A DAQ System for Linear Collider TPC Prototypes based on the ALEPH TPC Electronics

M. BALL[1], N. GHODBANE[2], M. JANSSEN[1,3], P. WIENEMANN[1]

[1]*DESY, D-22607 Hamburg, Germany*

[2]*CERN, CH-1211 Geneva 23, Switzerland*

[3]*University of Dortmund, D-44221 Dortmund, Germany*

Abstract

Within the international studies of a high energy linear electron positron collider, several groups are developing and testing prototypes for a Linear Collider TPC. This detector is planned to be used as a central part in the tracking system of a detector at such a machine. In this note we describe a DAQ system, which has been developed for the use in tests of TPC prototypes. It is based on electronics used at the ALEPH experiment at CERN.

1 Introduction

A large Time Projection Chamber (TPC) is proposed as part of the tracking system for a detector at the future electron positron linear collider [1]. To meet the different milestones and requirements, several institutes have started a joint effort to develop a technology based on the use of micro pattern gas detectors like GEMs or Micromegas instead of the conventional wire chamber based solution. A powerful but simple DAQ system is needed at this stage, based on already available technologies, to enable these tests with minimum effort and expenditure. The DAQ system described in this note is based on the existing electronics used at the ALEPH experiment at LEP, CERN, for the readout of the ALEPH TPC.

In this note we introduce the basic concept of the readout system, and discuss in detail each of the different components.

2 Principle of Operation

The ALEPH TPC DAQ is based on an FADC system. Signals from each channel of the TPC are sent to a preamplifier, which integrates the charge and converts it into a voltage signal. The integration time constant can be influenced by a feedback capacitor, and is set in its default configuration to 2 μs. The signal from the preamplifier is sent to the digitizer via a twisted pair cable. The first stage of the digitizer is a receiver/shaper amplifier, followed by an 8-bit FADC, operated at 12.5 MHz. The digitized information is written into a memory bank, which can keep up to four different events of 512 time slices each. The basic setup of the system is shown in Fig. 1.

The required performance of a DAQ for the Linear Collider TPC has been summarized in the TESLA TDR [1]. The requested performance and the relevant parameters for the ALEPH system are summarized and compared in Tab. 1.

Parameter	TDR DAQ	ALEPH DAQ
Sampling Speed	>20 MHz	12.5 MHz
ADC Range	9 bit	8 bit
Storage Depth	1 ms	512 time slices $\hat{=}$ 41 μs

Table 1: Table of requirements of the DAQ as specified in the TESLA TDR and as delivered by the ALEPH based DAQ described in this note.

3 Hardware Setup

The TPC DAQ is based on a combination of FASTBUS technology with VME based readout. The digitizer units (TPDs) are realised as FASTBUS modules, through a slightly modified version of a standard FASTBUS crate is employed. Through a special link (see below for more details) the FASTBUS crate is controlled and read out from a VME based CPU. The complete system is controlled by an external computer, running the Linux operating system. In detail the following components are needed to assemble a complete system (see Fig. 1):
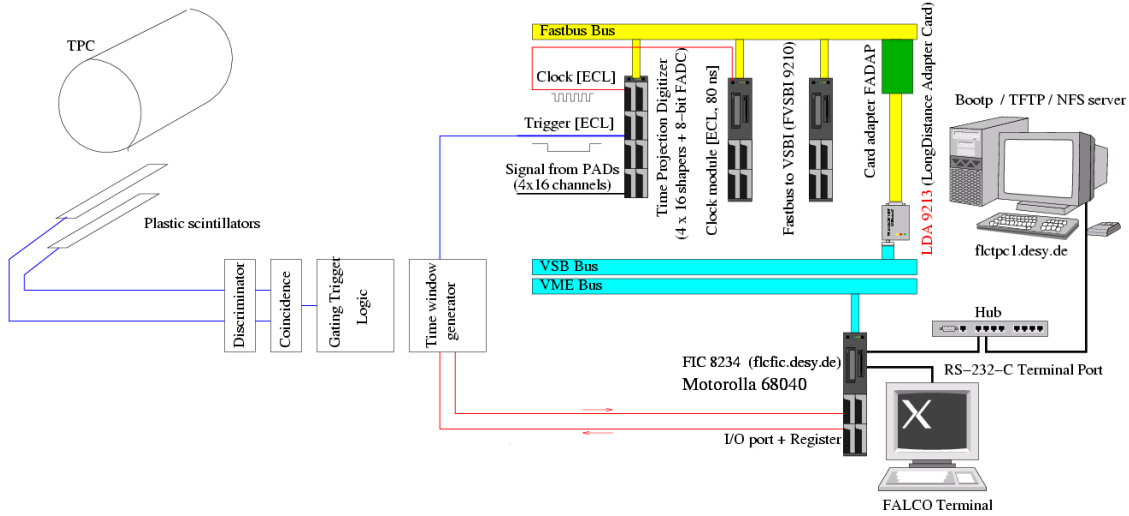
Figure 1: Setup à la ALEPH for the TPC.

