ISTITUTO NAZIONALE DI FISICA NUCLEARE

Sezione di Trieste

INFN/TC-07/10
July 18, 2007

THE UPDATE OF THE RICH CONTROL SYSTEM

Veronica Diaz[1]

[1)] *INFN, Sezione di Trieste, Padriciano 99, I-34012 Trieste, Italy*

Abstract

In this note, I report the features of the RICH Control System software package. This performs the programming and control of the COMPASS RICH-1 read-out. The porting of the all the software modules of the system to Windows XP Operating System and the Update of the hardware of the richctrl computer will allow to work with the updated XP Operating System that will be supported by CERN in the next coming years.

Published by **SIS-Pubblicazioni**
Laboratori Nazionali di Frascati

1 INTRODUCTION

The COMPASS is the NA58 (http://wwwcompass.cern.ch) project at CERN in Geneve, Switzerland. The COMPASS RICH-1 [2] is a large-size Ring Imaging Cherenkov detector which performs hadron identification. Its large-volume vessel is filled with C4F10 radiator gas. Cherenkov photons emitted in the gas are reflected by two spherical mirror surfaces. The photons are converted to electrons by the CsI photon cathodes of eight Multi-Wire Proportional Chambers (MWPC), which amplify the single photoelectrons and detect them. The quest for sufficient number of Cherenkov photons for this gas determines the overall length of the radiator vessel to be of about 3 m. The RICH-1 geometry results in a photon detector surface of 5.6m2. The surface is covered with eight proportional chambers (MWPCs), equipped with CsI photon converter layers. A dedicated radiator gas system establishes continuous gas circulation in a closed loop and ensures both optimum VUV transparency and constant relative pressure in the vessel.

The general architecture of the COMPASS RICH-1 read-out is described in [1].

The RICH Control System is a set of programs that allows programming and controlling the read out system of the RICH-1 [1,4] of COMPASS.

In a nutshell the system is controlled by the RICH Control Computer (richctrl.cern.ch), a PC, that programs and controls the data acquisition activities of 192 DSPs (Digital Signal Processor), organized in a daisy chain with time-sharing protocol, and an equal number of FPGAs (Field Programmable Gate Arrays).

The RICH Control PC has a dedicated multiprocessor custom board, called DOLINA, to handle the complete RICH read out system.

The first version of the system as presented here, was installed during 2001 and was improved with new features during 2002 and 2003 in several layers of the system. It was used with good performance during 2002 and 2004 data taking of the COMPASS Experiment.

A new version updated of the all modules presented here, was implemented by the author during 2005. The updated system was made in both aspects, at hardware level changing the control computer and at software level New Operating System. Both aspect will be explained in detail in this document.

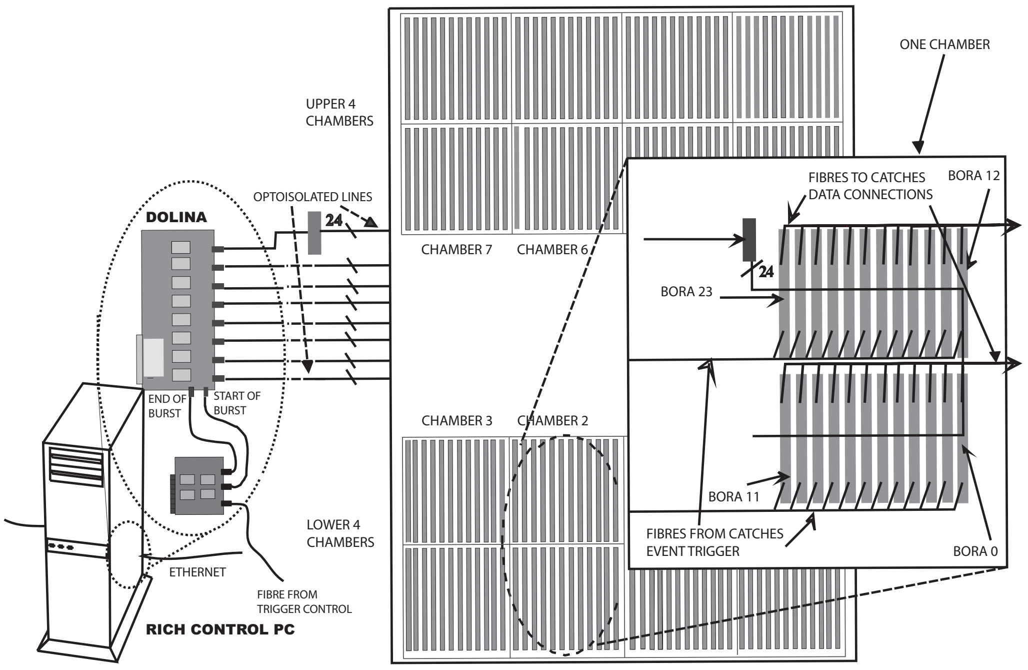


Figure 1: A top-level view of the front-end data acquisition of RICH-1

2 DESCRIPTION OF THE SYSTEM

The system can be described by the following scheme. The read-out system for the 82944 RICH-1 channels is located in the COMPASS experimental Area, building 888 at CERN (see figure 1). It is based on 192 BORA [1] boards connected to the pixels of the detector and organized in eight chambers with 24 boards by chamber.

