FASTBUS Software in the NA31 Data Acquisition Environment


Abstract

The NA31 experiment on CP-violation by the CERN-Dortmund-Edinburgh-Orsay-Pisa-Siegen collaboration is the first at CERN to take data using a full implementation of the FASTBUS Standard. All of the event-associated data is read via FASTBUS, either directly through a CFI or indirectly through a FASTBUS Block Mover/168E/Buffer Memory into a VAX 11/750. The features and performance of data acquisition using this configuration are discussed.

(presented by K.J. Peach, University of Edinburgh)
1. Introduction

The CERN-Dortmund-Edinburgh-Orsay-Pisa-Siegen [1] experiment on CP-violation at the CERN SPS is the first experiment at CERN to use a full implementation of FASTBUS [2] for data acquisition. The experiment had a short run in July and August 1984, with only part of the FASTBUS system installed, and had a run in April-August 1985 with the complete system. This paper describes briefly the software environment for the 1985 run.

2. FASTBUS configuration

The configuration of the FASTBUS and data acquisition system is shown in Figure 1.

![Diagram of FASTBUS and Data Acquisition System]

Figure 1: Configuration of FASTBUS and Data Acquisition System
The main components of the system are:

1. The FASTBUS ADC system under the control of the PDP 11/23*. This is described in the paper by C Arnault [3] submitted to the conference.

2. A VAX 11/750 to control the experiment (via CAMAC) and to collect the data, write tape, monitor etc. Two CFI's [4] and a FIORI [5] allowed the VAX to access the FASTBUS system.

3. A twin 168E system [6] accessed from the VAX through a CFICC [7] and CAMAC. This system is described in reference 6. The 168E's were used to filter the data, based on a partial reconstruction of the events, and accepted events were placed into a 2MByte buffer memory, for subsequent transfer to the VAX.

4. Three FASTBUS Crate segments and one FASTBUS Cable segment.

One of the FASTBUS crate segments (called the DAQ Crate) contained all of the modules to be read for a single event; these modules have been designed and built within the collaboration, and are briefly described in Appendix A. One special feature of all of these modules is that they had a single-event internal buffer, thus creating a one-event pipeline in the readout chain, and hence reducing the deadtime. The DAQ Crate also contained a CFI for direct event readout by the VAX, a Segment Interconnect [8] to access the cable segment and the second (Auxiliary) crate, and the FASTBUS Block Mover [9] (BM) which transferred data from the DAQ crate through the Switch Coupler Card [10] (SCC) to the 168E system. The auxiliary Crate (which was used mainly for testing purposes) contained the Cable Segment Terminator [11] (CST), a Segment Interconnect, CFI, FIORI and a Fastbus Display Module [12] (FDM).
3. FASTBUS Software Components

The software for this rather simple system must accomplish the following tasks.

1. System Initialization

2. System Testing and Diagnosis

3. Module Testing and Diagnosis

4. Data Acquisition

In fact, only a few programs were required for these tasks. Most problems could be diagnosed without the need for special programs, once the modules had been commissioned. The various software components are described below.

The CERN Standard software routines [13] were used in all applications. However, for some programs, only the List Compiler was used, the compiled set of instructions being stored on disk for later execution. A standard program (BUILDLIST) was written to compile a set of FASTBUS actions (see Appendix B) from a simple command sequence. The execution of these lists did not use the CERN List processor, but a set of CFI-specific routines which handled the CFI Command-Reply sequence and associated DMA using optimized CAMAC QIO's; this reduced the CPU overhead considerably, and was also about a factor of two faster in real time as compared with the standard routines using delayed list execution. Programs which used this second set of routines are indicated with an asterisk. All programs were written in VAX FORTRAN, and many assumed that the terminal was compatible with a VT100; these programs are not therefore easily transportable to other systems.
3.1 System Initialization

A single program FBRESET initialized the three-segment FASTBUS system from data stored in a simple database. Because the system was small, the complete database could be held in memory. The initialization proceeded in three stages.

