Implementation Of UNIX Workstation and BIT3 VME BUS Adaptor in Real Time DAQ System

by

HONGBIN YAO

B.S. Physics, Tsinghua University, China, 1987
M.S. Control, Institute of High Energy Physics, Academia Sinica, 1990

Submitted to the Department of Physics
in partial fulfillment of the requirements for the degree of
Master of Science in Physics

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY
February, 1997

© Massachusetts Institute of Technology, 1997. All Rights Reserved.

Department of Physics
January 23, 1997

George S.F. Stephans
Department of Physics
Thesis Supervisor

Accepted by

George F. Koster
Chairman, Physics Graduate Committee
Department of Physics

FEB 1 2 1997
Implementation Of UNIX Workstation and BIT3 VME BUS Adaptor in Real Time DAQ System

by

HONGBIN YAO

Submitted to the Department of Physics on January 23, 1997, in partial fulfillment of the requirements for the degree of Master of Science in Physics

Abstract

This thesis work is the first reported attempt to use a UNIX based workstation and BIT3 VME bus adaptor along with other SBCs(Single Board Computer) in a VME crate to perform a real time task in a DAQ(Data Acquisition) system. It is designed to increase the DAQ rate for experiment E917(which is an extension of experiment E866 at the AGS accelerator at Brookhaven National Lab) to allow the study of rare processes in Au + Au collisions. To solve the bottle neck of buffer memory, data transferring and event translation in the old DAQ system, a SGI workstation, one BIT3 GIO-VME bus adaptor and two BIT3 VME-VME bus adaptors are added. In this work, a mechanism is developed to synchronize between the processes on the SGI workstation and on the VME SBCs to enable real time performance and the effectiveness of the old run menu control by exchanging and polling flags. A new event processing mechanism on the SGI workstation is used, and a faster event transferring mechanism is implemented, in which the SGI workstation transfers large blocks of data from the VME crate via BLK DMA transmission. Code on the SGI workstation is developed to deal with event transferring, event building, event translation, event logging, event broadcasting and tape operation. Measurements are also made to test the performance of the new DAQ system under a variety of test and run conditions.

Thesis Supervisor: George S.F. Stephans
Title: Principal Research Scientist, LNS
# Table of Contents

Chapter 1  Introduction .............................................................................................................. 11  
1.1  Motivation of this thesis ................................................................................................. 11  
1.2  Introduction to DAQ system of Heavy Ion physics experiment ................................. 12  
1.3  Objective of this thesis work ......................................................................................... 15  
Chapter 2  Design of E917 DAQ Upgrade ........................................................................ 17  
2.1  Design philosophy ......................................................................................................... 17  
2.2  Overview of the design of E917 DAQ upgrade ............................................................. 17  
   2.2.1  Hardware configuration ........................................................................................... 18  
   2.2.2  Software configuration .......................................................................................... 22  
2.3  Coordination between the hardware and software ......................................................... 25  
Chapter 3  Design development ............................................................................................. 33  
3.1  Development Environment, hardware and software .................................................... 33  
3.2  The application of BIT3 VME-GIO, VME-VME high speed DMA engine .................. 34  
   3.2.1  Introduction to the BIT3 interface ....................................................................... 34  
   3.2.2  The application of the BIT3 interface .................................................................... 36  
3.3  Realization of real time response in a UNIX environment ............................................. 41  
3.4  New data processing mechanism: data merging and event translation on SGI workstation .............................................................................................................................. 46  
   3.4.1  Event Merging ....................................................................................................... 48  
   3.4.2  Event translation ................................................................................................... 49  
3.5  Multi level buffered data logging ................................................................................ 50  
3.6  multi job mechanism .................................................................................................... 52  
3.7  Event broadcasting, accessing database and others .................................................... 54  
Chapter 4  Performance measurements ............................................................................ 57  
4.1  Performance measurements .......................................................................................... 57  
   4.1.1  Performance of the BIT3 interface ....................................................................... 57  
   4.1.2  Memory mapping access speed via the GIO-VME adaptor .................................. 62  
   4.1.3  Data transfer speed of the BIT3 413-1 VME-VME adaptor ................................ 63  
   4.1.4  Disk writing speed ............................................................................................... 64  
   4.1.5  Event processing speed ....................................................................................... 65  
   4.1.6  Performance estimate .......................................................................................... 67  
4.2  Run parameter setting .................................................................................................. 68  
Chapter 5  Conclusions .......................................................................................................... 71  
5.1  Performance under the real run conditions ................................................................. 71  
5.2  Suggestions ................................................................................................................ 72  
Appendix A  Glossary ............................................................................................................ 75
List of Figures

Figure 2.1: E917 DAQ system hardware configuration diagram ........................................21
Figure 2.2: Software configuration on the SUN workstation ........................................23
Figure 2.3: Software configuration on the TADPOLE ..................................................24
Figure 2.4: Software configuration on the SGI workstation ........................................25
Figure 2.5: Data Flow in the E917 DAQ system (courtesy of ChuanMing Zou) .............30
Figure 2.6: Time relation during Data taking ................................................................31
Figure 3.1: Diagram of the BIT3 GIO-VME adaptor function blocks ............................35
Figure 3.2: Data flow through the BIT3 interfaces in E917 DAQ system .....................41
Figure 3.3: Block flow of the main program on the SGI workstation ............................45
Figure 3.4: YBOS bank format .....................................................................................46
Figure 3.5: Memory map of the buffer for event processing on the SGI .........................48
Figure 3.6: Format of the first YBOS record .................................................................51
Figure 3.7: Format of a regular YBOS record ...............................................................52
Figure 4.1: Speed measurement of lseek() in BIT3 driver (1000,000 loops) ................58
Figure 4.2: Distribution of GIO-VME reading speed measurement data (1000 loops) 59
Figure 4.3: Measurement of PIO speed of BIT3 interface (1000 loops) .....................60
Figure 4.4: Measurement of speed of DMA with poll (1000 loops) ..........................61
Figure 4.5: measurement of DMA speed with interrupt (1000 loops) ........................61
Figure 4.6: Measurement of memory mapping speed of GIO-VME adaptor (1000 loops) .....................................................................................................................62
Figure 4.7: Measurement of the BIT3 413-1 adaptor data transfer speed (1000,000 loops) ................................................................................................................63
Figure 4.8: Measurement of disk writing speed ............................................................65
Figure 4.9: Measurement of the event merging speed ..................................................66
Figure 4.10: Measurement of the event translation speed ............................................67
List of Tables

Table 2.1: Communication mechanism in E917 DAQ system.............................. 26
Table 2.2: CAMAC partitions and their rough data size .................................. 27
Table 2.3: FASTBUS partitions and their rough data size ............................... 27
Table 3.1: Flags used in E917 DAQ system.................................................. 43
Table 3.2: Data exchanged between the SGI and the VME via memory mapping .... 43
Table 3.3: Segment size in the event processing buffer on the SGI ..................... 47
Chapter 1

Introduction

1.1 Motivation of this thesis

If the data analysis is the brain of a physics experiment, then the data acquisition system will be its nerve. Its job is to record the result of the measurement and save it on the storage media for future analysis. In modern high energy heavy ion collisions, the beam is very expensive. This makes it necessary for the data acquisition system to have a good real time response and the ability to take data as fast as possible. That is especially important while studying rare reaction channels.

Modern high energy heavy ion physics experiments are an important part of the fundamental research which explores the nature of the material world and improves our understanding of the universe. They also supply the basis for future technology. In the meantime, as the experiment size becomes larger and larger, its data acquisition system becomes more and more complicated. To deal with large data size and high data rate from many components of the experiment puts higher requirements for computing capability, fast electronics, high resolution measurement, large amounts of data storage and real time control. On the other hand, parallel computing, advances beyond industry standard, robot and other types of high technology are also extensively used in the data acquisition system of a heavy ion physics experiment.

Experiment facilities can work for a long time. By replacing or redoing some key equipment with new technology, one can do some new measurements. E917 is a continuation of experiment E866. In E917, TRIMIT, a new trigger chamber with larger acceptance
and more wires was built at MIT using new technology; also, BVER, a new vertex detector was put in, so the data acquisition system had to be modified accordingly.

In our experiment, each event size is about 10-14KB. For each event, the data from the front-end electronics are collected into two VME crates (each of which takes about half of the event), and buffered on the TADPOLE, the major single board computer in the VME crate. In the last run, with a total of 4 sec spill length, in which the on-spill length is 1.2 sec (it is actually 1.4 sec but is cut to 1.2 sec by software because of the memory space limit), the event rate is about 260/spill (i.e. 2.6-3.6MB/spill total or 1.3-1.8 MB/spill per VME crate).

In the old data acquisition system, the data transfer speed (specifically from the SBE or the FIFO to the TADPOLE’s memory), the buffer memory, and the CPU power are the bottle necks. By using a faster data transfer mechanism, expanding the buffer memory space, and replacing a more powerful CPU, it is possible to improve the performance of the data acquisition system by a factor of 2 or more without tremendous modification of the old system. This is the original motive for this work. In the mean time, a lot of other experiments may be upgraded in the future. From this work, we will gain some useful experience.