As we mentioned before, the overall operation of the board is controlled by a DSP complemented by a FPGA that acts as a parallel co-processor of the DSP.

Each BORA communicates with the outside world through i) a 400 Mbit/s fibre optics link, used to transmit event data to the COMPASS CATCHes that constitutes the subsequent processing stages of the COMPASS acquisition system, and ii) through a serial connexion with DOLINA board, via a dedicated DSPs serial network formed by 24 DSPs for each chamber.

BORA boards are plugged directly onto the detectors and thus are located in the experimental area. The rest of the system is placed outside this area, in the RICH Barrack. There are two dedicated PCs to operate the RICH: richctrl computer and richsrv computer (standing for RICH server). Both PCs are interconnected through a dedicated ethernet link, which guarantees a high speed in the data transmission between the two computers (see figure 2).

There is a set of Application programs that allow the control, monitoring and programming of all the programmable chips in the system. This set of programs is described in detail in the section 3.1.

The richctrl computer (richctrl.cern.ch), is an Intel XEON dual processor of 2.80 GHz, 2 GB of RAM, and has a large amount of storage capacity with several hard disks. This is the actual hardware configuration of the computer as was updated during 2005.

The jobs are CPU intensive and the main application that runs in this computer has multiple threads to be executed on it. The software of the system profits the speed and the parallelism of the two processors.

Plugged in this computer we have the DOLINA; a custom multiprocessor board with PCI interface and eight DSP's inside. Each DOLINA DSP works as master of one of the serial chamber networks introduced before, allowing the control and supervision of the complete RICH electronics.

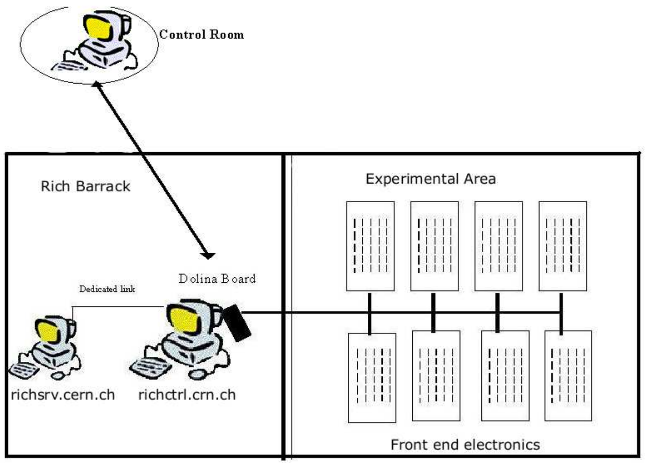


Figure 2: RICH ReadOut components organization

The richsrv computer (richsrv.cern.ch) stores the acquired data that are remotely saved from the richctrl computer. Due to this architecture the most relevant feature of this computer is the storage capacity, 400 GB disk storage. The rate of data arriving from the data acquisition system generates about 3,5 GBytes per day.

To allow the RICH Control Applications to communicate with the hardware, we have developed the DOLINA Device Driver working on Windows XP Operating System, as was updated during 2005. This module will be described in detail in section 4.

To complete this scheme, there is a third level composed by a set of scripts embedded in the main Central DAQ System of the experiment [11]. This module of the system is executed from the Central DAQ System of the experiment, which is physically located in the run control machine cluster.

These scripts embedded in the COMPASS DAQ System consist of a set of instructions and commands defined in the RCSL (RICH Control Scripting Language); more details of this module can be found in section 3.2.

3 THE USER APPLICATIONS

There is a set of User application programs *API* running in the computers of the RICH barrack: each one has a specific function in the system.

Some of these programs share some files and resources, so there is a unique structure for the set of processes that intercommunicate among them in order to synchronize specific tasks.

The RICH Control Application system consist of the following programs:

1. RICH Control application
2. RICH Control Server application
3. RICH Event Viewer application
4. RICH Noise Viewer application
5. RICHDataBackup application

3.1 RICH Control Application

Description:

This program is the main application used by the RICH operator; it performs the communication between him and the RICH Control Data Acquisition System. During the execution time this process communicate with the others applications sending messages and sharing data.

