Development of a Data Acquisition System for the BABAR CP Violation Experiment

R. Claus[1], P. Grosso[2], R.T. Hamilton[3], M.E. Huffer[1], C.P. O'Grady[1], J.J. Russell[1], I. Scott[4]

[1]Stanford Linear Accelerator Center, 2575 Sand Hill Rd., Menlo Park, CA 94025

[2]INFN Torino, Italy

[3]The University of Iowa, Department of Physics and Astronomy, Iowa City, IA 52242

[4]Royal Holloway & Bedford New College, University of London, Egham, Surrey, TW20 0EX, UK

Abstract

Experiences developing a data acquisition system for the BABAR CP violation experiment located at the Stanford Linear Accelerator Center are presented. The BABAR detector consists of multiple independent subdetectors joined with a data acquisition system consisting of a large number of embedded PowerPC single board computers residing in VME crates. The data acquisition software is layered on the VxWorks real-time operating system. It is partitionable to allow subsystems (as well as test stands) to operate independently. Data is assimilated into events through a combination of shared memory and a high performance network. This system presents data to a UNIX farm via a high speed non-blocking ethernet switch at a rate of 2 KHz.

Topics such as bootstrapping and loading 200 processors, NFS file access for these processors and software development and deployment are discussed.

I. INTRODUCTION

The BABAR detector was built to study CP violation in the B-meson system. It is now located in the Interaction Region of the PEP-II asymmetric e^+e^- storage ring facility at SLAC. The design of the detector and its Data Acquisition system (DAQ) are described in detail elsewhere [1], [2], [3], [4].

A. The Front End

The system consists of 6 subsystems (Silicon Vertex (SVT), Drift Chamber (DCH), Cherenkov Ring Imaging (DIRC), Electromagnetic Calorimeter (EMC), Instrumented Flux Return (IFR) and Trigger (TRG)) from which data is continuously digitized in so called Front End Elements (FEEs) located on the detector. Read Out Modules (ROMs) receive data from up to 32 FEEs over an optical fiber. ROMs send control signals to their FEEs and can inject signals into the front ends for calibration and data integrity checks over other fibers. The Level 1 trigger is implemented in hardware and uses a small subset of the detector information (e.g., deposited energy, tracks plus geometry constraints) to determine which data is to be processed by the ROMs for inclusion into events. These pieces are called event *segments*. The average *L1Accept* rate is around 2 KHz. Software running on the ROMs performs feature extraction and data compression.

Complete events are built in a hierarchical manner. Event *fragments* are built up in a ROM from event segments on a per crate basis. These fragments are then sent to a node in the UNIX farm where they are built into a complete event. The Level 3 trigger is implemented in software that runs on the farm processors. It is applied to these complete events to extract physics. The resultant rate of events written to the persistent store is 100 Hz.

Table 1 shows data contribution sizes by the various subdetectors (including those of the global (GLT), tracking (TRK) and energy (ENR) trigger systems). The average data rate from the detector is 2.35 GB/sec. This is reduced to 65 MB/sec by the L1 trigger. The L3 trigger cuts this down to 3.2 MB/sec going to persistent store.

Table 1 Subsystem contributions to the data size

Subdetector	Number of ROMs	KB/ROM per Event (Input)	KB/ROM per Event (Output)	Total KB per Event (Input)	Total KB per Event (Output)
SVT	26	0.33	0.33	8.5	8.5
DCH	4	3.5	1.0	14	4.0
EMC_BRL	80	12	108	960	8.5
EMC_ECP	20	8	70	160	1.5
DIRC	12	1	390	12	4.5
IFR	8	1.1	250	9	2.0
Total	155	--	--	1176	32.5

Table 1 Subsystem contributions to the data size

Subdetector	Number of ROMs	KB/ROM per Event (Input)	KB/ROM per Event (Output)	Total KB per Event (Input)	Total KB per Event (Output)
TRG_GLT	1	0.5	493	0.5	0.5
TRG_TRK	3	3	770	9	2.0
TRG_ENR	1	3	1360	3	1.0
Total	155	--	--	1176	32.5

B. The Fast Control and Timing System

The Fast Control and Timing System (FCTS) is responsible for distributing timing signals synchronous with the bunch crossings in the PEP-II interaction point to the FEEs. It also distributes such signals as the L1Accept trigger that is received by both FEEs and ROMs, as well as collecting inhibit (*FULL*) signals with which the system is throttled. Additionally, it provides a messaging path for the data acquisition system. Messages are accompanied by a 56 bit timestamp derived from the 476 MHz RF signal of PEP-II. The timestamp accompanying the L1Accept signal is what the Event Builder uses to recognize event segments coming from disparate ROMs as belonging together. The FCTS was designed to allow the DAQ system to be divided into several independent *partitions* which can be operated simultaneously. This has proved essential to the commissioning of subsystems by their respective groups.