The next two sections contain a brief description of the structure of the data acquisition system in a heavy ion physics experiment, and an overview of this thesis.

1.2 Introduction to DAQ system of Heavy Ion physics experiment

There are many ways to define a data acquisition system.

According to the requirement for the time response, it can be divided into:
1. Real-time system, which requires the system to respond within a time limit. This is the case in the data acquisition system of a heavy ion physics experiment.

2. Non real-time system, which does not require the system to respond by a certain of time limit.

According to the number of CPUs in the system, it can be divided into:

1. One CPU system, in which all the control and computing work is done by one CPU. Its advantage is that it is easy to maintain, and its structure is simple; the disadvantage is that if this CPU fails, the whole system breaks down. Therefore, it is not very reliable, especially when it becomes larger.

2. Distributed system, in which many CPUs are coupled together to some extent, while each of them performs a specific job. Its advantage is that it is more reliable. Moreover, for a large system, it is easier to distribute the work to many CPUs rather than to find one super computer to do all the work.

According to how the CPUs are coupled together, the distributed system can be divided into:

1. Hierarchy system, in which a CPU at higher level controls one or more lower level CPUs that directly relate to it, and there can be many levels. A CPU at lower level usually just responds to its master.

2. Parallel system, in which all the CPUs are tightly coupled to each other to perform the task, and each CPU does a specific job. If one CPU fails, only the relevant fraction will be effected.

In the design of the data acquisition system for a heavy ion physics experiment, distributed structure is well accepted. In practice, hierarchy structure and parallel structure are both used.
Like other data acquisition systems, that for a heavy ion physics experiment is also composed of 7 function blocks:

1. Front-end electronics and digitizers which convert the physical signals (mainly electrical voltage) to digital signals so that they can be read into the computers.

2. Trigger system which picks out the interesting physics events and generates the strobe signals to start the data taking process, or otherwise resets the front-end electronics in time.

3. Data transfer system, which transfers the digitized signals into the memory of the computers.

4. CPUs, which control the whole system and process data.

5. Storage media, which are used to save the data from the measurement for future analysis.

6. Monitor devices, such as sensors, scalers, and alarms ...etc., which are used to monitor the status of the system.

7. Consoles, which supply the user interface to control the system, and also keep the user informed of the current status of the system.

The software for a data acquisition system is primarily composed of 5 parts:

1. Operating system or executive, which is the interface of the user’s application program to the CPU. It also supplies job control, memory management, file management, and other utilities. Usually, it is a commercial product that comes with the computer.

2. Data reading programs, which control the data transfer system and get the data from the digitizers into the computer’s memory.

3. Event building programs, which put together in a certain structure the data for each measurement from different partitions of the experiment.
4. Data processing programs, which further translate the information of the electronics into that of the physical sensors (wires), pack the data into a certain format, and write it into files on the storage media... etc.

5. Run control programs, which implement the algorithm of the data taking control and the user interface in graphics and texts, and also monitor the status of the system.

1.3 Objective of this thesis work.

This thesis work is the first reported attempt to use a UNIX based workstation and BIT3 VME bus adaptor along with other SBCs(Single Board Computer) in a VME crate to perform a real time task in a DAQ(Data Acquisition) system. It is designed to increase the DAQ rate for experiment E917(which is an extension of experiment E866 at the AGS accelerator at Brookhaven National Lab) to allow the study of rare processes in Au + Au collisions. To solve the bottle neck of buffer memory, data transferring and event translation in the old DAQ system, a SGI workstation, one BIT3 GIO-VME bus adaptors and two BIT3 VME-VME bus adaptors are added. In this work, a mechanism is developed to synchronize between the processes on the SGI workstation and on the VME SBCs in order to enable real time performance and the effectiveness of the old run menu control by exchanging and polling flags. A new event processing mechanism on the SGI workstation is used, and a faster event transferring mechanism is implemented, in which the SGI workstation transfers large blocks of data from the VME crate via BLK DMA transmission. Code on the SGI workstation is developed to deal with event transferring, event building, event translation, event logging, event broadcasting, and tape operation. Measurements are also made to test the performance of the new DAQ system under a variety of test and run conditions.
In this design, the VME system is the event taking engine and is event (or trigger) driven. The SGI workstation is the data processing engine and is spill (or more precisely, flag) driven. The SUN workstation supplies the graphic menu user interface and always works as the control console. The details of the coordination between the SGI, the SUN, the VME system and the front-end electronics are described in section 2.3.

This thesis consists of 5 chapters. The second chapter will talk about the design of the upgrade of the E917 data acquisition system, and the coordination of different function blocks in the design. The third chapter will discuss the development of the major pieces in this thesis project. Chapter four will focus on the performance of the new data acquisition system and also give the result of the performance of different pieces of the design. Finally, chapter five will talk about the performance of the system under real run conditions, and then conclude with some comments.
Chapter 2

Design of E917 DAQ Upgrade

2.1 Design philosophy

The plan of the DAQ upgrade for E917 experiment started about one year ago, and the development started about a half year later, April, 1996. Since the physics run started in early November, there was only about six months time limit. In general, time, manpower and funding were the major concerns in the development of this project.

On the other hand, in case the new system did not work properly by that time, there should be an easy way to switch back to the old DAQ system so that the whole physics run will not be ruined. Therefore, the guidelines for the development of this upgrade project are: focus on the major bottle necks of the old DAQ system; while trying to improve its performance, manage to minimize the modification both in hardware and software in order to reduce the amount of work that is involved in the development; take advantage of the old system as much as possible; and design the new system in a way that it can be easily switched back to the old version in case that the new system does not work.

2.2 Overview of the design of E917 DAQ upgrade

E917 experiment consists of beam line, beam counters, target, level 1 trigger, level 2 trigger, more than 10 sub-detectors, FASTBUS and CAMAC digitizers, VME system and workstations. The general procedure of data taking is, while the experiment is running, the trigger supervisor(level 1) will accept trigger bits from each sub-detector. If after a specified scale down, no enabled trigger condition is satisfied, the trigger supervisor will issue a fast clear to the front-end electronics, otherwise it will start data taking unless level 2 is used. If so, a further decision will be made under stricter trigger conditions.
In E917’s data acquisition system, when data taking starts, the physical signals coming from the detector are first amplified and discriminated, and then digitized by the digitizers in the FASTBUS or CAMAC crate in parallel (for those channels that are hooked up to the same digitizer module, it may not be, and that depends on the design of the individual digitizer). In each FASTBUS or CAMAC crate, there is a segment manager which transmits the digitized data to the VME system through an optical link after the A/D conversion is done. Each CAMAC crate corresponds to a SBE board which is a MC68020 based single board computer in the VME crate, while each FASTBUS crate corresponds to a FIFO memory board. One SBE single board computer can manage up to eight FIFO boards. After data get into the FIFO’s or the SBE’s memory, the SBE will notify the TADPOLE, which is a MC68040 based single board computer in the same VME crate to read the data into its own memory, and after all the data from each partition has been read in, the TADPOLE will take further action to start the event processing procedure, and clear the trigger supervisor to start another data taking cycle. Beyond this point, the upgrade DAQ system is different from the old one. Please refer to section 3 in this chapter for further discussion.

2.2.1 Hardware configuration

The primary concern in the DAQ system improvement is “speed”. First of all, the hardware itself has to be fast, next there should be a good algorithm and mechanism. From our research, we know that the data transfer speed from SBE’s and FIFO’s memory to the TADPOLE’s memory, the CPU power, and the on_board memory space are the major bottle necks in the old system. In the VME crate, both SBE and TADPOLE run VRTX32 real time executive. A lot of software is already there to handle the data reading, the communication between SBE and segment manager, and the communication between SBE and TADPOLE. It takes a lot of work before one is able to make modifications or replacements
to these systems. Because we don’t have the time, manpower or money, we decided to keep part of the structure of the old system from the SBE single board computer to the front-end electronics.

To solve the major bottle neck, generally, we have two options:

1. Replace TADPOLE with a new VME Single Board Computer which has a more powerful CPU, more memory space, and a faster DMA engine.

2. Add to the old system a powerful workstation along with BIT3 VME adaptors.

SBC products have a better real time performance because they are supported by a real time executive. But this option was given up quickly because it turned out that the price for a reasonable SBC product, along with its supporting software, is almost as much as that for a good workstation, and there are two TADPOLEs in the system. On the other hand, the supporting software and development tools for the newer SBC product are different from those used for SBE and TADPOLE. This make it harder for program development because we have to deal with different environment in the same system, let alone that the software development cycle for SBC is longer.