1. The interfaces were initialized

2. All segment interconnect route tables and registers were initialized and loaded, and the ancilliary logic registers were set appropriately.

3. All segments were scanned for modules in each physical station on the segment (0:25 for a crate, 0:31 for a cable). The results of this scan were compared with the previous contents in the database, and changes were indicated; if there had been a change, the database could be updated on disk if required.

It should be noted that even for this small system, the time taken to initialize was about one minute. This was partly because the responses and loading of registers were checked at each stage during the initialization by the VAX, and partly because the initialization was controlled directly by the VAX, instead of (as would be better) from some intelligence in the FASTBUS system itself, which could perform the initialization after being loaded with the database.

3.2 System Test and Diagnosis

Much of the system was tested by the FBRESET program. However, a more thorough test was required before the multi-segment system was in use, and to test those parts of the system (particularly the AD lines between the GA and GP fields) which were not used in system initialization. The programs developed for these purposes were:

1. CFITEST: This was a general program which used an auxilliary memory (usually the FDM) to write, read and compare test patterns in single word and block transfer geographical and logical addressing; it was a combined test of the CFI,FDM, ancilliary logic and (if used over the cable segment) the SI's and cable segment.
2. FASTCOMP (*): The CFITEST program was too slow for high data volume testing of the cable segment and the SI's. A more simple program, which used only block transfers and geographical addressing with a more limited range of test patterns (only random integers) was used to write, read and compare over the cable. The random numbers were changed by one location, and one new random number was created, for each loop.

3. TESTGAC: This program tested the responses of the ancilliary logic (essentially CSR3) on a directly connected crate segment using an auxilliary FDM. Because it used the FASTBUS primitive routines, this program could only use the FIORI interface.

4. SITEST: This semi-interactive program allowed the loading and testing of Segment Interconnects.

3.3 Module Testing and Diagnosis

There were two types of test programs for modules. Each module had its own test program, usually written by, of interest to and often only understood by, the design team; these are not discussed here. However, once the module had been commissioned, it was often necessary to check from time to time, or when there were problems, simple module-independent functions, for example, to check whether CSR0 was readable, or to discover the settings of the CSR0 bits. These programs are described below, in roughly the order of increasing complexity.

1. FBSTUDY [14] : This program usually used the FIORI interface, and allowed essentially static or 'slow-motion' testing of modules. It had access to all of the CERN standard access mode routines, including the primitives. It also had a simple facility for storing useful test sequences, for executing sequences of operations and for looping. However, its main use was as a very low level diagnostic program, for stepping slowly though a FASTBUS sequence, or for setting the system into a static state for hardware investigation.

2. CFILOOP: This program used the CFI to execute singly or repeatedly the eight functions corresponding to the routines FRD, FWD, FRDB, FWDB, FRC, FWC, FRCB, FWCB.
For output, there were several options available - user defined, alternating, all 0 ,all 1 , incremental, bit-shift and random. This program was most useful when testing individual functions of a module and (by using the CFI EXLOOP facility) to allow engineers to fault trace at high speed without using the VAX CPU.

3. EXECLIST (*): This program had similar facilities to CFILUOP, except that a more complex sequence of operations could be executed, including full data acquisition execution with a simulated trigger.

4. BMTALK [15] : This program was used to control and thus to test the Block Mover, but could also be used with an auxiliary memory to test source modules at faster data rates than was possible with the CFI.

5. TALK [6]: The purpose of TALK was to study, test and debug the 168E hardware, to perform timing measurements, and to develop a library of utility routines from which the final data acquisition task was constructed. It was possible to write, read, compare and dump all six memories in the system, to run programs on the 68000 processor on the Microprocessor Controlled Interface (MCI) and the 168E's, and to perform test loops etc. Together with BMTALK, it allowed a check of the complete data route through the BM/168E system using events downloaded through a CFI.

Several more specific programs have been developed to test particular functions by modifying the EXECLIST program - for example the FASTCOMP program to gain speed, or perhaps to add some CAMAC action for specific modules. The development of these programs from EXECLIST required rather little effort.

