LEGIBILITY NOTICE

A major purpose of the Technical Information Center is to provide the broadest dissemination possible of information contained in DOE's Research and Development Reports to business, industry, the academic community, and federal, state and local governments.

Although a small portion of this report is not reproducible, it is being made available to expedite the availability of information on the research discussed herein.

LA-UR -87-1541

[illegible]

Los Alamos National Laboratory is operated by the University of California for the United States Department of Energy under contract W-7405-ENG-36.

TITLE: AN IMPLEMENTATION OF THE NEW IEEE STANDARD ROUTINES FOR FASTBUS

LA-UR--87-1541

DE87 010107

AUTHOR(S): Thomas Kozlowski and Will M. Foreman

SUBMITTED TO Fifth Conference on Real-Time Computer Applications in Nuclear, Particle and Plasma Physics
May 12-15, 1987
San Francisco, CA

DISCLAIMER

Los Alamos National Laboratory
Los Alamos, New Mexico 87545

[illegible]

[illegible]

AN IMPLEMENTATION OF THE NEW IEEE STANDARD ROUTINES FOR FASTBUS

Thomas Kozlowski and Will M. Foreman
Los Alamos National Laboratory*
Los Alamos, NM 87545
Los Alamos, N.M. 87545

We have implemented a subset of the new 1987 IEEE standard routines for FASTBUS for the General Purpose Master (GPM) [1], a FASTBUS master developed at CERN. Experiences in implementing and using the new standard routines for FASTBUS are reported.

1987 Standard Routines for FASTBUS

Standards for routines for FASTBUS were first published in 1983 [2]. Over several years of experience in developing FASTBUS software and in using the 1983 routines some deficiencies were apparent in this preliminary standard. The deficiencies include problems in the use of the routines in a multiple user environment, and too much implementation level detail defined in the standard.

The process to develop a new improved standard for routines for FASTBUS was begun in 1986. An initial proposal was made in 1986, the so-called "revisionist" routines [3]. This revised standard removed many of the previous deficiencies and added some new useful features such as automatic error reporting. It was oriented towards list processing, and it still included some details perhaps more properly left to the implementor. The "revisionist" routines formed a basis for the standard routines finally adopted in 1987 [4].

The 1987 standard has a minimum of implementation level details. Most data structures are hidden from the user of the routines in an "environment", referenced by an "ID" argument present in most standard routine calls. Environments plus port allocation and deallocation routines (the latter tied to a "process") provide good support for multiple user implementations. All parameter names and status returns are defined symbolically, actual numeric codes being left to the implementor. Provision is made in many cases for implementation specific extensions (for example, implementation specific operational parameters). Some features directly relevant to implementing the routines are discussed in more detail in a subsequent section.

The General Purpose Master

Architecture

The GPM was designed and prototypes built at CERN. It is now available commercially from CES [5] and Struck [6]. It is general purpose in the sense that it can be programmed in high level languages, can act as a FASTBUS master or slave, and has ports that can be connected to a host processor.

Supported by U. S. Dept. of Energy Contract W-7405-ENG-36.

The GPM is based upon a Motorola 68000 (M68000) series processor, and resides on a single FASTBUS board. It supports 256K bytes to 512K bytes of RAM and a large EPROM space. Direct FASTBUS access is available to from 128K bytes to 256K bytes of fast two-port memory. There are two RS232 ports (up to 19200 baud) that can be connected to a terminal or host processor terminal ports, and a parallel port that can be used as a high speed connection to another processor or device.

Software access to FB is through special addresses in the M68000's address space. For example, a primary address cycle may be done by a single instruction of the form

```
MOVE.L PRIM_ADDR,DGEOG
```

where "PRIM_ADDR" contains the value of the primary address and "DGEOG" indicates a FASTBUS primary address cycle to data space with the master asserting EG. Hardware option bits in control registers allow optional selection of pipe-lined block transfers, holding AS/AK at the end of a transaction, assertion of EG, etc. Error conditions are flagged by interrupts; for example, there are 16 interrupt vectors for the 16 possible address cycle and data cycle SS codes.

