Fermi National Accelerator Laboratory

FERMILAB-Conf-92/29

# Fastbus Standard Routines Implementation for Fermilab Embedded Processor Boards

S. Kent, G. Oleynik, J. Pangburn, J. Patrick, R. Pordes and M. Votava

*Fermi National Accelerator Laboratory*
*P.O. Box 500, Batavia, Illinois 60510*

G. Heyes and W.A. Watson III

*CEBAF*

October 1992

Presented at the *Nuclear Science Symposium*,
Orlando, Florida, Oct. 27-30, 1992

# FASTBUS STANDARD ROUTINES IMPLEMENTATION FOR FERMILAB EMBEDDED PROCESSOR BOARDS [1]

J. Pangburn and J. Patrick, Fermilab/CDF
S. Kent, G. Oleynik, R. Pordes and M. Votava, Fermilab/OLS
G. Heyes and W. A. Watson III, CEBAF

## *Abstract*

In collaboration with CEBAF, Fermilab's Online Support Department and the CDF experiment have produced a new implementation of the IEEE FASTBUS Standard Routines for two embedded processor FASTBUS boards: the Fermilab Smart Crate Controller (FSCC) and the FASTBUS Readout Controller (FRC). Features of this implementation include: portability (to other embedded processor boards), remote source-level debugging, high speed, optional generation of very high-speed code for readout applications, and built-in Sun RPC support for execution of FASTBUS transactions and lists over the network.

## I. INTRODUCTION

CDF, CEBAF and the Online Support DART project are developing new or upgraded data acquisition systems for experiments in the 1993/94 timeframe. All three projects use FASTBUS front end modules, which are read out by readout controllers incorporating embedded processors. The three projects have settled upon VxWorks [2] as the real-time operating system to be used throughout the daq systems. When it came time to implement software for the readout controllers, we agreed to a (distributed) collaborative project to design and code a new implementation of the IEEE FASTBUS Standard Routines [3]. It is envisioned that the readout controllers and associated software will be used in teststand, module and sub-system commissioning and full daq system applications.

The readout controllers used are the FASTBUS Smart Crate Controller, with an embedded MC68020 [4], and the FASTBUS Readout Controller, with an embedded LR33000 [5].

## II. SCOPE OF THE IMPLEMENTATION

### A. *Language Support*

The implementation supports C calling on the target controller board itself, and C and FORTRAN calling from host computers via the Internet.

C users can either call the lower case short or long name, *frd* or *fb_read_dat*, respectively. These map onto the real name of the function, *fb_frd_c*. The name *fb_frd_c* must be used to invoke the routine from the VxWorks shell, and will appear in the link map.

Similarly, FORTRAN users can either call the upper case short or long name, *FRD* or *FB_READ_DAT*, respectively. FORTRAN calling is not supported in the VxWorks shell environment.

The implementation provides:

1. include files defining all needed parameters, types and error codes.
2. high speed macros to be included in user source code when speed of the standard routines is critical (turbo mode).
3. object libraries for use by on-board applications. This includes one library for every combination of options (see section on library build options).
4. Host object libraries (for IRIX [7], AIX [8], SunOS [6], Ultrix [9] and VMS [10]) for remote communication.

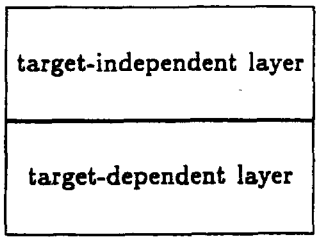
### B. *C Call Arguments*

Routines shall have a void return value unless it is explicitly described by the Standard as a function. IN parameters shall be represented in the argument list by their values, OUT and INOUT by the address of a variable of suitable type. Arguments are as specified in the Standard except for those of type FB_string.

An OUT parameter of type FB_string shall be represented by two function arguments, one of type (char *) and one immediately following it of type (int). The first argument shall give the address of a character array which is to receive the characters. The second shall be set to the size of the array available for writing the data. The routine will write a possibly truncated string whose size, including the terminator, is within this maximum length. An IN parameter of type FB_string shall be represented the same way, but the string shall be treated by the routine as read-only.

## III. Software Structure

To allow easier porting of this implementation to other target boards, the software follows the basic structure outlined below.



Thus, the target-independent upper layer needs no modification to support any number of target boards. Support for a particular target board is supplied via the target-dependent layer, which is implemented as a set of macros and error handling routines which perform all the actual "work" for FASTBUS transactions and error decoding. The upper layer invokes the lower layer transaction macros with a set of parameters which control their behavior. In order to reuse them in turbo mode, the lower level macros must contain no environment structure dependencies. Instead, they are passed everything they need as parameters.

## IV. The Environment

This implementation provides two environments which are implicitly created (created and/or initialized in *FBOPEN*):

> *FB_DEFAULT_EID*: default environment as specified in the Standard.
>
> *FB_TURBO_EID*: turbo mode environment, mostly a set of compiler variables.

Both environments are immediate mode only. Turbo mode cannot by its nature support delayed execution mode, since it causes specific inline code to be created. Environments behave as specified in the Standard.

## V. Error Handling/Reporting

### A. *Exception/Interrupt Handling*

For both the FSCC and FRC, FASTBUS errors generate processor interrupts or exceptions. The target-dependent portion of the Routines includes handlers for the interrupts and exceptions generated by the boards. These routines must behave in the following manner:

The handler routine will post the error detected into a global location *fb_errno* which the action routine checks before proceeding to the next operation. The multi-user synchronization ensures that no context switch occurs during the action, so the task which caused the error will detect it. After posting the error information, the signal handler returns control to the routine. It is the responsibility of the higher level routines to demote the error as appropriate, to report the error, and to return the appropriate return code.