BIT3 is a tightly coupled connection on the bus level between the workstation and the VME bus system, or between VME and VME system. Compared with network protocol, such as Ethernet and TCP/IP, it has a much higher bandwidth(26 MB/sec vs. 1 MB/sec). By using its optional dual port memory daughter card, we can add some extra memory space to the VME system. Via BIT3, we can easily switch to any workstation just by rewriting the interrupt handler if necessary (we don’t even have to if no interrupt is involved in our application). For us, it also means that we can keep using the TADPOLEs, and thus keep the whole hardware structure in the old system. This will save our investment, and also make it much easier to switch back to the old system if there is a need. Another advantage is that a workstation can be easily used for other purposes in the future.
From the point of view of hardware, the E917 DAQ system can be logically divided into two parts:

1. the newly added devices.
2. the old hardware that came with the old system.

In the E917 DAQ system, a Silicon Graphics CHALLENGE S/R5000(180MHz) workstation which is more than 10 times faster than the TADPOLE is added in. Besides, two 9GB HillBOX hard disk drives are hooked up to its two fast wide differential SCSI interfaces respectively; four TTI 8mm tape drive are chained to its narrow single ended SCSI interface; one BIT3 607 GIO-VME adaptor is plugged in its unit 0 GIO bus interface, and the Ethernet is connected. In the mean time, two BIT3 413-1 VME-VME connectors are also put in to transfer data between VME memories(each 413-1 consists of two identical VME cards). To make system switching easier, an additional VME crate is used to hold the BIT3’s VME cards; the two tape drives that were hooked up to the TADPOLE are kept although they are not used anymore. Except for the newly added devices, all the hardware is at the same position as before, so whenever there requires a switch, no change in hardware is needed.

In the second part mentioned above, there are two VME crates and one SUN IPC workstation. Each VME crate takes charge of a specific section of the experiment. The elements in each VME crate are: TADPOLE model TP40V single board computer, SBE single board computer, FIFO memory card, FBVME card(a SBE type single board computer that is in charge of FIFO cards, for convenience, we also use SBE to refer it), and interrupt module which talks to the trigger system.

In the old system, there is a crate interconnection between the two VME crates. With BIT3’s 413-1 VME connection being introduced, the old interconnection is still kept, but not used anymore.
The diagram of the new DAQ system's hardware configuration is shown as Fig. 2.1.

**Figure 2.1:** E917 DAQ system hardware configuration diagram
In Fig. 2.1, the sections that are not used by the new system are labeled with ‘*’.

2.2.2 Software configuration

In the new DAQ system, all the code on the SUN workstation, on the SBE single board computer, and on the segment manager in a CAMAC crate or a FASTBUS crate are changed very little in order to minimize the amount of work involved in the software development. Except that:

1. On the segment manager of a crate in which a new detector such as the BVER is put, additional reading code is inserted.

2. The run RDB database is moved to a new Alphastation. Accordingly, the rsh(remote shell) command used by the SUN workstation to access the RDB database is modified.

As to the software that run on the TADPOLE originally in the old system, the final event building, event translation, data logging, and broadcasting are stripped off and moved to the SGI workstation; while the run control(also called chairman), event reading; primary event building(it only builds the event within the same VME crate) are kept. Meanwhile, on both the SGI workstation and on the TADPOLE, additional software is put in to transfer data and coordinate with each other via the BIT3 interface.

The SUN workstation serves as the run control console and the development platform for the single board computers (such as the SBE and the TADPOLE) in the VME crates. During the run, it is also responsible of loading data or instructions and passing them through the Ethernet to the VME system or further to the CAMAC or the FASTBUS crates. Its software configuration along with its dependence on its host operating system and its utilities is shown in Fig. 2.2.
Figure 2.2: Software configuration on the SUN workstation

The VME system (including TADPOLE, SBE and FIFO) serves as the data taking engine, and is the soul of the DAQ system. In the VME crate, the TADPOLE is the master, and is responsible for the event reading and the primary event building. The SBE, more like a secretary, collects data from the CAMAC and the FASTBUS crates and waits for the TADPOLE to read. The TADPOLE in the top crate also interprets the instructions from the run console (the SUN) and administrates the data taking procedure. Its software configuration is shown in Fig. 2.3.
The SGI workstation serves as a real time data processing engine, its object is to free the CPU power and memory space of the TADPOLEs so that they can concentrate more on the data taking. On the SGI, event data are finally built, translated, broadcasted and saved. There is a control routine on the SGI to administrate the data transfer from the VME system via the BIT3 interface, to manipulate the data processing, and to coordinate with the instructions from the control console (the SUN). Its software configuration is shown in Fig. 2.4.
2.3 Coordination between the hardware and software

All the code on the single board computer (TADPOLE and SBE) and on the segment manager are loaded and started by the SUN workstation. However, the program on the SGI workstation has to start first, because the buffer memory and the BIT3 interface need to be set up before the whole data taking procedure can start. After the system starts running, it can run forever if no accident happens.

The VME system is the event taking engine and is event (or trigger) driven. The SGI workstation is the data processing engine and is spill (or more precisely, flag) driven. The
flags reside in the BIT3’s dual port memory card. As a result, the TADPOLE can access
them from one side (i.e. the VME bus) and the SGI workstation can access them from the
other side (the BIT3 cable) simultaneously, and no conflict will happen. The SUN worksta-
tion supplies the graphic menu user interface and always works as the control console.
After a control instruction is issued, the code or data will be downloaded to the VME sys-
tem (or further to the FASTBUS/CAMAC crates via optical link) from the SUN through
the Ethernet. The SGI copies from the SUN every time before starting a new run the con-
figuration files that are used to form the translation order and translation table. From the
point of view of data taking, the top and the bottom VME crates are parallel.

The communication mechanisms between the SUN, the SGI, and the VME system are
shown in table 2.1

<table>
<thead>
<tr>
<th></th>
<th>SGI</th>
<th>SUN</th>
<th>VME</th>
</tr>
</thead>
<tbody>
<tr>
<td>SGI</td>
<td></td>
<td>Ethernet</td>
<td>BIT3</td>
</tr>
<tr>
<td>SUN</td>
<td></td>
<td>Ethernet</td>
<td></td>
</tr>
<tr>
<td>VME</td>
<td>BIT3</td>
<td></td>
<td>Ethernet</td>
</tr>
</tbody>
</table>

In the E917 data acquisition system, there are two kinds of digitizer, ADC and TDC,
which sit either in a CAMAC crate or a FASTBUS crate. Each CAMAC crate corresponds
to a SBE VME single board computer, and each FASTBUS crate corresponds to a FIFO
memory board in the VME crate. In both scenarios, an optical link is used to transfer data
between the VME crate and the FASTBUS or the CAMAC crate. A FIFO memory board
is managed by a SBE type single board computer, and each SBE can manage up to 8 FIFO
cards. The data size of a CAMAC or a FASTBUS partition is different from event to event,
the rough numbers are listed in Table 2.2 and 2.3.
Table 2.2: CAMAC partitions and their rough data size

<table>
<thead>
<tr>
<th>Item</th>
<th>Partition</th>
<th>Rough size(Word)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>BEAM</td>
<td>82</td>
</tr>
<tr>
<td>2</td>
<td>LVL2</td>
<td>50</td>
</tr>
<tr>
<td>3</td>
<td>TOF1</td>
<td>15 to 70</td>
</tr>
<tr>
<td>4</td>
<td>TOF2</td>
<td>15 to 70</td>
</tr>
<tr>
<td>5</td>
<td>TOF3</td>
<td>15 to 70</td>
</tr>
<tr>
<td>6</td>
<td>T35</td>
<td>73</td>
</tr>
</tbody>
</table>

Table 2.3: FASTBUS partitions and their rough data size

<table>
<thead>
<tr>
<th>Crate</th>
<th>Partitions</th>
<th>Rough size(LW)</th>
</tr>
</thead>
<tbody>
<tr>
<td>FBU0</td>
<td>PHOS/HODO, ZCAL, NMA, BVER</td>
<td>1573/1189*</td>
</tr>
<tr>
<td>FBU1</td>
<td>TOF, GASC</td>
<td>482</td>
</tr>
<tr>
<td>FBU2</td>
<td>HH_TRCK1</td>
<td>212 to 500</td>
</tr>
<tr>
<td>FBU3</td>
<td>HH_TRCK2, TRF</td>
<td>117 to 300</td>
</tr>
</tbody>
</table>

(* For FBU0, 1189 is the data size when part of the PHOS ADCs are taken out of the data stream to increase the event per spill. The data size of each crate varies from event to event depending on the multiplicity of the interaction.)

According to Table 2.2 and 2.3, each event size is about 10 to 14KB.

While starting a run, the SUN accesses the run RDB database through the Ethernet to set up for the run, then the TADPOLE will start data taking and set the NEW-RUN flag, after the SGI workstation polls the flag, it will open a run data file.