Applications

Our present use of the GPM is for a particular experiment at the Los Alamos Meson Physics Facility (LAMPF) [7]. In the future it may be supported for general use at LAMPF as a FASTBUS master and host interface. In our particular application the GPM serves as both an intelligent master in the FASTBUS and as host interface to a MicroVAX II

In its application as an intelligent FASTBUS master the GPM will move data from frontend FASTBUS slaves to a microprocessor farm [7] via another FASTBUS slave. It will also be used for running high performance diagnostics in the FASTBUS network. In these cases the software executes in the GPM.

In our use of the GPM as a host interface it is connected to a MicroVAX-II via RS232 lines. In our particular application the speed of this connection is adequate (approximately 2K bytes/second). The GPM itself can also serve as a FASTBUS host processor with a terminal connected to one of the RS232 ports. A user is able to write and run M68000 resident code using high level languages (FORTRAN, PASCAL) with support of the MONICA monitor and debugger [8].

A point relevant to our implementation is that our particular application does not require use of a FASTBUS interrupt mechanism (FASTBUS Service Request the GPM does not support standard FASTBUS Interrupt Messages), and therefore the initial implementation of the standard routines does not include any of the interrupt handling routines. SR handling will be added to the implementation at a later time.

Implementation

Important Features of 1987 Routines

Some features of the standard routines have a direct impact on the implementation. These include environments, definition of a "process", error handling, and items mandated by the standard.

Environment and data buffers: The environment incorporates the current values of operational parameters that affect FASTBUS operations, status history information and pointers for sequential buffers. The internal structure and implementation is completely left to the implementor. Thus it is recommended that the environment be referenced only through the standard routines. The implementation must provide a way for creating and deleting independent environments in the case of multiple users. If the optional delayed execution mode environment ("list processing") is supported, the implementation can be significantly more complex.

The standard defines a mandatory default environment ID (FB_DEFAULT_EID) that corresponds to an environment that is always available to a user/process (no creation necessary). In the multiple user case the implementor must take care that the default environment for different users/processes are independent.

The implementation must take into account which of the buffering schemes offered by the standard are to be supported. Sequential buffers (data associated with multiple transactions is transfered to/from the same buffer in sequence) require an underlying structure of pointers and other accounting information, which is associated with an environment. The definition of an "external" sequential buffer (a data storage device for example) is left entirely to the implementor.

Processes: The standard routines document uses the terms "process" and "user" interchangeably. A functional definition for "process" in the context of the routines is the code executed between the calls to the FB_OPEN routine and the FB_CLOSE routine. Most multiple user computing environments (operating systems) incorporate the concept of a process which is compatible with this definition. It should be noted that a single process may create several simultaneous environments, and a separate default environment should exist for every separate process. The allocation or deallocation of a port is done for a particular process (not for an environment).

Error handling: The level of automatic error handling that is chosen will influence the complexity of the implementation. The standard defines a range of levels for implementing error reporting The mandatory lowest level includes only the routine FB_STATUS_GET_SUMMARY which permits retrieval of the status return from the last action routine called along with the byte counts for FASTBUS transfers. Higher levels permit a rather elaborate error reporter with many reporting options.

Mandatory routines and features: The standard mandates a subset of the routines and certain features. These are: the routines for simple transactions (single word and block transfers for a specified primary and secondary addresses, etc.), support of immediate execution mode environments, a default environment, setting and retrieving operational parameters, the simple array form of data buffer and sequential buffers if delayed execution mode is supported, those primitive cycle routines supported by the interface hardware, those interrupt and SR service routines supported by the interface hardware, and a minimum of status reporting.

Implementation on the GPM

For our application the GPM serves both as a master resident in FASTBUS and as a host interface. Therefore routines are needed to run both on the GPM M68000 and on the host processor, a MicroVAX II. The GPM resident routines are a basis for the VAX resident routines.

The GPM is delivered with the CERN MONICA debugger-monitor in EPROM. MONICA provides, besides a symbolic debugger, a simple operating system with support for a loader, RS232 support, stack and heap management, etc. This underlying software reduced the effort required in developing GPM software. Code development was done using CERN cross-software tools for the Motorola 68000 that run under VAX VMS [9]. These include FORTRAN, PASCAL, an assembler, an include file utility, a linker, and loader.