- One computer operating under Linux (SuSE 7.3 operating system),

- A 6U VME crate with:

  - A Fast Intelligent Controller (FIC 8234 [2]), A 68040 VME/VSB dual processor unit running at 25 MHz.

  - A VME/VSB adapter card, Long Distance Adapter LDA 9212 or LDA 9213.

- A modified FASTBUS crate with:

  - A FASTBUS to VSB translator unit (FVSBI 9210 [3])

  - A set of Time Projection Digitizer (TPD 6821 [4]) modules

  - One FASTBUS Adapter Card, FADAP 9211.

- A NIM crate containing several electronic modules for the trigger selection. The trigger signal has to be converted finally into an ECL signal, which can be used by the TPD.

In the following sections the way of the signal through the electronic chain is described. A complete description of the ALEPH TPC electronics can be found in [5].

3.1 The Preamplifier and Shaping Amplifier

Conventional TPCs like that of ALEPH have used a multi-wire proportional chamber technique to multiply the primary electrons at the endplates. With this method, thin anode wires are mounted just in front of the readout pads. Due to the high electric field in the vicinity of the wires, electron multipication takes place so that the produced charges induce measurable signals on the readout pads. Unfortunately this technique leads to a dependence of the resolution on the projected angle between track and wires. Moreover

the electric and magnetic field lines are not parallel close to the wires leading to a significant drift velocity component along the $\vec{E} \times \vec{B}$ direction. This eventually might limit the spatial resolution of the chamber. In addition wires and the necessary support structures add large amounts of material to the endplate, and thus compromize the performance of the calorimetry in the forward direction.

For the Linear Collider therefore a novel readout scheme for a TPC is proposed, based on micro-pattern gaseous chambers. In recent years GEMs and Micromegas have been developed to a point where their use in large scale detectors can be envisioned. Contrary to the wire chamber readout, TPCs with a GEM or Micromegas based gas amplification system directly detect electrons produced in the amplification step. Since this signal is very fast, the parameters of the preamplifiers need to be specially optimized for this situation.

The input stage of the ALEPH preamplifier is a charge-integrating circuit with a decay time constant of 2 μs. The effective feedback capacitor is C_f = 1 pF. The charge sensitivity is determined by the feedback capacitor C_f to be 1 V/pC. For a single pad of size 2 $\times$ 6 mm a typical primary electron signal is around 25 electrons. After gas amplification this translates into a preamplifier output signal of $\frac{f \cdot g \cdot q}{C_f} \approx$ 12 mV with the charge collection efficiency f due to the integration time, the gas gain g, the primary ionization charge q and the capacitance C_f of the storage capacitor.

The output signal of the preamplifier rises very fast – typical rise times are of the order of a few tenth of a ns – and then decays exponentially. The shaper symmetrizes the signal and converts it into a roughly Gaussian shape with a width of about 400 ns. The FADC then digitizes this signal at 12.5 MHz, resulting in a typically length of 5 time slice samples for one pulse.

Technically the preamplifiers are connected to the TPDs via twisted pair cables. The preamplifiers are supplied with power through the signal cables. $\pm$ 5 V needs to be supplied either from an external power supply or through a modification of the FASTBUS crate.

Both the preamplifiers and the shaping amplifiers were produced in thick-film hybrid technology. Measurements done at the time of production of the electronics showed that the gain and the noise of the devices were constant within an RMS of 1.5 % for the preamplifier gain, $\pm$ 5 ns for the FWHM of the output pulses after shaping.

A special FASTBUS module (SMTPD) exists which, apart from other things, provides the necessary 5 V to a modified FASTBUS backplane, so that the supply voltage can be applied to the preamplifiers.

3.2   The TPD

In this Section we discuss the Time Projection Digitizer (TPD). A more complete description can be found in [4].

The TPDs are implemented in FASTBUS technology and adhere to the FASTBUS standard. The modules are controlled through the FASTBUS control bus.