While taking data, during the on-spill time, if the trigger condition is satisfied after specific scale down, the trigger system will start the digitizers in the FASTBUS or the
CAMAC crates. After the A/D conversion is done, the segment manager in each crate reads the digitized data as programmed, then passes them to the SBE’s or the FIFO’s memory in the VME crate from the front panel through the optical link. For each event, the data from the front-end electronics are collected into two VME crates (each of which takes about half of the event). After the data from a partition are transferred to the VME crate, the corresponding SBE will inform the TADPOLE in the same VME crate. Then the TADPOLE will read data from the SBE or the FIFO across the VME bus. To increase the speed, a MC68040 instruction MOVE16 which directly transfers 16 bytes at one time is used. After all partitions have been read, the top TADPOLE will reset the trigger, and then the next event taking cycle will start immediately.

During the data taking, all the digitizers, all the segment managers, all the SBEs and the FIFOs, and both TADPOLEs work in parallel.

When the spill is over, the TADPOLEs first transfer the buffered spill data in their memory via Block Mode DMA to their corresponding BIT3 dual port memory cards through the BIT3 413-1’s cables (which are hooked to the front panel of the VME crate) respectively, then set the SPILL flag. As soon as the SGI workstation finds out the flag is set, it will read the spill data from the dual port memory cards through the VME bus and BIT3 interface, and reset the flag right after the data transfer is done. During the other time, while continuing to poll the flags, the SGI processes data using a multi-job mechanism. The whole spill data from the top and the bottom VME crate are first merged, then translated, and logged. Meanwhile, the TADPOLE will get ready for the next spill right after flagged by the SGI that the data transfer is done.

When a run is ended, the TADPOLE will stop data taking and set the END-RUN flag. After the SGI workstation polls the flag, it will close the run data file, update the run RDB database via the rsh(UNIX remote shell command), then reset the flag.
A run can be either stopped from the console, or ended by switching to a new run automatically. When a disk is full, it will be switched to the other disk automatically and a separate job will copy the run files from that disk to tapes.

The diagram of the data transfer during the data taking is shown in Fig. 2.5. (In this figure, VME BUS0 refers to the top VME crate; VME BUS1 refers to the bottom VME crate; and VME BUS2 refers to the newly added VME crate (crate 0).)

The diagram of the time relation in this system during data taking is shown is Fig. 2.6. As shown in this figure, for the CAMAC crate, the data from the digitizer modules are transferred to the SBE’s memory directly by memory mapping. For the FASTBUS crate, the segment manager first reads data from the digitizer module into its memory, then sends the data to the VME FIFO card through the optical link.

Please also refer to section 3 in the next chapter for some other discussions.
Figure 2.5: Data Flow in the E917 DAQ system (courtesy of ChuanMing Zou)
Figure 2.6: Time relation during Data taking
Chapter 3

Design development

In the development of the E917 data acquisition system, all the newly added hardware are commercial products, so no hardware design and development is involved. Generally speaking, the development work involved can be divided into four parts:

1. Implementation of BIT3 interface and the SGI workstation in the system.
2. Software development on the SGI workstation.
3. Software development in the VME crate, mainly on the TADPOLE.
4. Inclusion of the new detectors into the data stream.

Besides myself, Dr. ChuanMing Zou, Dr. James Chang, and the spokesman for this experiment Prof. Richard Seto (all of whom are from University of California at Riverside) are also involved in the development of this project. ChuanMing Zou did all the development work on the VME side; James Chang did the work to incorporate the new detectors such as BVER into the system and prepare its translation code. Richard Seto supervised this project and wrote the program on the SGI workstation to copy the run files from the disk to the tapes. In this thesis, I will mainly describe the implementation of the BIT3 interface and the work on the SGI workstation. For completeness, I may also talk about the work done by the others as necessary.

3.1 Development Environment, hardware and software

On the SGI workstation, ANSI C is the standard programming language. The CC compiler, make utility and xdbx debug tools supplied with the SGI IRIX5.3 operating system
are used in the software development. The test and implementation of the BIT3 interface were also done on the SGI.

The SGI Challenge S series workstation is a server machine, structurally the same as the SGI’s INDY series, except that the graphic interface along with unit 1 GIO bus interface’s DMA channel on the INDY are used here to supply the extra two differential fast wide SCSI and one narrow single ended SCSI interfaces. Because only one text console is directly connected to the machine, an X terminal linked to the Ethernet is used during the software development.

The 2 GB hard disk drive supplied with the machine is used to save the system software and the application program. Four 16MB memory chips are added so that the total internal memory space is increased to 128MB, which is the upper limit allowed by the system.

The development platform for the VME system, as well as the segment manager in the FASTBUS and CAMAC crate, is the SUN IPC workstation. The development software package is the VELOCITY VRTX32. It supplies the VELOCITY’s VRTX32 real time executive for MC68040 and MC68020 based VME single board computer, and also the tools to compile the application program, and download the code to the single board computer for execution or debugging if necessary. The program language for the VME system is mostly ANSI C, a little bit of assembly language is also used.

3.2 The application of BIT3 VME-GIO, VME-VME high speed DMA engine

3.2.1 Introduction to the BIT3 interface

The BIT3 607 VME-GIO adaptor is a high speed connection between the VME system and the SGI workstation on the bus level. It consists of three parts: a VME card, a GIO
card and a cable. An optional dual port memory daughter card can be plugged in the VME card and accessed from both the VME interface and the cable interface (the SGI side) simultaneously. No additional VME slot is needed. Up to 8MB memory space can be added. The function blocks of the BIT3 GIO-VME adaptor is shown in Fig. 3.1

![Diagram of BIT3 GIO-VME adaptor function blocks](image)

**Figure 3.1:** Diagram of the BIT3 GIO-VME adaptor function blocks

The BIT3 interface, which supports DMA (Direct Memory Access) and program I/O, can be selected by writing the control register on the adaptor card from either side (GIC or VME). In our application, both modes are used.

In DMA mode, in the case of transferring data from the SGI workstation to the VME system, on the GiO adaptor, the data will be sent first to its FIFO through the GIO interface. After one unit of the FIFO is full, the data which follows from the GIO interface will be put into the next FIFO unit. Simultaneously, the data in the full FIFO unit will be transferred to the VME card through the cable at high speed by DMA; whenever the data in a full FIFO is transferred, it will be released. In a similar way, the VME card receives the
data from the cable and transfers them to the VME memory: the data from the cable (SGI workstation) will be put first in the FIFO, and then transferred into the VME system’s memory by DMA when a FIFO unit is full, and so forth, until all the data has been transferred; the other way around is the same. The BIT3 interface supports the Block Mode DMA: in this mode it only increments the data address after transferring every 64 long words (one long word is 4 bytes) rather than every long word, thus it is faster than ordinary DMA.

On a workstation, the programmer deals only with the virtual memory space; the operating system supplies the mechanism of conversion between the virtual address and the actual physical address. On the GIO adaptor card, 16 page registers are used during the DMA transfer, so one DMA can transfer up to 64KB data (the page size on the SGI is 4KB). If the data size is larger, it has to be broken into several DMAs.

The memory mapping is utilized by program I/O using the BIT3 interface, in this mechanism, a section of the memory space on the SGI workstation is mapped to a section of memory space in the VME system. As a result, this portion of the VME memory can be accessed by the SGI workstation just like its own memory.

More detailed information about the BIT3 interface can be found in the manual from the manufacturer.

3.2.2 The application of the BIT3 interface
As mentioned in section 2.2.1 Hardware configuration, BIT3 is a high speed interconnection on the bus level between the workstation and the VME bus system, or between VME and VME system. By using BIT3, we can keep using the TADPOLEs, and thus keep the whole hardware structure in the old system. This will save our investment, reduce the amount of work in the development, and also make it much easier to switch back to the old
system if there is a need. Compared with the option of replacing the TADPOLEs with new single board computer, BIT3 is a safer and better choice.

In our application, we use the device driver supplied by the BIT3 company for its GIO-VME adaptor card. However, in our early efforts to apply the BIT3 interface to the new DAQ system, there were two major obstacles. First, in the original design, we were going to let the SGI read data directly from the VME crate into its memory during each event by using two BIT3 GIO-VME adaptors, with the two GIO adaptors plugged in the SGI Challenge S machine, and the two VME cards plugged in the top and the bottom VME crate respectively (refer to Fig. 2.1), and further process the data during the off-spill time. The TADPOLEs would just take care of the coordination with the SGI, the trigger supervisor and the SBE single board computers. But later we found that for the two GIO connectors on the SGI Challenge S machine, only unit 0 has the DMA capability, so the interface that plugged in unit 1 could not transfer data by DMA. Secondly, the BIT3 driver has about 350~500 microsecond overhead. For a large block of data transfer, such as 2MB and up, the overhead can be ignored. But in our application, there are about 10 partitions, in each of which the data size is about 50~4000 bytes. Even if the BIT3 interface could transfer data at 20MB/sec (actually it is not that fast in our application, with the overhead, the actual data transfer speed we can get is only 0.2~7 MBps), it definitely could not satisfy our requirement.

It took us about seven weeks (more than 1/3 of the time frame) to find a solution, however, only one section is used here to describe it.