Routines implemented: The following routines and features were implemented. In actuality only the short (6 character) form of the names is used in the GPM implementation because of limitations of the cross-software compilers.

environment management:

- FB_OPEN, FB_CLOSE
- FB_CREATE_IMMEDIATE_ENVIRONMENT,
  FB_RELEASE_ENVIRONMENT,
  FB_RESET_ENVIRONMENT

operational parameter management:

- FB_PAR_INIT, FB_PAR_SET, FB_PAR_GET

data buffers:

- FB_DECLARE_SEQ_BUFFER, FB_RELEASE_SEQ_BUFFER
- supported modes:
  FB_BUFFER_VAR, FB_BUFFER_VALUE, FB_BUFFER_SEQ

simple transactions:

- FB_READ_DAT, FB_WRITE_DAT,
  FB_READ_CSR, FB_WRITE_CSR,
  FB_READ_DAT_MULT, FB_WRITE_DAT_MULT,
  FB_READ_CSR_BLOCK_MULT, FB_WRITE_CSR_BLOCK_MULT

- FB_READ_DAT_BLOCK, FB_WRITE_DAT_BLOCK,
  FB_READ_CSR_BLOCK, FB_WRITE_CSR_BLOCK,
  FB_READ_DAT_BLOCK_MULT, FB_WRITE_DAT_BLOCK_MULT,
  FB_READ_CSR_BLOCK_MULT, FB_WRITE_CSR_BLOCK_MULT

- FB_READ_DAT_SA, FB_WRITE_DAT_SA,
  FB_READ_CSR_SA, FB_WRITE_CSR_SA

- FB_READ_LENGTH

route tables:

- FB_READ_ROUTE_TABLE,
  FB_WRITE_ROUTE_TABLE

- FB_READ_ROUTE_TABLE_BLOCK,
  FB_WRITE_ROUTE_TABLE_BLOCK

primitive actions and single lines:

- FB_CYCLE_ARBITRATE

- FB_CYCLE_RELEASE_BUS

- FB_CYCLE_PA_DAT, FB_CYCLE_PA_CSR
  FB_CYCLE_PA_DAT_MULT, FB_CYCLE_PA_CSR_MULT

- FB_CYCLE_DISCONNECT

- FB_CYCLE_READ_WORD, FB_CYCLE_WRITE_WORD,
  FB_CYCLE_READ_SA, FB_CYCLE_WRITE_SA

- FB_CYCLE_READ_BLOCK, FB_CYCLE_WRITE_BLOCK

- FB_LINE_READ, FB_LINE_WRITE

port allocation/deallocation and reset:

- FB_PORT_RESET

- FB_PORT_ALLOCATE, FB_PORT_DEALLOCATE

error reporting:

- o FB_STATUS_GET_SUMMARY
- o FB_STATUS_SEVERITY, FB_STATUS_MATCH, FB_STATUS_THRESHOLD
- o FB_STATUS_TRANSLATE

Environment and data buffers: Only immediate execution mode environments are supported by the GPM resident routines. This is adequate for most applications because of the efficient architecture of the interface to FASTBUS. When a new environment is created a fixed amount of space is allocated from the MONICA heap. Each environment includes an array of all current values all writable operational parameters. In addition it includes space for storing the summary status and the (future) supplementary status history, and for support of four sequential buffers (pointers). At execution time every routine that references the environment, checks if the environment ID has changed since the last such call; if it has, a new environment is created. Changing the environment involves some hardware operations (setting and clearing bits in GPM control registers). Setting the GPM's arbitration level (an operational parameter) can be problematical since the only access is through a FASTBUS transaction. The software simply trys to do the FASTBUS transaction.

The GPM process: All standard routine calls which do not directly originate from a standard routine call on the VAX (see the next section) are considered part of a single process on the GPM. In other words, the application resident in the GPM is a single process. This definition was chosen as the simplest, since MONICA is a single task monitor. Thus the GPM resident application can allocate the FASTBUS port, locking out all VAX resident applications using the FASTBUS routines.

