The Linux Based Distributed Data Acquisition System for the ISTRA+ Experiment

A.Filin, A.Inyakin, V.Novikov, V.Obraztsov, N.Smirnov, E.Vlassov, O.Yuschenko
(IHEP, Protvino, Russia)

Abstract

The DAQ hardware of the ISTRA+ experiment consists of the VME system crate that contains two PCI-VME bridges interfacing two PC's with VME, external interrupts receiver, the readout controller for dedicated front-end electronics, the readout controller buffer memory module, the VME-CAMAC interface, and additional control modules. The DAQ computing consist of 6 PC's running the Linux operating system and linked into LAN. The first PC serves the external interrupts and acquires the data from front-end electronics. The second one is the slow control computer. The remaining PC's host the monitoring and data analysis software.

The Linux based DAQ software provides the external interrupts processing, the data acquisition, recording, and distribution between monitoring and data analysis tasks running at DAQ PC's. The monitoring programs are based on two packages for data visualization: home-written one and the ROOT system. MySQL is used as a DAQ database.

Keywords: ISTRA, DAQ, Linux, VME

1 Introduction

The ISTRA+ experiment being the development of ISTRA-M [1] started in 1999 at IHEP, Protvino. Its aim is the study of rare (on the level of $10^{-6} \div 10^{-8}$) decays of charged light mesons, such as $K \to e\nu\gamma$, $K \to e\nu\pi^0$, $\pi \to e\nu\gamma$ etc.

2 DAQ Hardware

The DAQ hardware configuration is shown on Fig. 1. It's built around the VME system crate. That crate contains two PCI-VME bridges interfacing two PC's with VME, the interrupt module, the readout controller for dedicated front-end electronics, the memory module for readout controller data buffering, the VME-CAMAC interface, and interface for front-end electronics control link.

The PCI-VME bridge (SBS-BiT3 mod.617 [2]) provides the mapping of the PCI bus address space to the VME bus address space giving the transparent access from PC to VME bus locations as locations in its own memory with 2.2 μsec access time. It has a built-in DMA controller that can copy block of data from VME modules to PC memory with speed up to 0.4 μsec per word. Additionally the bridge can transfer selected VME backplane interrupts into PCI bus interrupts that can be served by CPU.

The interrupt module (CES RCB CORBO 8047 [3]) transfers external interrupts into VME backplane interrupts IRQ1 – IRQ7. It has 4 independent channels each of those has the trigger pulse input generating the VME interrupts and the BUSY level output that is set by trigger pulse and reseted by software. Each channel might be assigned to the own interrupt level. The DAQ uses three interrupt channels for START_OF_BURST, END_OF_BURST and EVENT interrupts generation.

The readout controller is started by event trigger pulse, autonomously reads the data from dedicated front-end electronics and writes those through VME bus into the buffer memory module with speed up to 0.6 μsec per 4 bytes word. It is activated by DAQ computer at START_OF_BURST time and deactivated at END_OF_BURST time. There is a test readout mode when controller isn't activated and data readout is started by EVENT interrupt and done by software directly using the VME front-end control.

We use the software driver developed by N.Kruszynska at NIKHEF [4] with slight modifications. We decided that in our case the driver has to: (i) establish the required PCI-VME mapping and (ii) reschedule the interrupt servicing process when interrupt occurs. All VME transactions have to be done not by the driver but by that process using the memory mapping. Technically the interrupt servicing process setups the mapping by the "mmap" system directive and goes to read the status of VME bridge by the "read" system directive. When an interrupt arrives the process is rescheduled and runs having a single interrupt status word. Finishing the interrupt processing the process resets the proper BUSY level at the interrupt module and again goes to read status. The implemented interrupt processing scheme provides a reasonable event readout time: in the interrupt mode it is $\sim 30\mu sec + 2.2\mu sec * NUMBER\_OF\_DATAWORDS$ at Pentium150 PC measured as length of BUSY level by oscilloscope.