The TPD, originally designed to digitize the signals of the ALEPH TPC, comprises 4 $\times$ 16 input channels, each of them built around an 8-bit Flash Analog To Digital Converter (FADC) preceded by a hybridized line receiver/shaper amplifier. The TPD uses the standard ALEPH TPC preamplifiers as signal sources.

The TPD supplies power to the preamplifiers as described in the previous Section. For testing purposes a specially designed module is available (SMTPD). This module in addition provides clock, trigger and pulse signals and is primarily used during the testing and debugging phase of the electronics.

Later during routine operation an external source was used to supply the $\pm$ 5 V to the preamplifiers.

When using old ALEPH equipment special care needs to be taken to make sure that a properly modified FASTBUS crate is available when the SMTPD modules are used.

3.2.1 Front Panel Input Signals Handling

A TPD accepts three different kinds of input signals:

- The clock (CK) at a frequency of 12.5 MHz, used to sample the signal from the preamplifiers. The clock has to be a standard NIM signal.

  The clock can be either supplied from the SMTPD – which is suitable for a small number of TPDs – or through an external clock generator. Simple ways of realizing such an external clock generator are given in the Appendix A to this note.

- The trigger or write signal (WR). The WR signal starts the acquisition procedure of the FADCs. The WR signal duration must be greater than 512 time slices $\times$ the CK period, i. e. 41 μs.

- The signals from the preamplifiers connected to the pads of the TPC.

Once the TPD gets the WR signal, each of the 64 input signals is passed through the shaper and the FADC, with a typical sampling time of 80 ns (CK signal). The digitized information is piped into a buffer which is 512 bytes deep, called the Raw Data Memory (RDM). Four parallel banks are available, allowing to store up to 4 events in the system before the events have to be readout into the acquisition computer.

The events stored in the TPD are eventually written into static memory, after applying some data selection criteria like threshold suppression, pulse time extraction, etc.

3.2.2 Control Status Registers

The full functionality of the TPD is available through various internal 32 bit registers. These are written and read by the FIC 8234 via the VSB to FASTBUS interface FVSBI 9210.

The TPD implements several control and status registers (CSR). The most relevant ones, which are used in the acquisition system are:

- CSR#0: This register is a general purpose control and status register implemented according to the FASTBUS specifications. It contains the TPD module identifier and the different control bits for proper operations. The CSR#0 bit significance is given in Tab. 2.

- CSR#1 is used for the access and DAQ Bank informations. The CSR#1 bit significance is given in Tab. 3.

- Two Next Transfer Address (NTA) registers, one for the DATA space and one for the CSR space.

- `CSR#0xC0000002` or DAC register implements four 6-bit values, so called DAC values to set the parameters of each FADC channel. These four values control the pedestal, linearity and the gain of the TPD.

3.3 The FASTBUS to VME/VSB Interface (FVSBI)

The FVSBI 9210 connects the VME to the FASTBUS, and thus allows the control of the FASTBUS system. It communicates with the VSB bus using two extension cards, a Long Distance Adapter card (LDA 9213) which is plugged into the rear of the VSB connectors. A passive adapter, the F-side Adapter (FADAP 9211), is connected at the rear of the FASTBUS crate to the LDA 9212.

There are two directions of communication. The FIC can control and set all relevant registers etc. of the TPDs in the FASTBUS crate. After the digitization step, the FIC reads the memory in the TPDs and transfers the data through the VME crate to the DAQ computer.

4 Acquisition Software Description

4.1 Approach Using the RDM Banks

Each of the 64 8-bit FADCs digitizes the signal from the ALEPH preamplifiers and stores this event in the so called Raw Data Memory (RDM). A simple approach consists in reading this memory once the digitization is done and apply offline threshold cuts and sophisticated algorithms to find the different charge clusters.

This simple approach has been implemented in the acquisition software and the different steps are:

- Initialization of the FASTBUS session.

- Initialization of the FVSBI master (see Section 3.3).

- Loop on the different FASTBUS slots and for each primary address (PA) try to read the CSR#0, where the identity of the FASTBUS module is written and store the PA associated with the different modules (TPDs and SMTPDs).

Having the list of TPDs in use, then, for each of them:

- Reset the TPD and all CSRs by setting the `CSR#0<30>` to 1.

- Set the DAQ bank to be bank 0, 1, 2 or 3 using `CSR#1<19:18>`.