Error handling: Because of the limited resources of the GPM microprocessor, only a simple error reporting mechanism has been implemented. All GPM FASTBUS errors generate interrupts. Simple interrupt service routines flag any FASTBUS error interrupt by setting a bit in a common area. This common area is then examined by higher level software if any error condition occurred, to generate the appropriate standard status return code. This code and the byte count that results from each action routine call is stored in the summary status history for the environment. The status information is available by a call to FB_GET_SUMMARY. Because the GPM is used in a VAX environment VAX format error codes were used. The VAX severity levels are the same as the severity levels defined in the standard (SUCCESS, INFORMATION, WARNING, ERROR and FATAL). The 1983 standard defined specific numerical values to be used for error codes.

GPM specific features of routines: Certain hardware features of the GPM necessitated adding some GPM specific features to the implementation of the routines. These additions were easily accommodated by the standard. The major additions were:

- The GPM timers do not map directly onto the timers defined in the hardware standard; there is one short timer used for most FASTBUS cycles and a long timer used when WAIT is asserted. They are both software settable. Operational parameters were added that correspond to these timers (FB_PAR_GPM_LONG_TIMER and FB_PAR_GPM_SHORT_TIMER). The operational parameters corresponding to the standard FASTBUS AK, DK, WAIT, and LONG timers were made "not-supported".

- A GPM option allows overlapping a FASTBUS block transfer read cycle with the memory store of the previous data word. Another operational parameter was added to allow enabling or disabling this option (FB_PAR_ENABLE_OVERLAPPED).

- To avoid interference of users with each other, port allocation is required for certain operations that leave the FASTBUS in a state such that another complete transactiion can not be carried out by another user (all primitive routines, setting the FB_PAR_HOLD_AS operational parameter, etc.). If a user attempts to do one of these operations without having allocated the port, a "port not allocated" status is returned.

- Some error return codes were defined that are specific to the GPM. These implementation specific error returns would best be reported as supplementary status, reporting a standard return code as summary status, e.g., FB_ERR_HARDWARE. Since our simple implementation does not support supplementary status, they are returned as summary status.

VMS implementation for GPM as host interface

The initial implementation of the standard routines on the VAX is simple. It is expected that it will become more elaborate in the future. The present implementation allows multiple users but each must allocate the VAX terminal device connected to the GPM's RS232 line when accessing FASTBUS. Only the routines that are included in the implementation of the routines resident in the GPM are implemented for the VAX. At present no sequential buffers are allowed for the VAX resident routines. Many of these features have been chosen to allow straightforward use of remote procedure calls for the VAX GPM link (see the next subsection).

Remote procedure calls: The use of remote procedure calls (RPC) allows a routine to be called on the VAX which actually, but transparently to the caller, executes on the GPM. Hence the desireability of maintaining close parallelism between the routines resident on the GPM and those resident on the VAX. The less parallelism there is the more hand tailoring of the remote procedure

call software that has to be done. Our implementation uses the remote procedure call software developed at CERN [10]. The implementation effort for the minimal initial system was (almost) trivial.

A VAX Process: The VAX process for the purpose of the standard routines is exactly equivalent to a VAX VMS process. The identifier is the VMS Process ID, which is passed to the GPM for each call.

A VAX user (process) is able to lock out another VAX user, or the GPM resident application by allocating the FASTBUS port (FB_ALLOCATE_PORT). It is necessary to handle carefully various problematical situations which can arise (for example, if a VAX process dies without deallocating the GPM FASTBUS port). Adding this multiple user capability does complicate the RPC implementation.

Implementation Overheads

Memory resources: Memory is not a particular problem in the implementation for the GPM. Of course the space associated with each environment is unavailable for other uses until the environment is released (FB_RELEASE_ENVIRONMENT). The present size of an environment resident in the GPM is 190 bytes. At worst space is wasted for each environment if there is unused status history memory (64 bytes at present, intended for future support of a supplemental status history), and unused sequential buffers (16 bytes for each unused buffer, at present a maximum of 4 buffers).

