A C++ based DAQ system for small and medium sized experiments with LabView as graphical user interface

Joerg Tutas

RWTH Aachen, Phys. Inst.

tutas@physik.rwth-aachen.de

According to the needs of typical laboratory test requirements a small DAQ system was developed based on VME. The features of an object oriented approach led to the choice of C++ as programming language for the kernel. This minimises the effort of implementing new devices into the existing framework, and the idea of using an object oriented data base for data storage can be implemented most naturally. LabView, a commercial graphical programming language, was selected to provide easy to use tools for comfortable control and on-line data presentation

Key words: C++, object oriented methods, off-the-shelf products, data acquisition systems, GUIs

1 Introduction

Many small and medium sized test setups are developed, operated and maintained in the field of High Energy Physics. People involved in this procedure often work for limited time on the project and have limited experience in setting up save and reliable data acquisition systems. Moreover lacking experience with required hardware devices which are sometimes complicated and badly documented lead to delays in the development process.

However the basic task of most devices is rather similar. They have to collect data which have to be delivered to a data acquisition program. The underlying hardware platform, be it VME or Camac, as well as the internal settings and procedures to initialise boards is not relevant to the user.

The aim of the project was to decouple all hardware dependent features of the devices such that they are hidden from the user. He only has to deal with the functional aspects of his hardware.

Portability of the code was also an important issue. Different Test Setups are equipped with different computer configurations which makes it important to provide building blocks which can easily be ported to different platforms at minimal effort.

Last but not least the operation of the DAQ systems has to be as easy as possible and has to provide good monitoring features to get fast feedback from the data. This is a problem which is not only present in HEP and commercial tools are at hand to help. LabView[1] was chosen for this purpose because it offers a great variety of possibilities and is available on many platforms (Mac OS, Sun, HP-UX, Windows NT/95, Windows 3.1).

VMEbus as most widely spread hardware platform was used as a basis for the project. As a starting point a Hewlett Packard VME CPU (HP 9000/743) and a Power Macintosh coupled to VME via a PCI - VME Interface was chosen as target machines. The gnu compiler was used on the Unix platform while on the Macintosh CodeWarrior 1.6 was used.

2 Architecture

The class library of our system is very simple. All boards are based on the abstract class **VME_Board**. Within this class all basic VME parameters are defined, such as Base address, address modifier, size of the board and the mapped address. At the same time the basis of a simple data structure is implemented in the base class. All real implementations of the VME boards are derived from that class.

It seemed practical to define two additional classes which both deal with Memory boards. One is called **VME_Buffer** and the other **VME_Memory**. The first provides buffer handling when streams of data have to be written to a buffer while the latter allows memory access in RAM mode with some service for memory allocation in different partitions. They can be easily exchanged whenever local shared memory on the CPU board is to be used.

Implementation of Camac devices is provided by the a VME - Camac interface[2] . This board represents a VME class from which in turn the Camac modules are derived. In this way all available VME and Camac boards are made available for easy implementation. An excerpt of the class structure can be seen from figure 1.

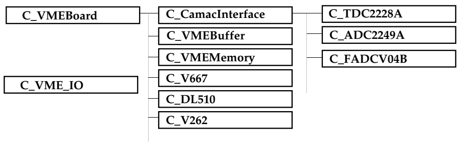

Fig. 1. Excerpt of the Class library.

Two functions are common to all boards which deliver data: ReadData() and StoreData(). They are defined in the base class and have to be overwritten where appropriate.

3 VME I/O implementation

Typically the VME interface is different on all platforms which complicates portability of programs. For this reason another independent class was introduced which deals with all the VME functionality. This class provides a set of routines which encapsulate all the VME access. It exists as a separate object version for each platform but contains the same functions. It is initialised once for each process.

[1] product of National Instruments

[2] VC16 from Wiener

At creation time all required initialisations on the system level are performed. Then for each new VME board the function *Map_VME_Board()* is called with all relevant parameters. Depending on the platform and according to the address modifier the VME address is mapped to a system address and timeouts are set correspondingly. VME access is further on provided by calls to functions as *Write(), Read(), BlockRead(), BlockWrite()*.

This encapsulation certainly reduces the overall system throughput but increases the portability of the code very much. A performance test was done on the HP. A VME I/O Register was used to measure two consecutive register accesses which were monitored with the help of the corresponding output pulses. In the first case (plain c) the access was performed directly via a pointer to the register address. In the OO case the access was achieved first by a call to the corresponding member function of the Board Class which in turn calls **VME_IO** to perform the access. Results are summarised in table 1.

plain c	direct access	690 ns
OO	encapsulated	1.13 μs
OO	encapsulated with inline fct for VME IO	1.06 μs
OO	direct VME access	690 ns
OO	optimised compilation	690 ns

Table 1
VME access times.

4 Application example

The complete DAQ system contains different building blocks as indicated in figure 2. The main process (DAQ) is responsible of collecting the data after occurrence of the trigger. Data are stored temporarily in an intermediate buffer before bigger amounts are written to disk. This process is controlled and monitored by the System SuperVisor (SSV) which can also be responsible for writing the data from the buffers to disk. A Monitoring process (MON) is responsible for monitoring the collected data. On request the DAQ process provides a copy of an event to a shared buffer where the data can be analysed by the monitoring job.

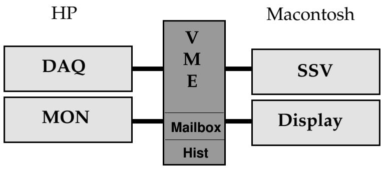

Fig. 2. DAQ structure.

In one implementation we use the HP as the engine for the basic DAQ process. LabView is installed on the Macintosh. The comfortable environment of LabView suggest to place the SSV process as LabView VI on the Macintosh. Communication between the two processes is provided as mailbox on the Dual Ported VME memory module (DPM) of 8 MB which is directly accessible from the Mac via PCI without VME access.

Monitoring of the data itself is done as a task on the HP. Analysed data are again stored in the DPM. From here they can be further analysed and displayed with LabView taking advantage of all the available features. Interfacing to LabView is done via a CIN[3] which

[3] Code Interface Node

increase speed and allows the use of the same structure definitions which are also used in the C++ code.

5 Outlook

Next implementations and performance tests will be made with a PowerPC 401 board from CES and a Dec Alpha station. The code has been ported to a Windows NT PC platform but could not yet be tested finally due to a lacking VME interface. With the help of D. Duellmann of RD 45 at Cern, first tests have already been started to implement Objectivity [4], an object oriented data base, directly in the online process. It will be interesting to study the performance of this configuration.

[4] Trademark of Objectivity, Inc