3.4 Data Acquisition Software

The data acquisition system was written by the NA31 collaboration to control the experiment, and to acquire the data via the FASTBUS, either directly through the CFI or indirectly through the BM/168E system. The software was written in VAX FORTRAN, and consisted of several independent programs, communicating mainly through a 2MByte shared memory area.
(the DAS Buffer). The main components of the system are described briefly below.

1. A suite of routines (The Micro-controller [16]) to control access to the DAS Buffer.

2. A supervising STATUS_MONITOR program. This program took no part in the management of the data acquisition system, but checked continuously for pathological conditions such as a program crash while locking access to the DAS Buffer, and tidied up the tables to allow the system to continue. This program also maintained a display of the current state of all data acquisition tasks, the state of the DAS Buffer etc.

3. A Master Control program which allowed the operator to control the experiment and the data acquisition tasks (via the DAS Buffer). This was the only program with which the operator communicated directly.

4. An Event Display program [16]

5. A Histogramming and Plotting system [17] to accumulate and display information produced by the data acquisition tasks, and controlled from the master program.

6. Four producer tasks to place data into the DAS Buffer. These tasks were:

   a. ZFASTCAM (*) to control the experiment through CAMAC IO registers, to handle all real-time interrupts and to acquire data through the CFI. This program also exercised direct control over the functioning of the Z168E program.

   b. Z168E to initialize and setup the 168E system, and to empty the 168E system buffer memory at the end of each data-taking burst.

   c. PLAYBACK to read data back from tape.

   d. ZMCDATA to read Monte Carlo data from disk.
PLAYBACK and ZMCDATA have no connection with FASTBUS, and serve in any case obvious functions; the other producer tasks are discussed in more detail below.

7. Up to eight consumer tasks. Three tasks ran permanently, the other five were allocated and run as necessary, for detailed checking of particular parts of the data and equipment. The three principal tasks were;

   a. ZMAG to write the data to tape. This program accepted all the data.

   b. RUNSUM to monitor a sample (1/n) of the events

   c. FORMAT to check the structure of a sample (1/m) of the events.

    Typically, RUNSUM analysed 1/20th of the data and FORMAT 1/100th.

At saturation, the CPU time was roughly 30% for ZMAG, 15% for Z168E and RUNSUM, 5% in FORMAT and the remainder distributed amongst the other tasks in the system.

3.4.1 ZFASTCAM

This program was responsible for the direct control of the experiment through CAMAC, and for servicing all real-time interrupts associated with the SPS machine, as well as taking data through the CFI.

The SPS machine super-cycle of 14.8 secs was divided into five cycles.

1. Cycle 1, of length 6 secs, during which time

   a. Z168E emptied the 168E system buffer memory

   b. the PDP11/23+ had control of the FASTBUS ADC system
2. Cycle 2 of length 3.7 secs, to allow the PDP11/23 to complete its task, and to prepare for the next cycle.

3. Cycle 3 of length 0.3 secs, during which two or three calibration events were read through the CFI by ZFASTCAM. (There was a small amount of CAMAC read in association with these calibration events).

4. Cycle 4 of length 2 secs to prepare for the next cycle.

5. Cycle 5 of length 2.8 secs, during which data was taken either directly through the CFI by ZFASTCAM, or indirectly through the BM into the 168E system, in which case ZFASTCAM was quiescent.

The only interrupts serviced by the VAX were associated with changes of cycle. For data taking during cycle 5, of necessity for the BM/168E system and by choice for the CFI, the individual event 'interrupts' were fielded directly by the interfaces and associated electronics. This had a particular advantage for data acquisition through the CFI, which allowed several events to be read into the VAX during a single DMA, thus reducing considerably the system overhead.

The other tasks performed by ZFASTCAM were principally:

1. Before each run, the DAQ crate was scanned to find the modules in the crate, and the result compared with the contents of the crate stored in the database. From this scan, the appropriate Fastbus lists were constructed for execution at the start of run (SOR), start of burst (SOB) and for data acquisition through the CFI (CYCLE5D). The scan list for loading into the BM was also constructed.