The DAQ uses the set of 6 PC's running the Linux operating system and linked into LAN by 100 MBits Ethernet. Two of those are equipped by PCI-VME bridges and work as primary DAQ computers. The first serves the external interrupts and acquires the data from front-end electronics. The second one is the slow control computer that executes such tasks as trigger logic control, beam-line magnets control etc. The remaining computers work as the secondary DAQ computers that host the control, monitoring and data analysis software.

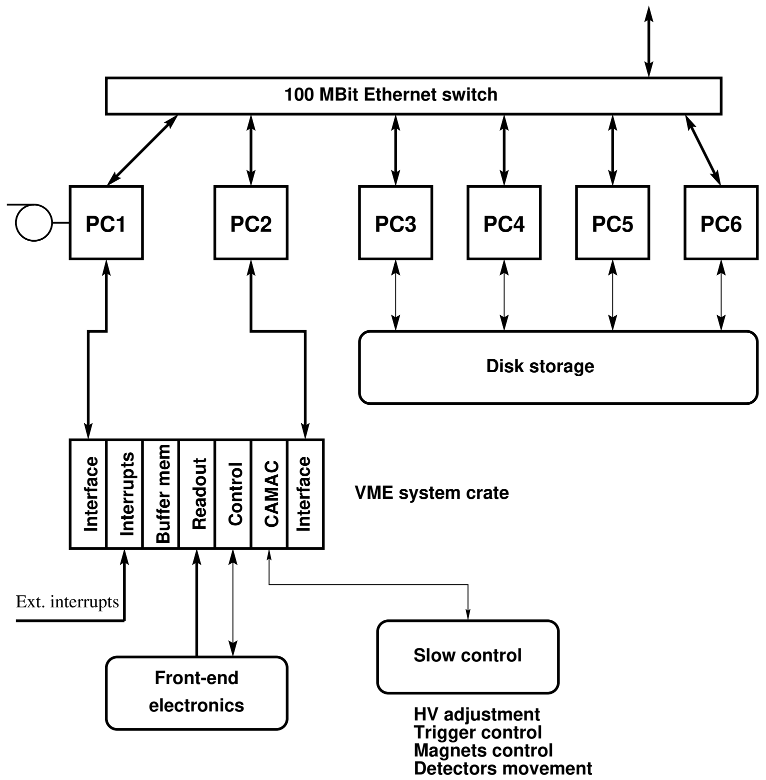

Figure 1: ISTRA+ DAQ hardware.

3 DAQ Software Architecture

The following requirements were considered during the system design:

- provide maximum possible event transfer rate between production and recording processes;
- make the system control easy to use by operator who is not a DAQ expert;
- provide a possibility to add dynamically monitoring programs ensuring that external monitor programs could not affect the main dataflow;
- make the code as much independent as possible from the event format;
- make it easy to reconfigurate the system.

The system should work under Linux OS and has been written in C++.

The requirement to maximize the event transfer rate to the recording process is the most important one. Less priority tasks should not disturb this one by taking system resources. Consequently the system architecture is distributed. It involves one computer (MAIN) for the main task (data taking and recording), one for the system control, one for monitoring daemon. User's monitoring programs can connect to the monitoring daemon either locally or remotely.

Typical configuration is shown on Fig. 2.