### B. *Automatic Error Reporting*

This implementation supports Automatic Error Reporting. Automatic Error Reporting consists of a call to a user error handling routine, and conditional use of the standard reporting mechanism, via *printf()*.

No status routine invokes the Automatic Error Reporter, nor do *fbopen* or *fbclose*. Therefore, a user application should always explicitly check the return code of these routines for success. Automatic Error Reporting makes use of the error handling parameters in the environment, and therefore must have an environment associated with it. When no environment is valid in the current context, error reporting is invoked using the default environment. Then, return codes from other routines need not be explicitly checked if one wishes to rely on automatic error reporting. See section 11.9 in the standard routine specification for a detailed description of Automatic Error Reporting.

### C. *User Error Handler*

This implementation provides a default user error handler that is automatically connected when an environment is created. There are two choices for this error handler: one does nothing, while the other uses a new Fermilab product called murmur [11]. The product installer has the flexibility to include either one in the standard routine library at product installation time. When the handler which does nothing is installed, error messages will be produced via the default error reporting mechanism. Any other user error handler can be used, as outlined in the Standard, section 11.9.3.

## VI. Implementation Extensions

This implementation's extensions beyond the IEEE standard include:

1. The routine *fb_dump_environment* which produces a formatted display of the current environment.
2. The predefined sequential buffer ID *FB_DATA_PATH* which represents a buffer whose contents are automatically sent to the target board's primary external (auxilliary) port. In a running system this would likely be an interface to an event building system.
3. Remote operation (see below).
4. The routine *fb_port_bind*: a convenience routine to provide a simple way for remote applications to switch among "paths" (target boards) via IP address, and

the parameter *FB_PAR_PATH* in the environment to store the current path.

5. The parameter *FB_PAR_FAST_STATUS* to provide faster routine execution by sacrificing ALL status information.

6. An implementation-provided error handling routine, *fb_fsdeh_c*.

### A. *Remote FASTBUS Operation*

In order to provide greater versatility for FASTBUS applications than the standalone environment of the VxWorks shell, this implementation supports remote FASTBUS operation via TCP/IP. This enables an application running on a host machine to use the FASTBUS port on the VxWorks board exactly as if the application were running on the board. All that is necessary to switch an application between RPC and resident mode is to relink it with an RPC or non-RPC library (since in the non-RPC library, *fb_port_bind* does nothing).

## VII. Restrictions

### A. *General*

This implementation imposes the following restrictions:

1. Retries are not possible in asynchronous mode (*FB_PAR_NOWAIT = FB_TRUE*).

2. *FS_SEV_FATAL* is indistinguishable from *FS_SEV_ERROR*.

3. Not all error codes may have their severity altered by *fb_status_set_severity*. An attempt to do so will result in an error being returned. These attempts are currently not checked.

4. If the user wishes to use the standard routines in asynchronous mode, then the FASTBUS port must be allocated via *fb_port_allocate*.

### B. *Unsupported Routines*

This implementation does not support the compound transaction routines (from Standard, chapter 7), FASTBUS interrupt and SR routines, or line access routines (from Standard, chapter 8). Also, the following routines are not supported in TURBO mode:

*FB_CREATE_IMMEDIATE_ENVIRONMENT, FB_CREATE_DELAYED_ENVIRONMENT, FB_RELEASE_ENVIRONMENT, FB_LIST_EXECUTE, FB_LIST_VALIDATE, FB_LIST_GET_POINTER, FB_PAR_PUSH, and FB_PAR_POP.*

### C. *Unsupported Operational Parameters*

The following operational parameters are not supported:

*FB_PAR_EXCEPTION_THRESHOLD, FB_PAR_REPORT_TERSE, FB_PAR_REPORT_ACTIONS, FB_PAR_BLOCKLET_SIZE, FB_PAR_FIXED_NTA, FB_PAR_PARITY, FB_PAR_SHORT_DATA_WORD, FB_PAR_SHORT_WORD_SIZE, FB_PAR_HOLD_BUS_NO_AR*

## VIII. Conclusion

Although actual implementation of providing for easy changing between the readout and diagnostic modes of running was more complicated than imagined, this implementation has been a success. The software is now being tested at collaborating institutions in the U.S. and Japan, and is ready for experimenters with the very different requirements of Collider, Fixed Target HEP and Medium Energy physics to start writing diagnostic, monitoring and readout code for data taking within the next year.

## References

[1] This work is sponsored by DOE, contract No. DE-AC02-76CHO-3000.

[2] VxWorks is a trademark of Wind.River Systems, Inc.

[3] See IEEE Std. 1177-1989.

[4] G. Cancelo, M. Bowden, R. Kwarciany, J. Urish. "An Intelligent Readout Controller for FASTBUS, The Fermilab FSCC." Fermilab note DAE/0012. 1989.

[5] S. Zimmerman, V. H. Areti, G. Foster, U. Joshi, K. Treptow. "FASTBUS Readout Controller Card for High Speed Data Acquisition." FERMILAB-Conf-91/290.

[6] SunOS is a trademark of Sun Microsystems, Inc.

[7] IRIX is a trademark of Silicon Graphics, Inc.

[8] AIX is a trademark of International Business Machines, Inc.

[9] Ultrix is a trademark of Digital Equipment Corporation.

[10] VAX and VAX/VMS are trademarks of Digital Equipment Corporation.

[11] See Fermilab Computing Division document PN457.