2. At the start of run and start of burst (end of Cycle 1) the appropriate list was executed by the CFI to initialize and reset the fastbus modules.

3. Various experiment modules were set or reset though CAMAC as appropriate at changes of Cycle.
4. At the end of each burst, a set of CAMAC scalers were read, to give information, for example, on beam intensities etc.

5. Monitoring functions were provided for subsidiary tasks as required.

The FASTBUS lists for execution were constructed from prototype lists stored in the database (requiring only the updating of the primary address of each module) for each of the three lists SOR, SOB and CYCLE5D. There was no CERN FASTBUS software directly used by ZFASTCAM.

ZFASTCAM also controlled directly Z168E as necessary. At the beginning of Cycle 1, Z168E was launched to empty the 168E system buffer memory, and ZFASTCAM waited for this to complete before continuing to Cycle 2.

3.4.2 Z168E

In contrast to the multiple tasks performed by ZFASTCAM, the program to control the 168E system was dedicated only to this task. However, the loading and initialization sequence for the 168E system was complicated, requiring the loading of the control program into the Micro-Processor Controlled Interface (MCI), and then the appropriate Filter program and data constants into the two 168E processors.

To increase the reliability of the system, many tests were built into the software, for example the downloading of all programs was verified by reading them back and comparing. Also, when the buffer memory was emptied, the integrity of the data structure was checked before events were released to the DAS buffer.

During each burst, the Z168E checked the status of the 168E system, and tried automatic recovery of the system in case of fault.

To facilitate diagnosis of problems, as well as monitoring the system performance, error reporting included a complete traceback of detected error
conditions; error messages and accounting information were also written into a journal file.

4. Performance details

Some brief details of the performance achieved with this configuration are given in this section, to allow evaluation of the potential of the FASTBUS system through the two interfaces used in this experiment. The actual performance depended on many factors, such as event length and trigger rate, and within this experiment the actual data rates depended strongly on the type of trigger and the beam. These performance figures should then be considered as being 'typical' only for this application. It should also be stressed that there are several ways in which these figures could be, and will be, improved before the next running period in 1986.

4.1 Data Acquisition through the CFI

The data rate through the CFI was limited by two factors

1. The CAMAC transfer time of 1.9μsecs/16-bit word.

2. The need to format the data from the 12 modules to allow fast decoding in the VAX; this took about 200μsecs/module.

With these constraints, the real time to read one event was about 4msecs. In addition, the overhead on launching a preloaded list, and starting the DMA was about 30msecs; however, this was reduced to about 4msecs/event by taking eight events in a single DMA operation, leading to an average time per 2kbyte event of about 8msecs, or 300 events in a burst of 2.8 secs (600kbytes of data).
4.2 Data Acquisition through the BM/168E system

The limitations on the BM/168E system are more difficult to define precisely. The time taken to read a typical event was less than 100\mu s, including formatting the data in the same way as the CFI. However, the filter program took a minimum of 800\mu s to process an event, and if the full filter was run, this rose to more than 1.5\mu s. With the trigger rates in the experiment, and with the particular filter algorithm and philosophy used in the 1985 running, the deadtime was typically 30\%. At this figure, the data rate was 1000 events/burst into the 168E system, and (after filtering) 500 events/burst into the VAX and on to tape. (The filter program rejected about 70\% of the data, but 25\% of the data was written to tape independently of the filter program result, in order to monitor the performance of the system.) This data rate in fact saturated the tape-writing capacity available in 1985.

5. Future Developments

There are some changes required in the data acquisition system before the next run, in order to meet the requirements of the experiment; apart from general improvements in the trigger and the filter algorithms, the changes which affect FASTBUS and the associated software are:

1. Change the CFI interface from CAMAC to Unibus, using the UFI [18] system.

2. Since the CFI and the 168E system both use the same type of CAMAC interface, a similar change from CAMAC to UNIBUS could also be made for the 168E system as for the CAMAC.