C. Reverse Dataflow

While most of the data flows from the detector toward the persistent store, there is a class of data that must flow in the opposite direction. Items falling in this category are configuration information, calibration signals, and the downloading of code and constants. FEEs have been designed to accept a known signal from the ROMs with which to stimulate the front ends. Whilst this doesn't test the detector element or sensor, it does allow the calibration of the analog data path and determines the integrity of the data path back to the ROM. Other methods are used to calibrate the detector elements and sensors.

II. REQUIREMENTS

Early in the planning stages of the experiment it was decided to use commercial off-the-shelf components where possible and to try to minimize the number of items that had to be developed in-house. This philosophy applies to software as well as hardware. The hope was that this would be more cost effective as well as provide a larger pool of experience and product support. In practice, it is not always clear that benefits gained from using commercial products, with their extraneous features needed to appeal to a wider market and their often short life cycles due to technology advances, outweigh in-house designs that are made to fit. Finding that spares are no longer available a couple years after a system has been implemented can be a big problem.

A. Processor Module

9U VME crates were chosen to be the form factor used to house the data acquisition hardware. PowerPC processors were chosen for the front end CPUs. There are a number of Single Board Computer manufacturers that can provide boards that meet the requirements. Motorola won the bidding competition with their MVME2306 processor boards [5]. These have 300 MHz 604r PowerPC chips, 32 MB of DRAM, 1+4 MB of flash memory, a DEC 21140 ethernet interface, a Tundra Universe PCI/VME bridge, a serial port and two PMC expansion slots. There is no Level 2 cache on this model.

These boards are embedded in the ROM modules with the VME connectors directly on the backplane. The front panels are removed and wiring is added to bring the front panel functions out to the front panel of the ROM. Additionally, a flex-circuit board is added on the back of the MVME2306 to bring the serial port and the LAN connector out through the P2 connector onto a custom interface card that plugs into the back side of the VME crate. The goal was to minimize wiring to the front panel of the ROM so that indicators and switches (reset buttons, etc.) would not be obscured, as well as reducing the number of times that connection have to be broken and remade when swapping modules. 100 Base Tx Full Duplex is run on the LAN so there has been some concern about whether the quality of the signal was significantly degraded by the flex-circuit and/or P2 connector. This has proven not to be a problem.

B. RTOS

VxWorks from Wind River Systems [6] was chosen to be the Real-Time Operating System to be used in the ROMs for mostly historical reasons. At the time (1996), PowerPC support for VxWorks was just emerging. This meant that we were on the cutting edge of making various components of the RTOS work. What we learned in the process is that VxWorks is much more suited to working with small quantities of processors, which is, after all, the model Wind River promotes. In the case of some 160 processors, the demand on the host and network was such that all processors never reliably booted from a common reboot signal (passed as an FCTS message). To solve this problem, a VxWorks program (as opposed to the Motorola-installed flash-resident monitor, PPCBug, which requires a jumper change to invoke) found on the VxWorks net news group to program the VxWorks kernel into one of the on-board

