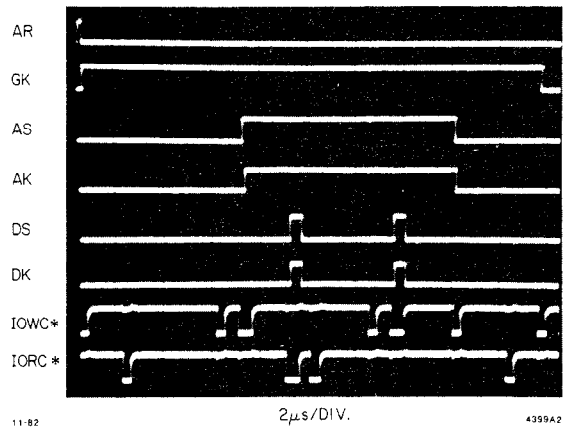

Fig. 2. Random Read-Modify-Write with Arbitration.

The SFC has arbitration inhibit logic which can be activated with a jumper. It also has a special 'SCRAM' option which if set causes the SFC to drop GK and AS/AK lock in the event of any error condition. This was done so that one would have a simple failsafe mechanism that ensures that the SFC can not tie up a segment in some loop involving an error. In addition to the previously mentioned BERR interrupt there is a general interrupt (GINTR) which is the 'or' of incoming SR, taking mastership, and being selected as a slave. All three interrupt sources are enabled as one with one control bit.

It has been seen how one can move 32 bits of FASTBUS AD lines plus specify a FASTBUS command with one MPU instruction. This results in at least two MULTIBUS cycles not counting those for fetching of the instruction and the fetching or storing of the data. The above mode is called the interlocked CYCLE mode. The SFC issues the FASTBUS strobes after all outgoing write data has been received from MULTIBUS or, alternatively, on the first MULTIBUS cycle involving a FASTBUS read. It will also work with 8 bit MPUs that only move a byte at a time. It just takes twice as many MULTIBUS cycles in that case. The SFC can be jumpered to expect either LSB's first or MSB's first in back-to-back cycles that move a 32 bit longword. Thus, it will work with either a 68000 or a 16032 type word order.[6]

There is also a COMMAND mode in which no data is moved on the MULTIBUS data lines. The I/O address bits contain the command as usual, but there is no new data. This feature can be used to clear AS or to take DS down at the end of a block transfer with an odd number of words. Or it can be used to write a repeated data word. One loads the AD register with a word, and then issue DS write commands. That way there is no need to move 32 bits over MULTIBUS for each FASTBUS data cycle. In this mode the MPU still waits for the AK/DK response from the bus.

Finally, there is an OVERLAPPED command mode similar to COMMAND mode except that the SFC DOES NOT wait for the slave's acknowledge before sending the XACK* back to the MPU. This allows pipelining between the MPU and the SFC . The next word in a block can be fetched while the last is being moved on FASTBUS. Or,, something useful could be done while waiting for a slow module or while waiting for an address connection involving multiple segments and SI's. For diagnostics one could do an overlapped command as master involving the SFC itself as the slave. Then one has control over slave SS response as well as slave response latency.

FASTBUS Slave Capability

The SFC defines its slave characteristics through software emulation. Once selected it enters a software state machine loop: 1) wait for command ready, 2) read command bits (RD, MS), 3) mask-shift-indexed branch to the appropriate command handler, 4) make DK, SS response, and 5) go back to step 1 until deselected.

The SFC hardware provides support for address recognition, but after that the slave is under software control. The SFC will respond to geographic addressing, general broadcasts, pattern selects, sparse data scan, SR scan, and logical addressing. Wait is generated automatically upon detection of a logical address within range. The slave software must further examine the IA to make sure it is valid and then make a pseudo DK response to set the SS code, drop WT, and let the AK go on. The IA can be 8,13,18,23,28, or 32 bits wide. After being selected the SFC generates WT on DS up (MS0=0) or DSt (MS0=1). Also it toggles DK correctly during block transfers so that the slave software does not have to monitor DK. Thus, all the slave has to do is issue the proper SS response, update its internal state, and supply read data, if any.