- Load the DAC values used by the 8-bit FADCs.

- Enable DAC writing by setting `CSR#0<12>`.

- Loop on the 64 channels and for each of them:

  - Define the channel to be loaded in the NTA_DATA register `NTA_DATA<14:09>`.
  - Load the DAC value in the `CSR#0xC0000002` register.

- – Wait for bit clear of `CSR#0<13>`.
  - – Wait a bit that the DAC value loaded is stable.

- Disable the DAC serializing, setting `CSR#0<28>`.

- Write to the Least Significant Bit (LSB) of CSR#1, `CSR#1<00>`.

- Enable the DAQ, `CSR#0<06>`.

- If a new trigger is accepted, then a WR cycle starts and `CSR#0<07>` goes up to 1. Then an image of the current number of time samples can be found in the DAQ counter `CSR#1<07:00>`.

- The DAQ stops if the DAQ counter reaches 511 and then the `CSR#0<07>` is reset to 0.

- Disable the DAQ setting `CSR#<22>`.

- Set the Access Bank to read to be bank 0, 1, 2 or 3 using `CSR#1<17:16>`.

- Read the data channel by channel or block by block.

4.2 Acquisition Software

The TPD readout has been described in detail in the Section before. The `C` program `tpcdaq.c` is well commented and the names of the different implemented functions are self-explanatory. In this Section, we list the different implemented functions and summarize briefly their task. These functions are called from the main program `TPCAcquire()` and are:

- `FastBusInit()`: Initializes the FASTBUS session.

- `FastBusScan()`: Scans the different FASTBUS segments and returns the PA of the different TPDs and SMTPDs present.

- `getFVSBInterfaceNumber()`: Returns the FVSBI 9210 interface number set on the front panel.

- `getFastbusModuleID()`: Tries to read the CSR#0 at a given PA and decode the module identity coded in this register.

- `FastBusClose()`: Finishes the FASTBUS session.

- `checkFBError()`: Decodes the FASTBUS error if any.

The SMTPD is not crucial for the data acquisition since it can easily be replaced by a NIM clock module. Nevertheless, as described above, it enables to test and calibrate the TPD. Thus several functions have been implemented starting from the definition of the different registers in reference [6]. These functions are:

- `SMTPDReset()`: Reset the SMTPD.

7

- `SMTPDClock()`: The clock is not automatically started at the power up of the SMTPD. this has to be done by software.

- `SMTPDSetSource()`: The SMTPD can be used as a testing module which generates the input signals for the TPD. This function has not been tested.

To access the different TPDs several functions have been implemented. These functions are:

- `TPDReset()`: Reset the TPD module.

- `TPDSetDAQBank()`: Select one of the four banks to which the sampled event will be written to.

- `TPDSetDAC()`: Set the DAC parameters to control the slopes of the different FADC channels.

- `TPDEnableDAQ()`: Prepare the TPD for the acquisition of the next event.

- `TPDSetDAQ()`: Prepare the TPD for the acquisition setting `CSR#0<07>`.

- `TPDDAQActive()`: Checks whether the `CSR#0<07>` set previously is still up. At the end of the digitization procedure this bit returns to 0.

- `TPDWaitForTrigger()`: Reads the CSR#1 time slice counter set initially to 1 and incremented during the digitization of the pulse. At the returns, this reaches 512 time samples.

- `TPDDisableDAQ()`: Disable DAQ setting `CSR#0<22>`.

- `TPDSetAccessBank()`: Set the access bank to one of the four banks, where the data have been stored during digitization.

- `TPDReadFADCCharge()`: Read the bank where the raw data are stored.

- `TPDReadRDM()`: Read the event from the Raw Data Memory (RDM) directly. As explained above. No selection criteria is applied to the event in the TPD.

The TPD is capable of applying selection criteria to the raw data stored in the RDM. To do this, several functions have been implemented. These functions are[1]:

- `TPDSetThreshold()`: Set the threshold to be used.

- `TPDSetLimitRegister()`: Set the Limit register to define the cluster (total number of time slices and number of time slices below threshold).

- `TPDDolist()`: Start or apply the selection on the data stored in the TPD banks.

- `TPDReadHLM()`: Read the data stored in the Hit List Memory (HLM) after the selection.

---

[1]At the time of writing this note, this part of the system has not yet been extensively tested. Please contact the authors to find out about the current state of this part of you intend to use these functions

The FIC 8234 implements a front panel Input/Output controller, which can be used to generate e. g. a gate (e. g. with a FLIP/FLOP). Several functions are provided to drive this controller. These are `FPInit(),FPClose()` and `FPWrite()`.