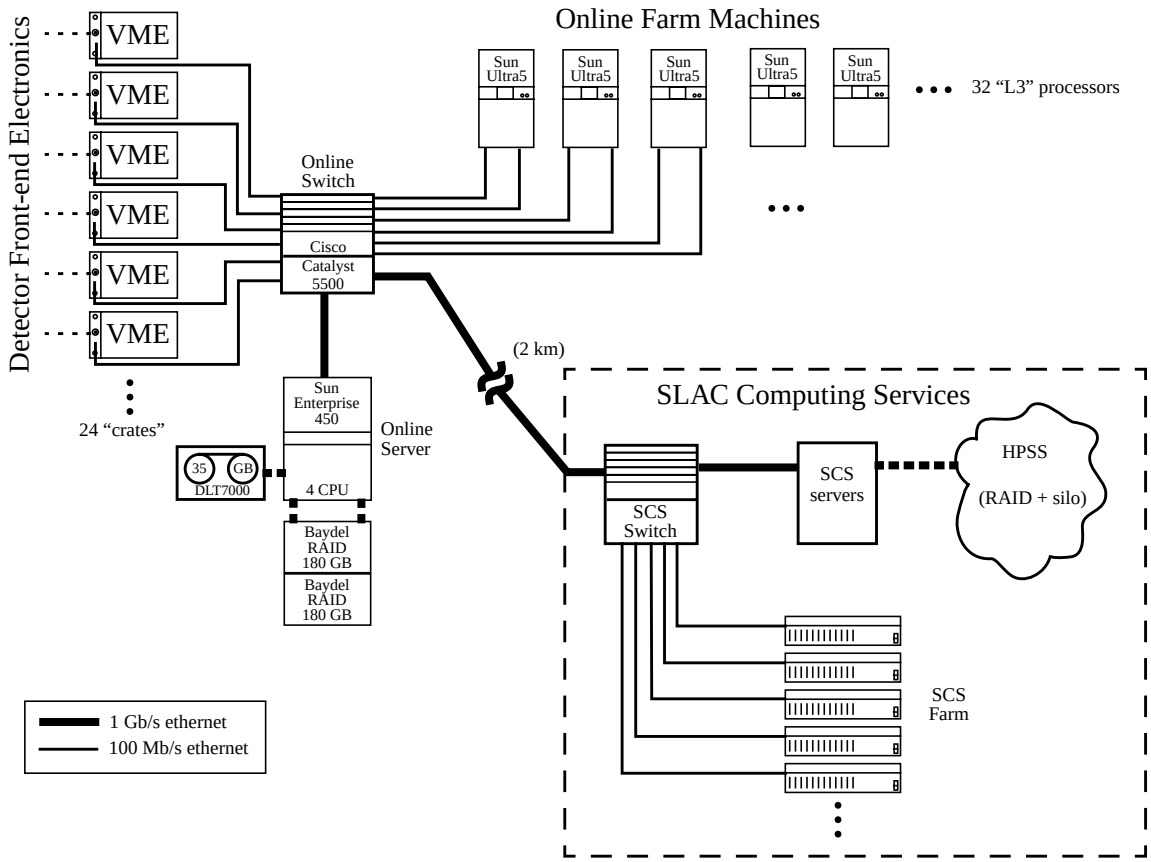

Figure 1: Network layout of the BABAR data acquisition and logging system

flash memories of the MVME2306es was used, under the assumption that it remains fairly constant. The difficulty that this solution presents is that while it takes only a short time to reprogram the flash (~1 minute plus whatever time for verification and testing), it requires an outage that must be scheduled. Outages cause sociological problems and are best avoided. Another drawback is that it is now more difficult to swap spare ROMs in since the version of the kernel that the spare contains in its flash may not be the most up-to-date one. A much better scheme would be to use a third-party loading technique to broadcast load the desired kernel (as well as everything else that needs to be downloaded, e.g., code, constants, etc.) via network multicasting to ROMs in an operator-selected working set. One can then have some confidence in what the processors are running. Unfortunately, there have been more pressing issues preventing us from embarking on this development path.

With the processors booting from flash memory, the boot time for a large quantity of processors has been reduced from several minutes with a long tail (observed to extend to at least 40 minutes) to around ten seconds. The Board Support Package (BSP) was modified to mount an NFS file system on the host server near the end of the booting process. This allows a startup script to be retrieved using the NFS protocol. The startup script contains the commands to load the data acquisition system libraries and executables from the NFS partition. This takes an additional 10 or so seconds. The DAQ software is not burned into flash as it is currently much more variable than the kernel.

Additional modifications to the kernel and also to the host's operating system (Sun Solaris) turned out to be necessary or there were long booting time tails (minutes and occasionally booting didn't complete) even when booting from flash. Items that fall in this category are increasing the NFS retry count, increasing the NFS timeout value, increasing the ARP retry count and avoiding the ARP broadcast protocol to translate an IP address to an ethernet address by seeding the VxWorks ARP table with the appropriate information for the host. On the Solaris server, it was found that the number of threads available for the NFS and mount daemons had to be increased dramatically from their defaults. The problems were generally due to all processors in the DAQ system arriving at the same point in their booting process at the same time and thus overflowing the host's ability to handle network requests, or being swamped themselves due to network traffic from their neighbors.