3. Install a fast FIFO memory on a second cable segment between the BM and the SCC, in order to reduce the system deadtime.

With these improvements, the data rate into the VAX directly should improve by perhaps a factor two, and into the 168E system by about the same.
6. Summary and Conclusions

A brief description of the main software components for the NA31 data acquisition system has been given. The system has performed reliably during the 1985 run, and some 40 million triggers have been written to tape. The performance of the system requires further enhancement for the 1986 run.

Acknowledgements

It is impossible to acknowledge all the contributions to the work described in this paper. We would therefore like to thank all of those people who have helped in setting up this experiment, and who have offered advice and assistance.
APPENDIX A

BRIEF DESCRIPTION OF THE SOURCE MODULES

A brief description of each of the source modules in the DAQ crate for the data in this experiment is given here; further details are obtainable from the module designers.

A.1 Pattern Unit (F680E)

This module contained 128-bits of information, strobed in 16 groups of 8 bits, and read as four 32-bit words. The module was designed and built at Pisa.

A.2 Wire Chamber Buffer Card (F6813)

This module read the data from the wire chamber encoders, and delivered the data in a maximum of 17 words (one status word, and then a maximum of four hits for each of the 8 wire chamber planes, packed two into each 32-bit FASTBUS word). This module was designed and built at Siegen.

A.3 Time to Digital Converters (F680B)

This module contained 32 6-bit TDC’s and a 5-bit Vernier on the common start. The data were packed into seven 32-bit FASTBUS words. The module was designed and built at CERN.

A.4 ADC System (F682A)

This module in the DAQ Crate served only to connect the DAQ crate to the memory in the ADC Crate, and is described in ref 3.
A.5 Scaler (F680C)

This module contained four 16-bit fast scalers, packed into two 32-bit FASTBUS words. The module was designed and built at CERN.

A.6 Time History Module (F680C)

This module provided 48 channels of Logical State Analysis of various signal in the trigger system and on various discriminated levels in the detector, spanning a range of up to 255 steps about the trigger time. This module was designed and built at Pisa.
APPENDIX B

BUILDLIST PROGRAM

The input file for this program consisted of

1. One line specifying the mnemonic name for the list

2. a BEGIN statement

3. a sequence of FASTBUS instructions

4. an END statement

There were three groups of FASTBUS instructions; the structure of these was:-

1. `<READ|WRITE> <WORD|BLOCK> <DATA|CSR> PA SA (<VALUE|LENGTH)>
   These instructions compiled into the corresponding sequences for single word or block transfers to or from data and CSR space.
   PA and SA are the primary and secondary addresses
   The VALUE was only required for WRITE WORD commands
   The LENGTH was only required for BLOCK transfer commands

2. `INSERT <FCODE|MARKER|WC|MW> (VALUE1 (VALUE2
   where WC inserts a wordcount into the data stream, and MW inserts both a marker word and a word count
   This command entered specific instructions into the FASTBUS list.
   VALUE1 was not required for INSERT WC
   VALUE2 was only required for INSERT FCODE

3. `<SET|CLEAR> CONTROL BIT
   BIT is a bit-mnemonic, as defined in ref 13.
   This did not directly compile into the FASTBUS list, but controlled the setting of the option bits in the control array.

The values could be entered in decimal, Hexadecimal or Octal form.
Example

TESTLIST
BEGIN
SET CONTROL FOASUP
WRITE WORD CSR 0 0 FFFF0000H
SET CONTROL FONOPA
WRITE WORD CSR 0 0 80000000H
WRITE WORD CSR 0 0 00001080H
WRITE WORD CSR 0 0 80000000H
CLEAR CONTROL FOASUP
WRITE WORD CSR 0 16 00004021H
CLEAR CONTROL FONOPA
INSERT MARKER 0000FFFFH
END
Listing of the list built from Example above