The data read from the TPD by the FIC 8234 can either be stored using NFS or using the more powerful client/server approach explained below. The drawback of the first approach is that the FIC 8234 communicates only with one computer, its NFS server, whereas the second approach enables the FIC to communicate with several computers which can run reconstruction programs in parallel. Several `C` functions have been implemented for this purpose. These are `NFSFileConnect(),NFSFileDataSend()` and `NFSFileDisconnect()` for the approach using the NFS transfer protocol and `TCPIP-ClientDataSend()` for the TCP/IP client server based solution.

5   Network and Server Configuration

The master processor of the whole data acquisition system is the Fast Intelligent Controller, the FIC 8234. The FIC is operated in diskless mode. This means that upon startup the operating system has to be loaded through the network from a network disk drive.

The FIC uses the BOOTP and TFTP protocols to find and upload its operating system:

1. The FIC broadcasts a BOOTP request to find a BOOTP server.

2. If a BOOTP server is present it replies to the BOOTP request and sends the network settings (IP address, gateway address, name server address, TFTP server address, etc.) to the FIC. Care has to be taken that a proper EPROM is used with an up-to-date version[2].

3. The FIC reboots itself with the settings it has received from the BOOTP server.

4. The next step of the network configuration is that the FIC starts a request for a TFTP server to load the boot image of the OS-9 operating system.

5. The last step of the configuration is to establish a NFS connection to the Linux PC to have access to the file space where the acquisition software is located.

The different steps to implement all these technical modifications are presented on the web page http://www.cern.ch/ghodbane/tpc.

The system has been developed using Linux, SuSE 7.3 professional edition, with the following software versions:

- Linux Kernel: 2.4.10-4GB.

- BOOTP server: bootp-DD2-4.3-87 (as an RPM).

- TFTP server: tftp-0.20-22 (as an RPM).

- Kernel NFS server and NFS utilities nfs-utils-0.3.1-87 (as an RPM).

Newer distributions should work as well, but have not been tested.

---

[2]Older EPROMs send BOOTP requests with 255.255.255.255 as source address. Such packets are ignored by the BOOTP server. The correct source address is 0.0.0.0. You have to ask CES Electronics to replace OS-9/68040 2V4 Rev. A2 by OS-9/68040 2V4 Rev. 1.4.
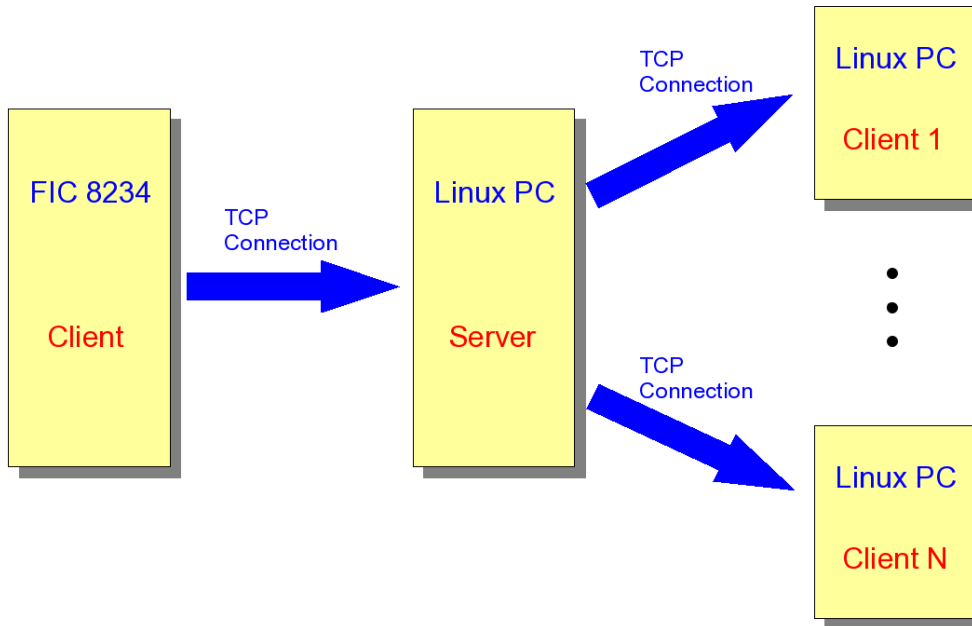
Figure 2: The client server based data distribution system used by the DAQ system. Data processed by the FIC CPU are sent to a central data distribution server which in turn forwards the data to an arbitrary number of clients connected to it.