For the VAX resident routines, each environment that is created, results in the creation of an environment on the GPM, thus environments for VAX users actually reside on the GPM.

Performance: For the full unrestricted implementation of the standard routines, the average execution time of a typical simple transaction routine (FB_READ_DAT), that is a primary address cycle, followed by a secondary address cycle, followed by a data cycle, was measured to be 198 microseconds, of which 36 microseconds is the duration of AS/AK. A "full" implementation means full checking of input parameter validity, and support of all options (operational parameters).

For a more restricted implementation, that is a minimum of parameter checking and options (no cycle suppression allowed, etc.), the execution time for FB_READ_DAT was measured to be 126 microseconds, of which 22 microseconds is the duration of AS/AK. The normal and "quick" versions of the routines are maintained in separate libraries.

If one codes the transactions in MACRO, that is not using the standard routines at all, and only checking for FASTBUS error conditions, one can proabably achieve an execution time for the same equivalent transaction of approximately 20 microseconds, of which 6 microseconds is the duration of AS/AK.

Implementation Effort: The implementation effort was not carefully monitored. However, the estimated implementation time for the GPM resident routines was approximately 2 to 3 full time months. The implementation of the equivalent routines on the VAX, was less than one month, much of which was devoted to understanding the idiosyncracies of the remote procedure call software.

Summary

The minimum of implementation level details in the standard document is helpful from the point of view of the implementor (and the subsequent user also). The standard for the most part is defined at a fairly abstract and functional level. This gives the implementor more freedom then would otherwise be available, and results in quite portable routines. The explicit provision in the standard for implementation dependent features in operational parameters, error returns, and some routine calls, makes it quite easy to adapt the standard to particular hardware or to a particular application environment.

Some of mandatory features can seem unecessary. The default environment, and the mandatory requirement that primitive routines and interrupt related routines supported by the hardware must be implemented are examples. But a goal of the standard routines is to increase uniformity of implementations, and therefore portability, and these mandatory features help in achieving that goal.

The error reporting defined in the standard document can be difficult to understand, because of its potential complexity. But in our low level implementation of error reporting there were no problems.

In the case of the GPM, some performance improvement could be obtained by supporting delayed execution mode. This is especially true in its use as a host interface. We have not looked at these aspects in detail however. Because of our particular application and the resultant simplicity of the implementation we have chosen not to implement delayed execution mode. It may be useful to do so at some time in the future.

Finally, our initial experience with implementing and using the new standard routines for FASTBUS is definitely a positive one.

References

[1] H. Muller, "Online Fastbus processor for LEP," Proceedings of Conference on Computing in High Energy Physics, Amsterdam, Netherlands, 1985.

[2] Specifications for Standard Routines for FASTBUS, FASTBUS Software Working Group, FSDG085, April 1983.

[3] Catherine van Ingen, "Experience with an Implementation of the Revised Standard Routines for FASTBUS", IEEE

Transactions on Nuclear Science, Vol. NS-32, No. 4, August 1983.

[4] FASTBUS Standard Routines, U. S. NIM Committee, March, 1987.

[5] Creative Electronics Systems, Case Postale 122, 1213 Petit-Lancy, Switzerland.

[6] Dr. B. Struck, Hauptstrasse 95, D-2000 Tangstedt, Federal Republic of Germany.

[7] M. A. Oothoudt et al, "The MEGA Data Acquisition System", Proceedings of Fifth Conference on Real-Time Computer Applications in Nuclear, Particle and Plasma Physics, San Francisco, California, May 1987.

[8] H. v. Eicken, MONICA a Symbolic Debugging Monitor for the M 68000, (User manual in preparation), DD-Division, CERN, Geneva, Switzerland, 1986.

[9] J. D. Blake, Use of Microprocessor Cross Software under VAX VMS, CERN internal report, CERN/DD/SW-LM/T9, Geneva, Switzerland.

[10] T. J. Berners-Lee and A. Pastore, RPC User Manual, CERN DD-Division, OC-Group internal report, Geneva, Switzerland, March 31, 1987.