This application is executing in the richctrl computer, allows the user to send commands and programs through the RICH DSP network to the BORA boards and indirectly to the CATCHes [11] using the optical fibres from the BORA Boards, offering the complete control of the system by the user.

This application performs three fundamental set of tasks:

1. The communication with all the devices available in the system: This is the main task of the RICH Control Application which properly controlls the Data Acquisition System, and allows the study of the RICH Read-Out behavior and the hardware programming.

In this set of tasks the program manages several kinds of internal data packets: engineering frames, calibration and noise measurements, thresholds, events samples, commands, programs and generates several kinds of files depending on data. More information about the different kind of packets can be found in [3].

In order to have the control and a good administration of the packets that arrives to the PC, there is one set of threads running permanently waiting for each type of packet.

The packets arrives to the memory of the DOLINA board and are stored in a buffer in the RAM memory of the richctrl computer. Each thread is ready for processing or dispatching (o sending a pointer to) a piece of memory (packet) to another specific thread to complete the processing. The specific thread that constructs the data files or answer an specific command possibly put themselves to sleep waiting for another packet in order to complete the piece of information necessary to saves in a file or sends an answer or acknowledge depending of the packet content.

This scheme using multi threading programming makes this module complex due to the threads synchronization and the interaction with DOLINA device. A complete description of this scheme is described in the RICH Control System Web site.

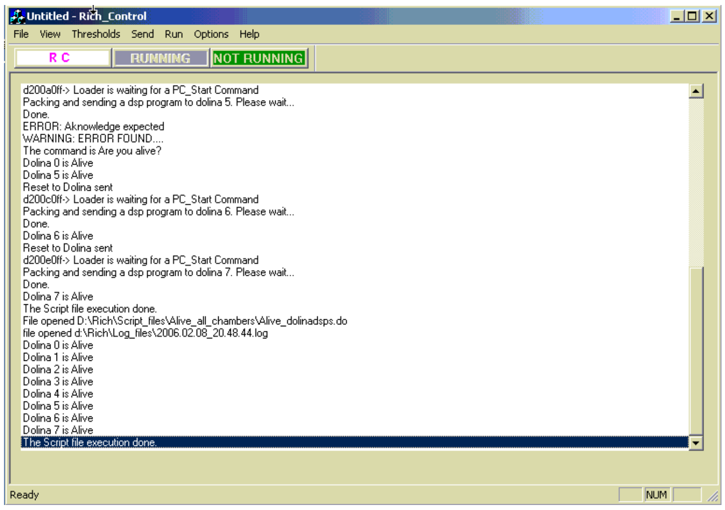
2. The communication between the user and the data acquisition system provided by the windows application Graphic User Interface (GUI), the script editor and the main view application.

   This set of tasks forms the main User Interface. It is formed by an user friendly interface with pop-up menus, several dialogs that allows several kind of activities and one editor where you can entry a new script or select one stored script that performs one RICH standard task. You can see a view of this GUI as an example in figure 3 and all the details of this application and settings are explained in the RICH Control System Manual [5].

   This module defines the control of the basic settings of the RICH Control System application. Their values are saved in the RICH control configuration file.

   There is another application that shares the access to this file: the ”RICH Control Server Application”, which is described in section 3.2.

   Both processes communicates between them using messages. The RICH Control Server process sends to the main application messages when a new command arrives from the Compass Control Room. Then the main application answers back and changes the settings in the configuration file when is necessary.



```
d200a0ff-> Loader is waiting for a PC_Start Command
Packing and sending a dsp program to dolina 5. Please wait...
Done.
ERROR: Aknowledge expected
WARNING: ERROR FOUND....
The command is Are you alive?
Dolina 0 is Alive
Dolina 5 is Alive
Reset to Dolina sent
d200c0ff-> Loader is waiting for a PC_Start Command
Packing and sending a dsp program to dolina 6. Please wait...
Done.
Dolina 6 is Alive
Reset to Dolina sent
d200e0ff-> Loader is waiting for a PC_Start Command
Packing and sending a dsp program to dolina 7. Please wait...
Done.
Dolina 7 is Alive
The Script file execution done.
File opened D:\Rich\Script_files\Alive_all_chambers\Alive_dolinadsps.do
file opened d:\Rich\Log_files\2006.02.08_20.48.44.log
Dolina 0 is Alive
Dolina 1 is Alive
Dolina 2 is Alive
Dolina 3 is Alive
Dolina 4 is Alive
Dolina 5 is Alive
Dolina 6 is Alive
Dolina 7 is Alive
The Script file execution done.
```

Figure 3: A view of the main RICH Control Application

3. There is a scripting language implemented in the core of the RICH Control application. *The RCSL (RICH Control Scripting Language)*, is a script language that follows the philosophy of Tcl/Tk or Pearl and easily allows the creation of complex task using a set of atomic commands. More details of this see RCSL (The RICH Control Scripting Language) document.