6 Client Server Based Data Distribution

The data processed by the FIC CPU are sent via a TCP connection to a central data distribution server (DDS) running on a Linux PC. This server forwards all data received from the FIC to any other clients connected to it. Possible data receiving clients are data writing programs to make the data persistent, online monitoring software to check the data quality, etc. In principle an arbitrary number of clients can connect to the DDS. In order to minimize the load of the FIC CPU and to keep the data transfer through the slow 10 MBit/s connection of the FIC as low as possible, the server does not run directly on the FIC but on a separate machine. A sketch of the setup is shown in Fig. 2.

6.1 Data Transfer Protocol

The data are sent in chunks as 8 bit `char` arrays to the DDS. Since all numbers to be transferred are integers, they are represented either as 8 bit `char`s or 32 bit `int`s, depending on their range. An `int` is packed into a `char` array by distributing the 32 `int` bits over multiple `char`s.

At the beginning of each event, a `0x61='a'` character is sent to indicate the start of a new event. This is followed by an event number, the UNIX time when the event has been recorded and the number of TPDs used. These numbers are filled sequentially into a character array which is transferred to the DDS *en bloc*. Then for each TPD, the TPD ID and the number of pulses is filled into a character array and sent to the DDS. The meaning of "pulse" depends on the user requirements. It can be just a couple of time slices in a

channel which exceed a certain threshold value, but it can also be the full FADC spectrum of a channel. The latter definition has been adopted as the default setting, i. e. the number of pulses is equal to the number of channels. The next data to be transferred in a large array is the contents of each time slice for all the pulses. Finally at the end of each event, a `0x65='e'` character is sent to indicate the end of the event.

In order to avoid control characters from being transferred, the protocol works on a 7-bit per character basis. The most significant bit (MSB) of each character is always set to one[3], so that only 7 payload bits remain per character. The only exceptions with MSB off are `0x61='a'` and `0x65='e'` which indicate the beginning and the end of an event, respectively.

6.2 LCIO Data Writing Client

The developed setup uses LCIO [7], the Linear Collider data model and persistency framework. A client has been implemented which connects to the DDS and writes the transmitted data to disk in LCIO format.

The LCIO classes are kept as general as possible in order to avoid any unneeded restrictions. The class `IMPL::TPCHitImpl` provided by LCIO is designed in such a way that data from both FADC and TDC based TPC readouts can be stored without unnecessary overhead. Therefore a FADC specific class called `TPCPulse` has been added to the LCIO class collection to allow easy access to all relevant information. This class is defined in the following way:

```
#ifndef LCIO_TPCPULSE_H
#define LCIO_TPCPULSE_H 1

#include "LCObjectHandle.h"
#include "IMPL/TPCHitImpl.h"


typedef lcio::LCObjectHandle< IMPL::TPCHitImpl > TPCHitImplHandle ;


class TPCPulse : public TPCHitImplHandle {
public:
    TPCPulse( DATA::LCObject *lcObj) : TPCHitImplHandle(lcObj)  {;}
    TPCPulse( IMPL::TPCHitImpl *lcObj) : TPCHitImplHandle(lcObj)  {;}

    virtual ~TPCPulse() {;}

    int     getNBins() const          { return _lcObj->getNRawDataWords() - 1; }
    int     getTime(int bin) const    { return _lcObj->getRawDataWord(0) + bin; }
    int     getCharge(int bin) const { return _lcObj->getRawDataWord(bin + 1); }
    int     getTPDID() const          { return _lcObj->getCellID() / 100; }
    int     getChannel() const        { return _lcObj->getCellID() % 100; }

    void    setPulseSpectrum(int tpd, int channel, int time0,
                            const int* charge, int nbins){
        lcObj()->setCellID(tpd * 100 + channel);
        int* a = new int[nbins+1];
        a[0] = time0;
```

---
[3]Note that all control characters have a MSB set to zero.