To solve the problem that only one GIO connector on the SGI Challenge S machine can support DMA, we had many discussions with the SGI company and the DEC company. We thought about switching to a different platform, such as an Alpha workstation which supplies more than two PCI bus connectors that support DMA. Moreover, the driver
for the PCI-VME adaptor can be bought from the DEC company. Although from the discussion, we knew that it may also have the overhead problem, we still ordered the hardware and the driver (in both UNIX and VMS version), and started testing. Meanwhile, the information we got from the SGI company was contradictory, but more sources suggested that the SGI's INDY machine may support DMA on both GIO connectors. Then a SGI INDY machine was shipped from University of California. The test shows that it works. While this problem seemed solvable, the overhead problem still remained. On the other hand, it was also a dilemma as to whether to use the Alphastation or the INDY. Because a lot of work had been put into the SGI workstation, it was easy to keep working on it, but the INDY machine has only one narrow SCSI interface, and we need more; for the Alphastation, its CPU power and I/O capability were not a problem, but a lot of work that already had been done on the SGI machine had to be redone. Moreover, the data organization used on the Alpha machine is Little Endian System, and on the VME and SGI it is Big Endian System. So from the VME system to the Alphastation, the data has to be converted before being processed and then converted back again before data logging. The data logging was done in the VME crate and we wanted to follow the same data format as in the old system.

As to the overhead problem, one solution is to modify the driver for our needs. At first we hoped that we could talk the DEC company or the BIT3 company into modifying their driver according to our application and thus reduce the overhead. However, after asking a high price ($10,000~20,000), eventually they refused. Being desperate, we did some more tests such as writing the control register from the VME side to initiate the DMA. If this had worked, we could have minimized the overhead under our control. The test was not successful, but during the discussion with the BIT3 company, we learned that two BIT3 VME cards connected through a cable can be used to transfer data between two VME
memories. Through some tests, we confirmed that it works, and this finally solved the overhead problem.

BIT3 413-1 is the product number which refers to the configuration that uses two VME cards to transfer data at high speed between two VME memories. Its working mechanism is the same as the GIO-VME adaptor, except that now both cards are VME cards and the data transfer is initiated by a CPU in the VME crate. In the 413-1’s configuration, either of the VME cards can have a dual port memory daughter card, but only one is allowed in a pair at the same time.

By using BIT3’s 413-1, the overhead is decreased to about 55 microseconds (compared to 350), and 30 microseconds of this is the system overhead. We will discuss this in detail in the next chapter.

The system configuration is adjusted accordingly. Two BIT3 413-1s are introduced. In each of them, one VME card is put in the top or the bottom crate where the TADPOLE is, and the other (which has a 8MB dual port memory daughter card) is put in the VME crate 0 where the VME card of the BIT3 GIO-VME adaptor is also put.

However, in the final system we could not use the BIT3 413-1’s BLK Mode DMA to directly read data from the SBE and the FIFO memory cards which correspond to the FASTBUS crates. ChuanMin Zou did some tests and found out that the design of the FIFO memory card can not support BLK Mode DMA.

In the final application, we still let the TADPOLEs read event data from the SBEs and FIFO memory cards by directly using MC68040’s instruction MOVE16 which transfers 16 bytes at one time and is the fastest mechanism that we can use. The memory space on the TADPOLE, now freed from event processing, is used to buffer raw data during the onspill time only. When a spill is over, the two 413-1s will move the data from the TAD-
POLE to their dual port memory daughter cards. The dual port memory card has two advantages:

1. It is treated as an internal memory space by the 413-1 configuration and also can be accessed as a VME bus memory in the crate.

2. Transfer of data to the dual port memory through the cable does not use the VME bus. Therefore, the two 413-1 adaptors can transfer data in parallel without interfering with each other.

Because one GIO-VME adaptor is capable of transferring data from the dual port memory cards to the SGI workstation during the off-spill time, having only one DMA channel on the SGI challenge S machine is no longer a problem, and its extra SCSI interfaces become an advantage; along with the convenience of the Big Endian System that is also used in the VME system. These changes made it possible to use the SGI.

By default, the BIT3 driver requires root privilege if the default data transfer parameters need to be changed, therefore, we disabled this feature in the reinstalled driver, so that the DAQ program can be run by an ordinary user. The default maximum DMA size on the SGI machine which is 4MB is increased to 8MB by modifying the system file systune on the SGI.

In the E917 DAQ system, the data flow through the BIT3 413-1 VME-VME adaptors and 607 GIO-VME adaptor is shown in Fig. 3.2.
3.3 Realization of real time response in a UNIX environment

In a real time application, the job with a higher priority must be run first, and the job with a lower priority has to release the resources and be put to sleep when any higher priority job is waiting. The bottom line is that a request from the application has to be processed within a certain time limit.

UNIX is the operating system on most workstation products, but it is designed for a multi-user environment, not for a real time application. If a job in a UNIX environment has higher priority, as defined here, it means that the length of the allowed time slice is larger. A job with lower priority does not give way to a higher priority one during its time slice. In our application, real time performance is critical. We require that the SGI workstation start transferring data from the VME system to its memory as soon as it is notified. Interrupt, signal and queue are the common mechanisms to implement a real time application. Although they are supplied by the UNIX system, it is not guaranteed that they can be
processed within a certain time limit, and the maximum latency is also too long, about 200 microsecond or even longer.

After some research, the flag exchange mechanism was developed to realize real time response in our application. It involved defining a group of flags in which each flag refers to one specific case and may infer a certain priority. A flag is set by one side (mostly the TADPOLE) and reset by the other. On the SGI workstation, the main program keeps looping and polling the flags all the time whenever there is a chance. If a flag is found set, the main program will create a child process immediately to perform a specific action, reset the flag to inform the other side, and then continue the regular routine. The main loop is always running on the CPU, and all the other application processes are directly started by the main loop rather than the operating system. This avoids the latency caused by the operating system and makes sure that a flag will be taken care of in time whenever it is set, thus guaranteeing real time performance.

The mechanism of the flag exchange between the SGI workstation and the VME system is implemented by the memory mapping through the BIT3 interface. One section of the memory space on the dual part memory card is mapped to the SGI workstation's memory and TADPOLE's memory space at the same time via the BIT3 GIO-VME adaptor and 413-1 adaptor respectively. The memory mapping setting on both dual port memory cards are identical.

The flags used in the application and their explanations are listed in Table 3.1.
### Table 3.1: Flags used in E917 DAQ system

<table>
<thead>
<tr>
<th>NAME</th>
<th>SET/RESET</th>
<th>TYPE</th>
<th>EXPLANATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>NEW_RUN</td>
<td>TADPOLE/SGI</td>
<td>Byte</td>
<td>Start a new run</td>
</tr>
<tr>
<td>END_RUN</td>
<td>TADPOLE/SGI</td>
<td>Byte</td>
<td>End a run</td>
</tr>
<tr>
<td>SPILL</td>
<td>TADPOLE/SGI</td>
<td>Byte</td>
<td>Data for last spill is ready to be transferred</td>
</tr>
<tr>
<td>SWITCH_RUN</td>
<td>SGI/TADPOLE</td>
<td>Byte</td>
<td>Automatically switch to a new run</td>
</tr>
</tbody>
</table>

Besides the flags, there are some other data exchanged between the SGI workstation and the VME system via the memory mapping. They are all written by the TADPOLE and read by the SGI.

### Table 3.2: Data exchanged between the SGI and the VME via memory mapping

<table>
<thead>
<tr>
<th>Item</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1 Long Word</td>
<td>Run number</td>
</tr>
<tr>
<td>2</td>
<td>1 Long Word</td>
<td>Length of data (in byte)</td>
</tr>
<tr>
<td>3</td>
<td>1 Long Word</td>
<td>Length of circular buffer</td>
</tr>
<tr>
<td>4</td>
<td>8 Words</td>
<td>Drop FASTBUS bank if set</td>
</tr>
<tr>
<td>5</td>
<td>1 Word</td>
<td>Broadcasting enable if set</td>
</tr>
<tr>
<td>6</td>
<td>1 Word</td>
<td>Broadcasting rate in percentage</td>
</tr>
<tr>
<td>7</td>
<td>Long Word</td>
<td>Reserved</td>
</tr>
</tbody>
</table>
The implementation of the flag exchange mechanism on the SGI workstation is shown in Fig. 3.3. Note that some details of the main program on the SGI are not shown in this figure.
Figure 3.3: Block flow of the main program on the SGI workstation
3.4 New data processing mechanism: data merging and event translation on SGI workstation.

After data from each CAMAC or FASTBUS crate is read into the TADPOLE’s memory, they will be packed in the YBOS bank format, as shown in Fig. 3.4

<table>
<thead>
<tr>
<th>Long Word</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bank size in LW(exclude itself)</td>
</tr>
<tr>
<td>Bank Name</td>
</tr>
<tr>
<td>Bank Number</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>Bank Type</td>
</tr>
<tr>
<td>Data</td>
</tr>
</tbody>
</table>

**Figure 3.4:** YBOS bank format

In the old system, the TADPOLE’s memory was divided into four parts: raw data buffer, translation buffer, tape buffer and broadcasting buffer. After one event is read in, the translation starts immediately, and data taking will not resume until the event processing is completed. When a spill is off, the data buffered in the lower TADPOLE will be transferred through the crate interconnection to the top TADPOLE, merged, logged onto two tape drives, and broadcasted as a certain percentage if requested.