Only a subset of the users of the front end system modify the VxWorks kernel configuration. To keep track of the changes made to the VxWorks kernel, version management provided by the CVS program [9] is used. To avoid having many copies of the VxWorks distribution in developers' directories, only the changed files are tracked. A shell script creates appropriate soft links to the VxWorks distribution in the developer's directory tree. Kernel developers can thus build their own versions of the kernel independently. To keep track of what is actually in the flash, the VxWorks build procedure has been modified to create a symbol in the kernel with a version code. This code, when

converted to a string, is also used as the name of the CVS tag that identifies a completed development point.

C. Network

The decisions, and validation of the hardware leading to those decisions, made to build the LAN network plant are fairly completely described in [7]. One of the features that was strived for was scalability of the system so that misestimates of the data rate and/or size or potential increases in PEP-II luminosity could be compensated for. To summarize, a Cisco Systems Catalyst 5500 non-blocking ethernet switch is used with 100 Base Tx, full duplex connections to form the fabric over which event data is moved from the front end processors to the Level 3 trigger processing farm. The network is in its own subnet so as to prevent interference from unrelated network traffic and vice versa. A diagram of the plant is given in figure 1. There are some 75 Sun Ultra 5s in the Online Farm. Depending on requirements, more or less of these machines can be allocated to the Level 3 trigger processing. Currently that number stands at 32. The remaining Ultra 5s are used for prompt reconstruction of the data.

There is a gigabit ethernet link from the switch to a Sun Enterprise 450 server. This 4 CPU machine serves the file system and also gives access to the permanent event logging device. Events are intermediately stored to disk for prompt reconstruction purposes. Additional processing power is provided by SLAC Computing Services (SCS) in the form of a farm of Sun machines consisting of some 200 processors. These processors are also attached to a Cisco Systems switch. Since approximately 2 km separates SCS and BABAR, the SCS switch is connected to the DAQ switch with another gigabit ethernet link running on optical fiber.

D. Event builder

Events are built from data segments in two stages. First, feature extracted data segments are moved from data taking ROMs in the VME crates to the corresponding Slot 1 ROM of the VME crate. The Slot 1 ROMs build up event fragments from these segments according to the segments' FCTS timestamps. A table of available farm nodes is established at initialization time. The timestamp is used to form an index into the farm node table to determine which node is to receive the event fragments. This ensures that all fragments of a given event are sent to the same farm node. The UNIX select () mechanism is used to determine when all contributions of an event have arrived, and to time out those that are missing. Because of the time ordered nature of event arrival, the event builder recognizes that when it receives an event fragment that is newer than one it is waiting on, it can declare the old one lost and proceed with the processing of the contributions it already has.

Each stage of the event building has buffering associated with it. The buffering is used to smooth out rate variances. When a stage's buffers are used up, it goes into resource wait mode until the condition causing it is alleviated. If the condition persists long enough, the next stage upstream will run out of buffers and also go into resource wait mode. This process continues up to the point where the FEEs can no longer buffer new data. This causes a FULL signal to be asserted back to the Fast Control and Timing System. Once FULL is asserted, L1Accept triggers can no longer be generated.

The event builder code has gone through several stages of evolution. The portion of the event builder code that is used in the online UNIX farm to build events from event fragments can be used equally well in the Slot 1 ROMs to build event fragments from event segments. In order to handle the traffic between the data taking ROMs and their Slot 1 ROMs, a second Cisco Systems Catalyst 5500 switch has been installed (not shown in figure 1).

One of the early design decisions was to base the network communication on the TCP/IP protocol stack. Initially, the TCP protocol was used as the underlying transport for event data. It provides a reliable communication mechanism with flow control. However, it was deemed that the price paid for these features in the associated CPU overhead on the VxWorks side was too high. It was felt that the datagram nature of the event stream is more suited to the UDP protocol so the event builder was modified to use it. An acknowledgement scheme is used to make the transport reliable. Packetizing was implemented to allow segment contributions to be greater than the 32 KB UDP packet size limit. Datagrams are allowed to grow as large as 200 KB per ROM. However, the dynamics of the system are such that this can't happen too frequently or the switch's buffering will be exceeded and network packets will be dropped. Due to CPU overhead in the network stack, the maximum L1Accept rate that can be achieved with this event builder is 1.3 KHz, below the 2 KHz expected for normal data taking. In order to raise the maximum L1Accept rate that can be handled by the system, we are in the process of creating a segment level event builder to work over the VME backplane.