11

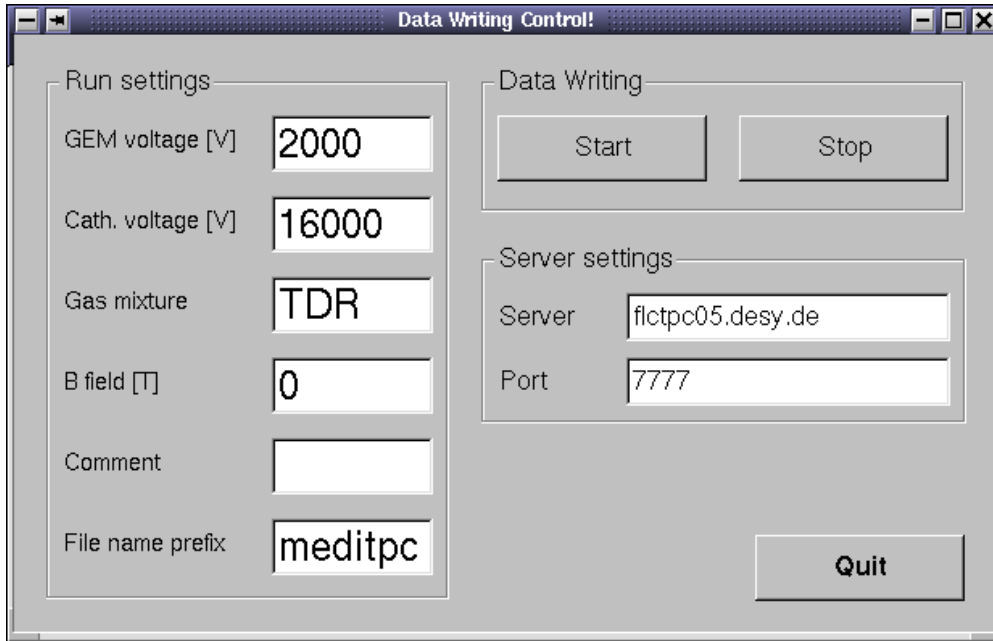**Figure 3:** A screen shot of the control window of the LCIO Data Writing Client.

```
        for (int i=0; i<nbins; i++) a[i+1] = charge[i];
        lcObj()->setRawData(a, nbins+1);
        delete[] a;
    }

};

#endif
```

A screen shot of the program control window is shown in Fig. 3. In addition to the data from the readout electronics, important run parameters like cathode voltage, magnetic field value etc. can be given. They are stored in the run description string[4] available in each LCIO run header. The file name prefix provided by the user is used to compose the data file name. It has the following components: `prefix.YYYYMMDD.XXX.slcio`. `YYYYMMDD` is the date when the run has been started and `XXX` a serial number starting from `000`. Every 1000 events a new file is created and its serial number is increased by one.

Data recording is started by pressing the start button. Only after the start button has been pressed, an LCIO file is opened and a TCP connection to the given server and port is established. A run can be stopped by pushing the stop button which closes the LCIO data file and the TCP connection to the DDS.

---

[4]Each field is separated by a "|". Therefore a "|" should not be contained in the entered string to avoid confusion.

7   Summary and Conclusion

A DAQ has been developed which is based on existing hardware which was used in the ALEPH experiment at LEP. The DAQ offers a low-cost solution to the need of having DAQ systems of intermediate power available during the R&D phase for a LC TPC. The DAQ offers reasonable performance, and can be used to study setups of reasonable complexity.

It is clear however that this DAQ system can not be considered as a prototype for the eventual acquisition system a the LC TPC. Acquisition speed, package density are too low, the power consumption is much too high for this. Significant R&D is needed to develop and design a powerful yet compact readout system for a future LC TPC.

8   Acknowledgments

References

[1] T. Behnke, S. Bertolucci, R.-D. Heuer, and R. Settles, TESLA Technical Design Report, DESY 2001-011 and ECFA 2001-209, http://tesla.desy.de/new_pages/TDR_CD/start.html

[2] FIC 8234, Dual 6804 Fast Intelligent Controller, User Manual, Version 2.0, C.E.S. SA.

[3] FVSBI 9210, FASTBUS to VSB Interface, User Manual, Version 0.4, C.E.S. SA.

[4] B. Lofstdet, *TPD, Time Projection Digitizer*, User Manual.

[5] C. Bowdery *et. al.*, *The ALEPH Handbook 1995, Volume 1*, 1995, 123-176.

[6] G. Schuler, *A Source Module, SMTPD, for the ALEPH Time Projection Digitizer*, User Manual.