In the new DAQ system, the TADPOLE’s memory is only used to buffer the raw data, and the data taking will resume right after the event is read in. When a spill is off, the spill
data buffered in the two dual port memories will be moved into the SGI workstation, first logically merged, then translated, and eventually logged on to disk and tapes.

From the SGI workstation to the VME system, two sets of data are transferred via the BIT3 interface:

1. the circular buffer block.
2. the data block.

The circular buffer is a C structure defined to hold event information, such as event number, event type, number of fragments, whether the fragment is from a FASTBUS crate, fragment size, and the starting address of the fragment, etc. Some information is used for the buffer circulation, and it is still defined, but not used any more.

On the SGI workstation, two identical buffer areas are defined (total size is about 52MB), and the data for one spill will be put in the two sets of buffers alternatively. The program on the SGI supplies the mechanism for buffer switching and interlocking to make sure the data in the buffer will not be corrupted by accident.

The memory map of the buffer and the size of each section (in bytes) is shown in Fig.3.5 and Table 3.3 respectively.

<table>
<thead>
<tr>
<th>Block Name</th>
<th>Size</th>
<th>Block Name</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIZ_OF_SGI_BUF_TAG</td>
<td>16 byte</td>
<td>SIZ_OF_RAW_CIRC_BUF</td>
<td>196KB</td>
</tr>
<tr>
<td>SIZ_OF_BT_DPRAM</td>
<td>8MB</td>
<td>SIZ_OF_SGI_PAGE</td>
<td>4KB</td>
</tr>
<tr>
<td>SIZ_OF_SGI_FFT_BUF</td>
<td>10MB</td>
<td>SIZ_OF_SGI_BC_BUF</td>
<td>1MB</td>
</tr>
</tbody>
</table>

Table 3.3: Segment size in the event processing buffer on the SGI
**Figure 3.5:** Memory map of the buffer for event processing on the SGI

### 3.4.1 Event Merging

For a regular event, both the top and the bottom TADPOLE will have a record with the same event number, but for a special event (such as PAUSE, NEW_RUN, END_RUN, Start_of_spill and End_of_spill), only the top TADPOLE has a record. When the spill is
off, the spill data and the circular buffer from both VME crates will be transferred to the address starting from SGI_RAW_CIRC_BUF1, SGI_RAW_BANK1, SGI_RAW_CIRC_BUF2 and SGI_RAW_BANK2 respectively (refer to Fig. 3.5) via the BIT3 interface. Then the merging program will loop over the circular buffer from the top VME crate; if it is a special event, only the fragment pointer will be adjusted to point to the correct address in the SGI's memory, if it is a regular event, the information in the circular buffer from the bottom VME crate relevant to this event will be inserted in the top circular buffer, and then all the fragment pointers will be adjusted, and so forth, until the end of spill mark is reached. The whole operation is only on the circular buffer. No data is moved or copied.

In the old system, there are two types of circular buffers: the raw circular buffer and the tape circular buffer. The raw circular buffer was used for translation. The data merging is based on the tape circular buffer which is not used anymore in the new system.

3.4.2 Event translation.

In both the old and the new DAQ system, only the FASTBUS data is translated.

The main body of the event translation program on the SGI workstation is transplanted from the code that runs on the TADPOLE in the old system, and some code is added to translate the data from the new detectors such as BVER. As to the latter part, James Chang did the work such as modifying the transload.sav file (which contains the ADC readout information), and the translation table reading program to reflect the BVER detector. He also wrote the translation code for this partition which I eventually incorporate into the translation program.

Because the SGI workstation physically is a different machine, some modification had to be done. In the TADPOLE's translation program, it was necessary to get some variables
from other modules, such as the event building program, and some of its buffer and table memory used in the translation was directly mapped to the physical address in the VME system. The translation code consists of seven separate C program files.

The transplant of the translation code went through three steps:

1. Study the TADPOLE’s translation code, cut off its connection with the event building and other modules, remove its dependency on the physical memory space in the VME system, and modify the program accordingly.

2. Write a test translation program which reads untranslated events from an old run file(on the disk) and outputs the results to a disk file on the SGI workstation, so that we can check the translation.

3. Incorporate the working new translation code into the program on the SGI workstation.

In the new DAQ system all the events are translated.

3.5 Multi level buffered data logging

In the old DAQ system, the data were directly logged onto the tapes. According to the manual, the speed of the tape drive can be up to 500KB/sec which is the performance for large data size. If the data size is small, the frequent mechanical start and stop of the tape drive will severely lower the tape speed, increase the chance of writing error, and thus make the data taking process less stable. Large data size and fewer mechanical pauses will make the data logging faster(thus closer to the manual speed) and smoother. In our application, the data of a whole spill is written first into a run file on the disk which has a higher speed, a much better dynamic mechanical characteristic, and can handle the data flow during the data logging more easily than the tape drives. At the second level, after the disk is full, all the files on the disk will be copied to the tapes at once.
In this mechanism, a basic requirement is having the disk drive faster than the average data flow of the DAQ system, and the overall tape copying speed faster than the disk writing speed. This insures that the data will not pile up at one section, and that the computer memory or the disk space can be released in time for the next set of data.

In a run file, the data is further packed into a YBOS record which is a fixed length record(7680 Long Word). There are three of them:

1. start record, i.e. the first one in a file.
2. regular record.
3. end record, i.e. the last record in a file.

The format of the YBOS records are shown in Fig. 3.6 and Fig.3.7

![Figure 3.6: Format of the first YBOS record](image)
Figure 3.7: Format of a regular YBOS record

The format of the last YBOS record is almost the same as the regular one, except that after the last event in the run file, all the remaining long words in the record are filled with the value -1.

In the old DAQ, a processed event was copied to the tape buffer and then packed in a YBOS record; now it is done directly in the translation buffer, making the event processing more efficient.

3.6 multi job mechanism

On the SGI workstation, the data transfer via the BIT3 interface and the tape or disk writing use very little CPU, but take quite some time to finish. To enhance the CPU efficiency and guarantee that the event processing can be finished in time, a multi-job mechanism is necessary.
The data transfer and data processing activities on the SGI workstation are basically controlled by the main loop program. The idea is that after data have been processed, they are handed over to a disk writing job. The main program keeps its regular looping. Similarly, while data are transferred via the BIT3 interface, if there are any unprocessed data, they are passed to an event processing job and then sent to a disk writing job.

The regular way to realize a multi-job mechanism is via signal, message, and shared memory between the jobs. When it is necessary, the main loop wakes up a specific job (to write the disk or process the data) by sending a message or a flag; then lets it go back to sleep after finishing its assignment.

SGI's IRIX operating system supplies a system function called sproc(), which creates a child process which can share the virtual memory space and the file table with its parent. This makes our life much easier. By using sproc() utility, the shared memory is automatically setup. Whenever there is a need, a child process will be created to do a specific job, and then dies by itself after finishing. No signal or message is necessary, and the main program can just keep looping.

One technical problem solved in this application is that after the child process is dead, the system resources such as the stack space or whatever will not be returned to the system automatically. If we keep creating child processes, the system resources will soon be used up and no more processes can be created. The solution is the following: in the main loop, before creating a specific child process, use the waitpid() function to make sure that the previously created child is finished. This call will also notify the system to reclaim the resources possessed by the defunct process.

In our application, there are two types of child process:

1. Disk writing. This is used in both run data logging and broadcasting.
2. Event process. This will also create its own disk writing and broadcasting child processes after finishing the event process.

### 3.7 Event broadcasting, accessing database and others

In the old system, the event to be broadcasted is put in a buffer, and sent to a VAX machine over the Ethernet, and another program on the same VAX machine will accept the event from the Ethernet and put it in an event pool. The other on-line monitor programs can take the event from there.

In the current system, the broadcasting is done differently. After an event is processed, if it is to be broadcasted, a copy of this event will be put in the broadcasting buffer and after each spill written to a broadcasting file on a cross mount disk on an Alphastation. Except when the file is written on a cross mount disk, it is treated the same as a regular run file. The idea is that the disk file is more secure than the event pool whose mechanism is more complicated and relies heavily on the networking.

Each time a new run is started the .TS_config file (which contains the information about what partitions are in the data stream) and the transload.sav file (which contains the translation table information) will be copied from the SUN workstation to the SGI, and read by the main program to form the translation order and translation table. After a run is ended, the run information will be written into the RDB by calling a rsh from the SGI.