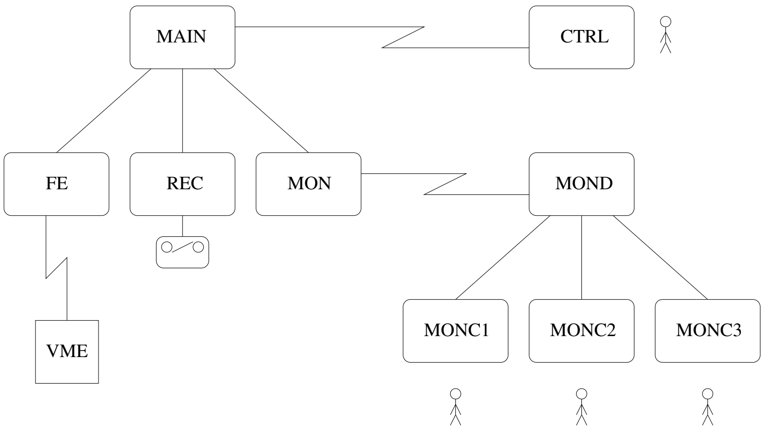

Figure 2: Processes: MAIN – main process, FE – front-end process, REC – recording process, MON – provides events for MOND, CTRL – run control program, MOND – monitor daemon, MONCx – user monitoring programs.

3.1 Interprocess communication

For processes located at different machines a simple protocol over TCP/IP was developed. We recognized the idea of CORBA but found that in our case TCP/IP suits well.

For interprocess communication on the MAIN computer a segment of shared memory was used[1]. A contiguous block of shared memory is accessible for four processes running on the MAIN machine. This block is divided on fixed size buffers. The size of each buffer is guaranteed to be larger than biggest possible event (5 Kbytes). The buffers are binded in circular linked list.

This data structure is created, deleted, and reseted by the MAIN process. Other processes use these buffers. The FE process works as a producer filling the buffers by events. The REC

[1] More simple implementation could be done using pthreads. Unfortunately at the time of the first version development no suitable implementation of threads under Linux was available.

process works as a main high-priority consumer. The MON process works as a secondary consumer.

The low-level synchronization of memory buffers access is done with mutexes and condition variables which are implemented via semaphores.

3.2 Processes

The purpose of the MAIN process is to start and stop FE, REC and MON child processes. It also provides the communication between the child processes and the Run Control program.

All interactions with electronics goes through and only through the FE process.

The REC process has a unique task of storing events on an I/O device. Recording can be done on disk files only, on the magnetic tape device only, or on both. As a tape device a DLT unit is used. Event recording is always done in an "exchange" mode allowing to read data later at any platform. User can setup the configuration parameters defining a name of tape drive, tape record size, maximum size of disk files, maximum size of tape files, tape capacity etc.

The purpose of the MON process is to provide events for the monitor daemon (MOND). It is not guaranteed that this process will take all events from buffers.

The monitor daemon program (MOND) works as a bridge between the main dataflow (FE $\rightarrow$REC) and user monitoring programs. The monitoring programs can not affect the main dataflow by any means.

Remote monitoring programs can be started at any workstation with a TCP/IP connection to the target system. They verify the performance of the detector and the quality of the data during data acquisition. To visualize the results of the data analysis two programs were developed: one is based on the GTKPLOT package and another one on the ROOT system [6].

The run control program (CTRL) coordinates all activities in data taking and gives an operator the full control over entire DAQ system. This is a GTK+ [5] program which provides a graphical interface.

The online system also uses MySQL [7] database for storing time history of the data taking conditions, ambient data, and configuration parameters of hardware and software.

4 Conclusions

Our main design goal was to use the set of PC's running the Linux OS for data acquisition, monitoring, and analysis. We got from that many benefits such as: (i) usage of the modern powerful hardware providing required computer resources; (ii) creation of the uniform software environment at front-end and the remaining computers; (iii) usage of the high quality free general purpose software jointly with the specific HEP software. After four successful data-taking periods we can conclude that the choice of PC's running Linux was a robust and reliable idea.

References

[1] V.N.Bolotov et al, "Experimental Setup ISTRA-M to Study Rare Decays of Charged Light Mesons", IHEP preprint 95-111, Protvino, 1995.

[2] `http://www.sbs.com`

[3] `http://www.ces.ch`

[4] `http://www.nikhef.nl/~natalia/projects/vmehb.html`

[5] `http://www.gtk.org`

[6] `http://root.cern.ch`

[7] `http://www.mysql.com`