During broadcasts WT is generated for DS, but the software DK command does not result in a DK. It only clears the WT and optionally sets SS. There is an AUTOMATIC SLAVE option which if set inhibits normal WT generation and produces DK automatically. This is useful for self diagnostics and for stand alone demonstrations.

The slave only responds with interlocked CYCLES or with a COMMAND. There are no OVERLAPPED slave commands. There can be no BERR during a slave response. In defining the slave device to be emulated the user does have to consider that some CSR's in FASTBUS control space require hardware supports not provided by the SFC thus making them impractical to simulate.

FASTBUS Host Capability

With the SFC the programmer can assert GK and RB in order to preempt a segment and take it over as a HOST. The only CSR's supported in hardware are the logical address register and the arbitration level register, and both of these can be accessed and loaded by the SFC.

Therefore, if it were desirable from a systems point of view, one could use an SFC together with a powerful 16 bit MPU as the FASTBUS HOST. This processor would have responsibility for maintaining the data base used in managing a multi-segment system as well as for initialization and diagnostics.[7] By splitting out the HOST function from the main data aquisition computer one can arrange a minimal backup that assures equipment integrity during downtime.

Self Diagnostic Capability

The SFC can address itself geographically, logically, or in a broadcast. The automatic WT generation controls are turned off in all these cases except for logical addressing. Thus the software can test all of the master handshake logic, the slave handshake logic, all address recognition logic, wait generation logic, and timeout counter operation. Typically the diagnostic software would put the SFC in auto-slave mode and then perform read and write CYCLE's to itself. To test WT generation the software would logically address itself and use OVERLAPPED commands so as not to hang up when WT is asserted. The same approach works for testing proper operation of the timeout counter.

Logic Design

The SFC uses an entirely asynchronous design to translate MULTIBUS handshakes into FASTBUS and back. The interface hardware is about 1/3 TTL and 2/3 ECL or level translators. The bulk of the logic is contained in PAL devices programmed with a high level PAL equation assembler. The balance of the logic consists of registers, drivers, and receivers, with a small amount of random logic for arbitration and signal conditioning.

Future Plans

A PC version of the SFC is in preparation. Every effort will be made to produce a single width module. This presents some problems due to the width of the MULTIBUS connector. Customized female P1 and P2 connectors will have to be made a part of the PC implementation.

Another feature of the PC version will be a small wire wrap area on board where the user will be able to kludge in small amounts of customization logic. One of the first uses for this area will likely be to add a high speed sequencer for the SFC. Detailed timing analysis shows that if driven from a high speed RAM instead of through MULTIBUS with all the attendant address decoding and handshaking, the SFC may be capable of a speed/throughput increase of nearly an order of magnitude for such sequences. The PC version will have a local 32 bit TTL bus plus jumper posts to wrap to in order to simplify adding such a sequencer later.

Acknowledgements

The author wishes to thank B. Bertolucci, D. Gustavson, R. Larsen, and H. Walz for assistance in interpreting the FASTBUS specification.

References

1. 'SLAC FASTBUS Controller - Summary Spec', S. Deiss, SLAC, 8/5/82.

2. 'FASTBUS....Tentative Specification', U.S. NIM Committee, 6/7/82.

3. ´Proposed Microcomputer System Bus Standard (P796 Bus)´, IEEE Computer Society, October 1980 with December 1980 errata.

4. ´iSBC 86/12A Single Board Computer Hardware Reference Manual´, INTEL, 1979.

5. ´The SUN Workstation´, Bechtolsheim and Baskett, Stanford U., C.S. Dept., 4/30/81.

6. ´DB16000 Development Board User´s Manual´, National Semiconductor, 1982.

7. ´Software for Managing Multicrate Fastbus Systems´, Deiss and Gustavson, SLAC, NSS, 1982.

8. ´Standard Routines for FASTBUS´, FASTBUS Software Working Group, NSS, 1982.