On the SGI workstation the tape copying is a separate job from the main loop program and it was written by Prof. Richard Seto. When a disk is full, the main program will create an empty file to indicate which area is full, the tape copying program can check the existence of this file and decide its further actions. Now the tape copying job is started by the shift people, it will instruct the shift people to mount the tapes, copy the files in the area.
As a precaution, both the flag files and the run files are now deleted manually by the shift people after files are copied.
Chapter 4

Performance measurements

4.1 Performance measurements

In the upgrade system, during the on-spill time, the event data are buffered on the TADPOLEs’ memory. During the off-spill time, the buffered data are transferred to two dual port memory daughter cards in crate 0 via BLK Mode DMA by two BIT3 413-1 VME-VME adaptors respectively. Then the TADPOLEs will flag the SGI to transfer data from the two dual port memory cards into its own memory. After the data are transferred into the SGI, they will be merged, translated and logged on disk. Eventually the data files will be copied onto tapes.

In this section we will discuss the performance measurements of the different function blocks in the new DAQ system, mainly on the SGI workstation, such as the data transfer speed of the BIT3 interfaces (including both 413-1 VME-VME adaptor and 617 GIO-VME adaptor), flag exchange speed, data merging speed, event translation speed and disk logging speed. Later on, we will estimate the performance of the upgraded system based on the measurements performed.

4.1.1 Performance of the BIT3 interface

Before transferring data via the BIT3 GIO-VME interface, the lseek() function needs to be called on the SGI workstation every time to set the VME start address for the driver. Its execution time, as determined by measurement, is shown in Fig. 4.1:
**Figure 4.1**: Speed measurement of `lseek()` in BIT3 driver (1000,000 loops)

In this measurement, several consecutive `lseek()` calls were made. This was repeated for 1000,000 times, and the total execution time was recorded.

From fitting, we determined that each `lseek()`'s execution time on the SGI is 3.13 microseconds.

Similarly, we measured the data transferring speed via the GIO-VME interface with the BIT3’s device driver. The data is shown in Fig. 4.2 (for 1000 loops).
**Figure 4.2:** Distribution of GIO-VME reading speed measurement data (1000 loops)

During this measurement, the key device driver parameters are:

1. Lowest trace level (only fatal errors are reported.)
2. Block DMA mode
3. PIO data width 32 bit (4 bytes)
4. DMA threshold 256 bytes
5. DMA poll size 2KB
6. During Block mode DMA, transfer only pauses at every 256 bytes
7. Privilege checking disabled

When data size is less than 256 bytes, it is transferred by PIO (program I/O). The measurement data (for 1000 loops) is shown as the first four points in Fig. 4.3; the data after that behaves differently and belongs to a different case, as is explained in the next paragraph.
Figure 4.3: Measurement of PIO speed of BIT3 interface (1000 loops)
From fitting(dot line and triangle marks in the figure), the data transfer speed via PIO is 0.83 microsecond/byte or 1.2MB/sec; the driver's overhead is 266 microsecond.

When the data size is between 256 and 2K bytes, DMA is used to transfer data, and the DMA-Done is polled to signify the end of the data transfer. The measurement data in this case is shown as the first four points in Fig. 4.4 (Other data appearing in this figure will be explained in the next paragraph). According to the fitting, the data transfer speed in this case is: 6.55MB/sec, and the driver's overhead is: 319 microseconds.

When the data size is larger than 2KB, the interrupt is used to inform the CPU when the DMA is finished. The data (for 1000 loops) as measured is shown in Fig. 4.5. From fitting, in this case, the data transfer speed is: 8.65MB/sec, the driver's overhead is 836 microseconds.
Figure 4.4: Measurement of speed of DMA with poll (1000 loops)

Figure 4.5: measurement of DMA speed with interrupt (1000 loops)
The BIT3 adaptor’s data transfer speed has a strong dependence on both the access speed of the VME memory and the load in the VME system. The specification is 26MB/sec with a 20ns memory. In our system, the VME memory is slower. The transfer speeds as measured individually from the TADPOLE’s, the SBE’s, and the dual port memory to the SGI are a little bit different, but all are close to 10MB/sec.

One factor which contributes considerably to the BIT3 driver’s overhead is the switching time between the system mode and the user mode. One DMA can only transfer as much as 64KB, and the data transfer pauses every 256 bytes in order to let other parties in the VME crate have a chance to use the bus. Thus, a bigger data size means more DMAs and more pauses. That is why the overhead becomes bigger with a larger data size.

4.1.2 Memory mapping access speed via the GIO-VME adaptor

The measurement is shown in Fig. 4.6 (loop=1000).

![Speed of memory mapping via GIO-VME adaptor](image)

**Figure 4.6:** Measurement of memory mapping speed of GIO-VME adaptor (1000 loops)
The measurement data are almost on a perfect straight line. According to the fitting, the map accessing speed is 2.8 microsecond per access (a long word), or about 1.4 MB/sec (the specification is 2.0 microsecond per access with a 20ns memory).

The memory mapping is implemented by the mapping mechanism of the operating system on the SGI, and there is no overhead during the access.

4.1.3 Data transfer speed of the BIT3 413-1 VME-VME adaptor

The measurement is shown in Fig. 4.7 (loop=1000,000, uses DMA Done polling).

![Measurement of the BIT3 413-1 speed](image)

**Figure 4.7:** Measurement of the BIT3 413-1 adaptor data transfer speed (1000,000 loops)
The measurement data show good linearity. After the fitting, as measured, the 413-1’s data transfer speed is 8.35 MB/sec (From the TADPOLE’s memory to 413-1’s dual port memory). and the overhead is 55.9 microsecond.

The specification data transfer speed is 34 MB/sec with 20ns VME memory. But the speed to/from the BIT3’s dual port memory card is much slower, and the performance is not listed in the specification. On the other hand, our VME memory is also slower. We tried unsuccessfully to persuade the BIT3 Corporation to improve the circuit design.

In the overhead, about 25 microseconds are used for control register writing. The remaining 31 microseconds are explained as follows: if the data size is only 256 bytes, the data will be transferred to the FIFO on the first VME card, then to the FIFO on the other VME card through the cable, and eventually to the other VME memory; those sequences are taken one by one and are not parallel. If the data size is bigger than 256 bytes, starting from the second 256 bytes, both FIFOs on the two VME cards will be transferring data to/from its VME memory at the same time, and the overall transfer speed will be higher. So this 31 microsecond overhead comes from the extra time used to transfer the first 256 bytes, and it is ‘built-in’.

4.1.4 Disk writing speed.

In our system, the disk is hooked up to a fast wide differential SCSI interface with a specification speed of 20 MB/sec. In our application, we use fwrite()/buffered write) to write to the disk. In the measurement data (as shown in Fig. 4.8, the time measured here is real time, no loop is involved), before the sharp rise in the graph, the linearity is pretty good. From fitting, the disk writing speed in our system is: 19.2 MB/sec, very close to the specification.
Figure 4.8: Measurement of disk writing speed

Because we write to disk only once every spill (about 4 second), the overhead of the disk write (about 436 microseconds) is not important.

One explanation for the sharp rise in the measurement data shown in Fig. 4.8 is that the buffer size of the system for the fwrite() is about 7 MB. When the data size is larger than that there are some extra system expenses. As a result, the write takes a longer time.

4.1.5 Event processing speed

The measurement in this section is made under real run conditions.

1. Event merging

The time spent on the event merging depends on the number of partitions in the data stream. The measurement data with all the partitions included is shown in Fig. 4.9.
Figure 4.9: Measurement of the event merging speed

After fitting, the merging time is **5.80** microsecond per event.

2. Event translation

The event translation is more complicated. It depends on many factors: the partition numbers in the data stream, the individual partition, its size, and whether one wants to drop banks, etc. Here we show the data with all (FASTBUS) banks dropped and with all partitions in the data stream. This gives an idea about the system’s performance. The event per spill in each case is 155, 320 and 723.
According to the fitting, the translated data rate coming out of the translation program on the SGI is **8.9 MB/sec**. Considering that the translated event size is about 60% of the raw event, the translation processing speed on the SGI is around **14 MB/sec**. The other system expense (i.e. overhead) during the event translation is about **0.015** second.

**4.1.6 Performance estimate**

The flag polling is implemented via the mapped memory through the GIO-VME interface. Its access speed is 2.8 microseconds (i.e. 357 KHz), much higher than our possible event rate, so there is no problem here. The off-spill time, which is about 1.6 sec, is the only time window for the SGI workstation to transfer the raw data from the VME system into its own memory. According to our measurement, during each off-spill period, the
maximum size of the raw data that can be transferred is about **14 MB**. The system’s event processing and disk writing capability is far beyond this. Considering that each raw event is about 10-14KB, the upper limit of the system’s performance could be 900-1000 event per spill.