[7] F. Gaede, T. Behnke, N. Graf, T. Johnson, *LCIO – A persistency framework for linear collider simulation studies*, LC Note LC-TOOL-2003-053, Project Home Page: http://lcio.desy.de

A   Building a Simple External Clock Generator

A simple external clock can be built using a dual gate generator. The setup described in this note has been tested with a LeCroy NIM Module, model 222. The required cabling for such a clock is shown in Fig. 4.
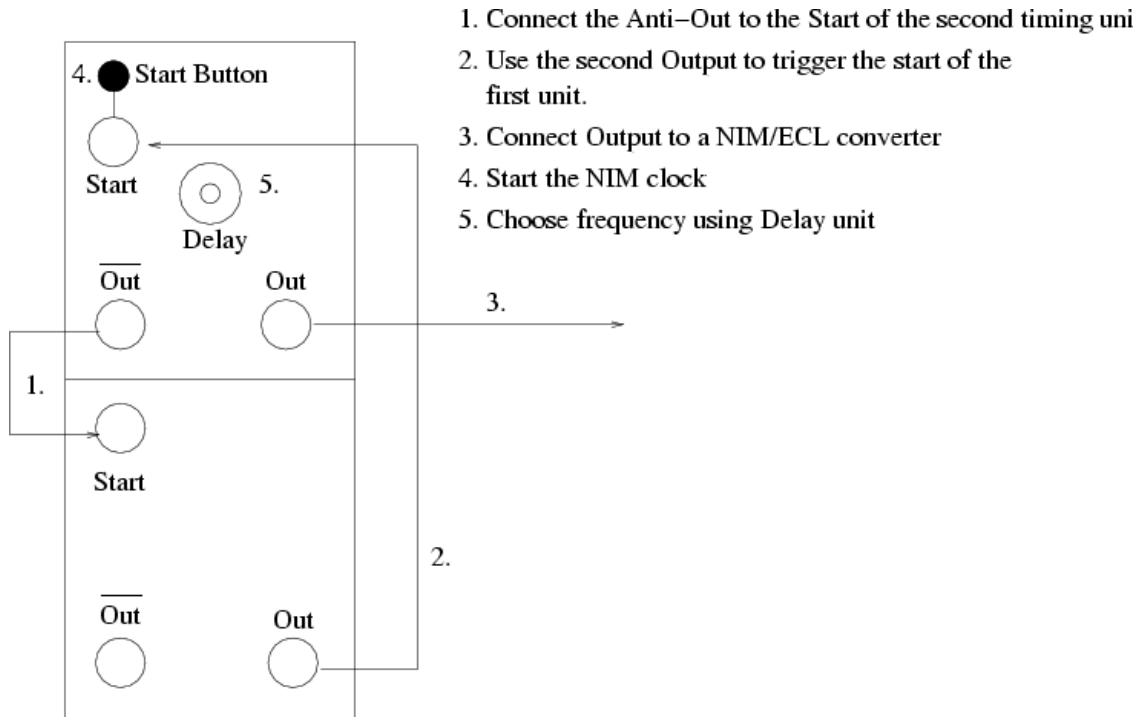
Figure 4: Setup of an external clock generator using a dual gate generator.

Bit	Read Significance	Write Significance
00	Error flag	set error flag
04	Serialize (SR) assertion enabled	Enable SR assertion
05	SR asserted	
06	DAQ enabled	Enable DAQ
07	Status DAQ active	
08	Status Dolist active	enable Dolist
09	Status hit counter overflow	
11	Test input enabled	
12	Digital to Analog Converter (DAC) enabled	Enable DAC writing
13	Status DAC serial load	
20		Disable SR assertion
21		Reset serialize (SR)
22		Diable DAQ
24		Disable Dolist
28		Disable TEST input
29		Disable DAC serial load
30		Reset module (including CSR)
31		Reset registers and counter but not CSR

Table 2: CSR#0 bit assignement

Bit	Significance
08:00	DAQ counter
17:16	Access bank
19:18	DAQ bank

Table 3: CSR#1 bit assignement

Bit	Read Significance	Write Significance
30		Reset the SMTPD
31:16	Module identifier: 0x6917	

Table 4: CSR#0 bit assignment for the SMTPD

Bit	Read Significance	Write Significance
07:00		Pulse amplitude value

Table 5: CSR#1 bit assignment for the SMTPD

Bit	Read Significance	Write Significance
06	CP out enabled	Enable CP generation
07	Signal pulse enabled	Enable signal pulse
08	Pulse routed to channel output	Route pulse to channel output
13	CP internaly generated	Enable internal CP oscillator

Table 6: CSR#2 bit assignment for the SMTPD