E. Finite State Machine

The BABAR online system has been built around a finite state machine. Figure 2 shows the state machine exerciser graphical user interface (GUI). The state diagram contains some 22 states for implementing Run Control, Level 1 triggering, calibration, etc. State transition messages are passed around the system via the Fast Control and Timing system. One of the big problems that this presents is how to keep all the consumers of state information (all ROMs in a partition) synchronized so that they all have an identical picture of what state the system as a whole is in. This problem has yet to be solved satisfactorily.

F. Object Oriented Methodology and C++

The BABAR collaboration adopted object oriented techniques and the C++ computer language for both online and offline systems. This practice has also been propagated into the front end processors. The Solaris native compiler is used for the Sun platforms and the GCC compiler is used for the PowerPCs. All features of the language (e.g., exception handling) aren't available for the version of GCC that was supplied by Wind River for use with VxWorks. This causes developers to avoid using

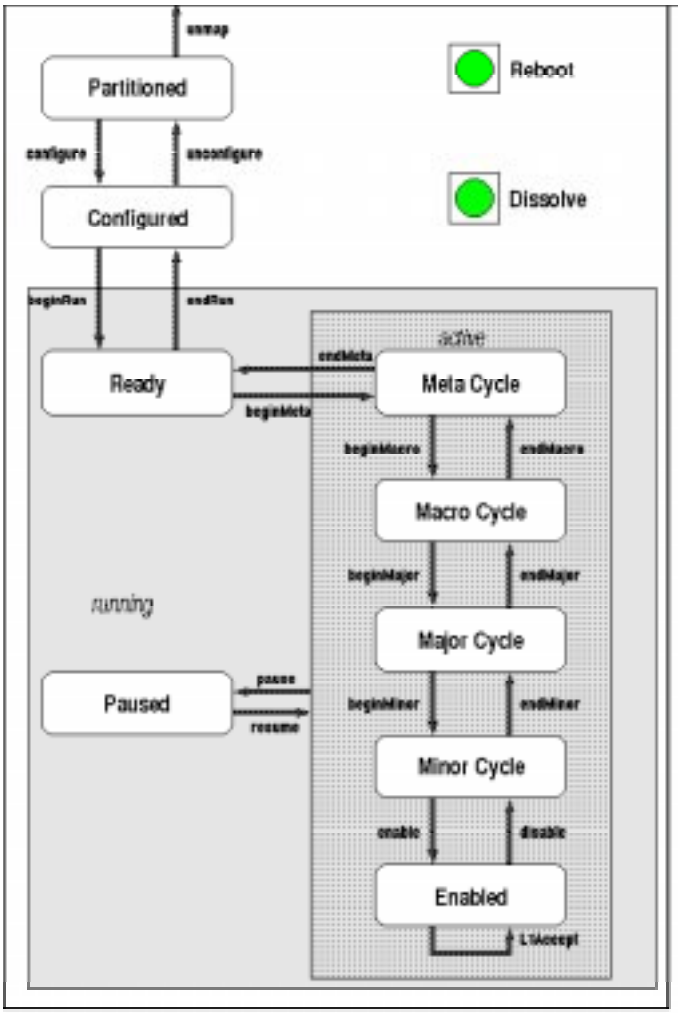

Figure 2: Finite State Machine exerciser GUI

these features on other platforms where they are available to prevent non-portable code from being produced.

A number of features of the C++ language have proven not to be as useful as originally anticipated, such as templates and inlining. It becomes very difficult to maintain independence between disparate blocks in large systems. As a result, a small, seemingly innocuous change to a private function or class section, can cause a large portion of the system to be recompiled simply because a header file was modified.

Header files have other unpleasant side effects. Because header files are often nested, compile time can be increased dramatically as the compiler must open each file and process it as it encounters it in turn. This leads some programmers to replicate the desired bit of header file (e.g., a one line macro definition of a register address) in their code, just to avoid including the file. While it is not difficult for a computer to keep track of the header file dependency tree (e.g. via *make*, etc.), often the programmer, *a priori*, has no idea what the consequences of including a particular header file into his code are.

G. RISC programming

Most of the members of the group that is involved with producing code for the front end processors came with a background in Motorola 68K and similar processors. It turns out that the RISC processors are a different ball of wax. One of the differences that affects the programming method is that the RISC processor has a clock speed many times the bus speed to memory, allowing several instructions to be executed between memory access cycles. This can be used to advantage by rearranging instructions, avoiding memory references and ensuring that the data in the data cache is completely used before the cache is flushed. Processing elements in an array should be done sequentially in a single pass, for example. Keeping loop sizes so that they fit within the instruction cache can also improve performance. Often it is helpful to break up a large loop into several small ones.