### 4.2 Run parameter setting

1. To speed up the data transfer, the BIT3 GIO-VME device driver is configured as:

   - **TRACE_LVL** -- **BT_TRC_ERROR**, only the error is reported.
   - **DMA_ADDR_MOD** -- **VME_A32_NB**, use Block Mode DMA, the address space is A32.
   - **PIO_ADDR_MOD** -- **VME_A32_NP**, the PIO address space is also A32.
   - **THRESHOLD** -- 256, data size bigger than 256 bytes will be transferred by DMA.
   - **DMA_PAUSE** -- **FALSE**, disable the pause every 64 byte during the DMA.
   - **DATA32_SIZ** is set, i.e. data is transferred by 32 bits.
   - **DMA_POLL_SIZE** -- 256, i.e. the DMA Done is notified by interrupt.
   - Privilege checking disabled.
   - Memory mapping, protection set as **PROT_READ | PROT_WRITE**, i.e. both read and write are allowed.
   - The detailed BIT3 interface related address setting can be found on sgiuc0, /usr/people/daq/yao/tp40v/FFT/inc/bit3.h

2. **RUN_LIMIT** 0x1300000 (i.e. 300M Long Word, or 1.25GB). When the RUN size is larger than this limit, the program will beep every spill thereafter to remind the shift people to end the run and start a new one. Meanwhile, it will set a flag to inform the TADPOLE to switch to a new run. But, it is up to the TADPOLE as to decide what to do.

3. **DISK_LIMIT** 0x68a0000 (i.e. 1.75G Long Word, or 7 GB). The difference between this number and the disk size is a little bit more than the **RUN_LIMIT**. This is to guarantee that a disk always has enough disk space for the last run. When the last run is ended, the disk writing will be automatically switched to another disk. There are two disks, and each has two directories. When one disk is switched back, it will go to the
directory not used the time before. This is to avoid conflict when the data in the other directory having not been or currently being copied.
Chapter 5

Conclusions

5.1 Performance under the real run conditions

Under real run conditions, the new system works well. Compared with the old system, it runs smoother, and is more stable. The event per spill in the current system can reach \textbf{927}. (Spill length: 4.6 sec, on-spill 3 sec, and part of the PHOS ADCs are taken out of the data stream), almost three times as much as the old system. In this design, the limits of the SGI workstation and the BIT3 interface have not been reached yet. On the VME side, the memory on the TADPOLE is the factor that limits the event per spill. In general, the performance of the new system is better than we expected. Therefore, the upgrade of the E917 DAQ system is successful.

The coordination between the SGI workstation, the SUN workstation and the VME system works well. Two window type interfaces are implemented to supply the online run information and do the tape copying. Now the shift people only need to do tape copying about every five or six hours at most after the disk is full.

The object of the new broadcasting is to supply an easier and more reliable mechanism than the event pool, which heavily relies on the networking. But in practice, when the broadcasting file is opened by the SGI workstation (which happens only when the SGI needs to write data into the file), the monitor program on the Alphastation cannot open the file on the cross mounted disk at the same time; otherwise there will be a conflict. However, when the program on the SGI workstation is not writing, it is OK. On the other hand, unlike with the event pool, the on-line monitor can not keep updating after the end of the file is reached. Generally, the broadcasting needs improvement but it works.
The collaboration is satisfied with the performance of the other parts of the new DAQ system.

5.2 Suggestions

The bottle necks in the current system are:

1. On the TADPOLE, the memory space is 8MB, and about 5-6Mb can be used to buffer the raw data. If there were more memory, we could have more events per spill.

2. Because the design of the FIFO memory card does not support BLK Mode DMA, we have to keep using MC68040’s instruction MOVE16 to read data from the front-end. Its speed is about 6MB/sec, and with the top and bottom crate running independently, the total speed is 12 MB/sec. The BIT3 VME card has about 30 microsec build-in overhead during the DMA, it is still too much for small size data transfer. The ideal solution is to directly use Block Mode DMA with low overhead (less than 10 microsec). That will speed up the DAQ system much more.

3. The speed of the BIT3 interface is only about 10 MB/sec in our system, this is because our VME memory is slow. By using a faster VME memory, the speed can be doubled.

As we discussed in the previous chapters, the BIT3’s dual port memory daughter card has a lot of advantages, but for some reason, possibly because of its current design or the speed of the dual port memory chips used on the daughter card, it achieved only about 10 MB/sec. Perhaps the BIT3 company can improve this in the future.

BIT3 is not the only option available for transferring data from the VME system to the workstation. The Lincoln Lab of MIT is using a commercial product that can transfer data at 70MB/sec between the VME system and the main frame, it can be used with SGI Chal-
lenge DM series, and is more expensive. Unfortunately, we did not have the time to do some detailed research on that.

Computer and electronics technology develops rapidly. The single board computer and the workstation are now much faster and more powerful. In some ways, the boundary between them is less obvious than before. For example, a single board computer can also have an on-board hard disk and fast wide SCSI interface. Moreover, for the I/O interface, IBM has introduced a more powerful standard SCS which can transfer data at 80 MB/sec via a cable with just 4 wires. As to the storage media, the 9GB hard disk will soon be history, and the newer product will be 20GB or higher. Higher speed tape drives with large storage are also available now. As the network becomes essential, and is faster and more reliable, new display technologies such as LCD are also becoming more and more popular. Moreover, the window system, the distributed data base, and new software development tools also develop rapidly. This will effect how people design their DAQ system in the future.

The basic elements in a DAQ system are still about the same, but there are more and more choices. As to which to choose and how to configure one’s own system, it should really depend on individual situations, according to your time limit, man power and funding.
Appendix A

Glossary

- 413-1, VME-VME bus adaptor manufactured by the BIT3 Cooperation.
- 607, VME-GIO bus adaptor manufactured by the BIT3 Cooperation.
- 963, device driver for 607 VME-GIO adaptor supplied by the BIT3 Cooperation.
- ADC, Analog to Digital convertor
- BLK Mode DMA, refers to Block Mode DMA in VME bus specification.
- Big Endian System, term of computer memory organization, in which the highest order byte of a word or a long word starts from the smallest address.
- CAMAC, a bus standard that is often used in heavy ion physics experiment.
- DAQ, stands for Data Acquisition.
- DMA, stands for Direct Memory Access.
- FASTBUS, a bus standard that is often used in heavy ion physics experiment.
- FIFO, stands for First In First Out.
- GIO, a bus standard used on SGI’s INDY and Challenge series workstations.
- GB, Gega bytes.
- IRIX, SGI’s UNIX version that runs on its workstation product.
- KB, kilobytes
- Little Endian System, term of computer memory organization, in which the highest order byte of a word or a long word starts from the biggest address
- LW, stands for Long Word (one long word is four bytes).
- MC68020, MC68040, CPU chips that are produced by Motorola Cooperation.
- MB, a million bytes.
- Mb, a million bits.
- PCI, a bus standard that is used on PC and Alphastation.
- PI0, stands for program I/O.
- RDB, a relation database software that runs on a VMS machine.
- SCSI, a interface standard that is often used on tape drives and disk drives.
- SBC, stands for Single Board Computer.
- SBE, the single board computer that is in charge of FIFO or a CAMAC crate in our VME system.
- TADPOLE, the single board computer in the VME crate that in charge of data taking and run control.
- VME, a bus standard designed for interconnection between single board computers.
- VRTX32, a real time executive that run on a single board computer.
- Word, one word is two bytes.
- YBOS, a specification of data organization that is often used in physics experiment.
Appendix B

Acknowledgment.

First I will thank my wife, she supports me with all she can, especially during the time I was working on my thesis project down in Brookhaven National Lab, NY, she took care of my son and the family all by herself.

I am also grateful to my advisor, George Stephans. During the years I was with the group, he always tried his best to help me and offer me guidance and advice. I will also like to thank Craig Ogilvie, he is so kind to be the reader of my thesis, from the past years we worked together, he taught me a lot.

Also thank the U.C. Riverside group, especially Richard Seto, he trusted me and supported me to the end, and made this thesis project happen. Also thank Chuanming Zou and James Chang, we worked hard and closely side by side on this project as a wonderful team in the past six months, and we succeeded.

Thanks to Marge Neal, she is so nice and so kind. I still remember she invited my family to her house in last summer, it made me feel very warm.

Also thank all the people in the group, Steve Steadman, Mark Baker, Larry Ahle, George Heintzelman, James Dunlop and Barb Corbisier. They all helped me a lot in the research, in the experiment and when I prepared for all the three qualifiers during the past few years.

Finally, I thank MIT, I learned a lot here.
References

[4] BIT3 Model 963 GIO-VME adaptor driver For IRIX Manual, BIT3 Corporation
[13] AlphaStation 500 Series User Information, Digital Equipment Corporation
FIXED FIELD

ill name

index biblio

COPYES

Archives Aero Dewey Eng Hum

Lindgren Music Rotch Science

TITLE VARIES □

NAME VARIES □ in degree book "VME" is replaced by "VMZ"

IMPRINT (COPYRIGHT) ______

COLLATION 80p

ADD DEGREE □ DEPT □

SUPERVISORS ______________________
______________________________
______________________________
______________________________
______________________________
______________________________
______________________________
______________________________

NOTES

cat'r date:

DEPT Phy

YEAR 1997 DEGREE M.S.

NAME YAC, Hongbin