*** LISTDUMP *** of FASTBUS list - length 2048 Words (32-bit) options 10012
Bytes i/o 4 0 unused 0 0 Byte-offset 160 unused 0

<table>
<thead>
<tr>
<th>Element</th>
<th>OPCODE</th>
<th>Data</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>101006F</td>
<td>0</td>
<td>Start List, List Identifier</td>
</tr>
<tr>
<td>2</td>
<td>80</td>
<td>1</td>
<td>Trigger/Execute Control</td>
</tr>
<tr>
<td>3</td>
<td>22</td>
<td>0</td>
<td>Embedded Primary CSR Space Address Write</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>0</td>
<td>SA Embedded Write</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>FFFF0000</td>
<td>Embedded Random Data Write</td>
</tr>
<tr>
<td>6</td>
<td>4</td>
<td>0</td>
<td>SA Embedded Write</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>80000000</td>
<td>Embedded Random Data Write</td>
</tr>
<tr>
<td>8</td>
<td>4</td>
<td>0</td>
<td>SA Embedded Write</td>
</tr>
<tr>
<td>9</td>
<td>0</td>
<td>1080</td>
<td>Embedded Random Data Write</td>
</tr>
<tr>
<td>10</td>
<td>4</td>
<td>0</td>
<td>SA Embedded Write</td>
</tr>
<tr>
<td>11</td>
<td>0</td>
<td>80000000</td>
<td>Embedded Random Data Write</td>
</tr>
<tr>
<td>12</td>
<td>4</td>
<td>10</td>
<td>SA Embedded Write</td>
</tr>
<tr>
<td>13</td>
<td>0</td>
<td>4021</td>
<td>Embedded Random Data Write</td>
</tr>
<tr>
<td>14</td>
<td>36</td>
<td>0</td>
<td>AS/AK disconnect</td>
</tr>
<tr>
<td>15</td>
<td>70</td>
<td>FFFF</td>
<td>Insert Embedded Marker into IPB</td>
</tr>
<tr>
<td>16</td>
<td>61</td>
<td>0</td>
<td>End List</td>
</tr>
</tbody>
</table>
References

CERN, Geneva, Switzerland

H Bluemer, G Dietrich, F Eisele, W Heiren, M Kasemann, K Kleinknecht, B Panzer, B Renk
University of Dortmund, Dortmund, Germany

D J Candlin, P McIlnes, J Muir, K J Peach, B J Pijlgroms, I P J Shipsey, W Stephenson
University of Edinburgh, Edinburgh, UK

C Arnault, E Auge, R L Chase, M Corti, J-P Coulon, J-C Drulot, D Fournier, P Heusse, A-M Lutz, J-P Richer
Linear Accelerator Laboratory, Orsay, France

A Bigi, M Calvetti, R Carosi, R Casali, C Cerri, R Fantechi, S Galeoti, A Nappi, D Passuello, G Pierazzini
University of Pisa, Pisa, Italy

C Becker, D Heyland, M Holder, G Quast, M Rost, W Weihs, G Zech
University of Siegen, Siegen, Germany


[3] C Arnault et al., these proceedings.


[7] CAMAC FASTBUS Interface Camac Coupler (CFICC), B Struck (Hamburg), STR300.
[8] These were the University of Illinois designed Segment Interconnects.
[9] L Pregernig, The Block Mover (F6809), (CERN) to be submitted to the IEEE Nuclear Science Symposium, October 1985 (San Fransisco).
[10] M Rost, Switch Coupler Card (F6812), (Siegen)
[11] This was the University of Illinois Cable Segment Terminator.
[14] see A Lacourt, CERN, for details of FBSTUDY.
[15] see P Clarke, CERN, for details of BMTALK.
[16] See M Corti, Orsay, for details of the Micro-controller and Event Display.
[17] see C Youngman, CERN, for details of the histogramming and plotting system.
[18] Unibus-FASTBUS-Interface, B Struck (Hamburg). This uses the standard CFI FASTBUS Coupler, and interfaces to a custom-built personality card on a standard Digital DRE11 unibus interface.