Another point is that RISC processors use prefetch and write posting techniques to keep pipelines flowing. One has to be careful when operating on I/O device registers to not allow things to happen out of sequence. Setting a memory *guarded* attribute can help with this endeavour.

By careful study of compiler assembler output, one learns tricks on how to make the compiler produce the best code. The most straight forward way to code something in a high level language does not necessarily generate efficient object code. Timing sections of code can help determine the cacheing effects of the processor. Getting the best performance out of a piece of code is an iterative process.

The PowerPC has helpful features like bus-snooping to detect whether other devices accessing memory have modified locations that it has cached. Atomic operations are carried out through a *reservation* mechanism. If the reservation is still held by the processor after the operation, the operation proceeded atomically. Other entities take away the reservation by accessing the address that is the target of the operation. The PowerPC does not block because a bus master is accessing the address.

H. Code Management

The Online Dataflow code is part of the BABAR collaboration-wide software development. As such it started out as “just” another directory tree that interested parties could check out from the collaboration-wide CVS repository. Despite much effort, it has proven impossible to prevent too many interdependencies to arise between major packages. Thus, changes in one package can make many other packages need recompilation, and a lot of time is spent getting back to a coherent state. During the development phase of the code, where one cycles over making changes and testing frequently, this problem got to be severe enough that one avoided making changes that would cause dependent packages to need to be recompiled, thus impeding development.

In the case of ROM code, this problem arose between the core transport group and the subsystem groups that supply the feature extraction code. To solve the problem, the incremental linker that VxWorks provides is used. The core dataflow code is compiled into object files that are stored in a global area that is used by everyone. Subsystem groups no longer create their own versions of these files. Since the core dataflow group is

now in control of what is in these files, they are easier to update. These files are loaded by the startup script that VxWorks runs at boot time. After the core files are loaded, an application startup script is called. This script typically loads the subsystem feature extraction code. At the conclusion of this script, the acquisition system code is started.

A similar approach has been taken for dataflow code in the UNIX domain. Instead of dependent packages linking against updated versions of the dataflow tree in private areas, the code is now compiled into shareable images stored in a globally accessible place. Symbol resolution thus takes place at run time. This dynamic linking technique ha proven to be a tremendous benefit.

Dynamic version management is another issue to consider. As the software "plant" grows, more and more packages appear that are chained together by the data they consume (clients) and the output others produce (servers). This transfer of outputs to inputs can happen through data files and it can happen via messaging through network sockets, for example. Changes in the server easily prevent the client from consuming the data correctly and vice versa. By including a version number in the data stream, the client can recognize that it doesn't know how to handle the input data and act appropriately instead of producing an incorrect result. Incorrect results can lead to hours of debugging a server program when it is in fact a change to the client made unbeknownst to the server programmer that prevents the system from working.

III. Conclusions

The BABAR data acquisition system has been built and is currently taking data. In the process of its development a number of stumbling blocks were encountered. Some of these were adequately resolved while others provide lessons for "the next time". Much work remains to be done to improve the system's throughput (e.g., the VME event build), and robustness to adverse and uncommon conditions.

IV. Acknowledgments

We gratefully acknowledge Department of Energy contracts DE-AC03-76SF00515 and DE-FG02-91ER-40664 for supporting this work.

[1] BABAR Technical Design Report, SLAC-R-457 (1995).

[2] R.T. Hamilton, *et al*., "The BABAR Data Acquisition System", this conference.

[3] I. Scott, *et al*., "BABAR Data Acquisition", Proceedings of the CHEP 1998 Conference.

[4] P. Grosso, *et al*., "The BABAR Fast Control System", Proceedings of the CHEP 1998 Conference.

[5] MVME2306 Single Board Computers are manufactured by the Motorola Computer Group division of Motorola Inc., 2900 South Diablo Way, Tempe, AZ 85282.

[6] The VxWorks RTOS and Tornado Development interface are products of Wind River Systems, Inc., 1010 Atlantic Ave., Alameda, CA 94501-1153.

[7] T.J. Pavel, *et al*., "Network Performance Testing for the BABAR Event Builder", Proceedings of the CHEP 1998 Conference.

[8] See the comp.os.vxworks net news group.

[9] See http://www.cyclic.com.