3.2 RICH Control Server Application

This process allows the control of the RICH Data Acquisition from the Control Room.

It is possible to send commands from the Control room in order to Start and Stop the acquisition, for instance.

In this application the client/server model in network programming is used.

A client application has to make a request to be recognized by another computer in the network. This second computer has a server application running that fulfills the clients request and returns the information.

In our case the richctrl computer (in the RICH Barrack) will have the RICH Control Server process (socket server) running continuously, waiting (listening) for a command, while the client process will be running on the computer of the Control Room, sending a request each time it is necessary.

This server process locally exchanges messages and information with the RICH Control application which, in its turn, sends the commands to the BORA's.

Both processes communicates using interprocesses messages and access to some memory locations where important information is shared between both processes.

The status of the exchanged information and the final values is registered in the RICH configuration file. This file is accessed for the main application too.

The RICH Control Server process sends to the RICH Control application some messages when a new command is arrived from the Compass Control Room. The RICH Control Application sends back the answer message to the server process after a specific task is completed.

For instance it sends the ”The RICH is started” message after the RICH BORA boards have received the ”Start Of Run” command through the DSP serial network.

3.3 The Viewers Application

The ”RICH Event Viewer application” allows the visualization of a complete RICH event acquired by the control PC.

The events are saved by the RICH Control application on the local disk in a file in the event folder and this feature allows recovering events by date, run number, etc.

This application shows in color scale the event values of all the 82344 channels of the RICH and it is possible positioning the mouse over a pixel read the specific hit value, it is possible perform a zoom of a interesting area too.

When the events arrive to the RICH Control computer they are saved on the local disk by the RICH Control application in a file with extension ”evt” in the event folder, where it is possible to find the date, the run number and the burst number.

This file has only two data per channel: the address of the pixel and the hit value. Then an event file is saved remotely on the richsrv computer for each BORA board with the complete event information: event header, quantity of hits and event trailer. More details about the event file information is given in [5].

The same procedure is used to visualize the noise or the calibration values of all the 82344 channels of the detector in this case the ”RICH Noise Viewer application” is used See figure 4.

The noise measurement start with a ”channel test” command transmitted from the PC to the BORA DSP to request the measurements of the noise level and pedestals of individual channels corresponding with the BORA (432 channels). The format of the packets for channel test command (short packet) is presented in [3]. The number of samples (samples-number) to be used in the measurements (from 1 to 2048) is a parameter of the channel test request.

When a ”channel test” packet arrives to the BORA DSP, the DSP program calculates the sum of the raw values obtained by channel and the sum of the squares of the raw values obtained by channel taking ”samples-number” samples.

Then the DSP sends the packets to the PC containing such sums together with the corresponding channel ID. The format of the noise network packets (long data packet) is presented in [3].

The RICH control Application uses the sums to calculate pedestals (average output voltage when the input has no signal) and noise level (the standard deviation without input signal) by channel.

The raw channel values acquisition is done by the BORA FPGA after receive an internal or external trigger and more details of this procedure is presented in [3].

This allows perform specific studies about the performance and Behavior of the electronics and its interaction with the environment.

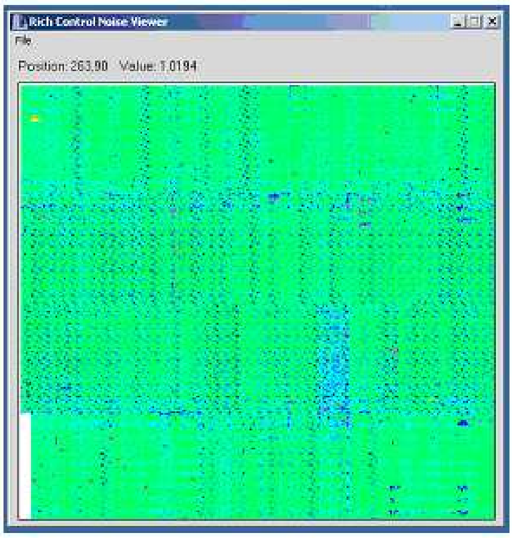


Figure 4: Noise viewer

3.4 RICH Data Back Up Application

The rate of data arriving from the Front-End system to the Control PC is around 3,5 GBytes per day, the events are saved remotely on the richsrv (richsrv.cern.ch) computer in the RICH Barrack, as we mentioned before.

There is an application called RICHDataBackup running continuously copying the data from the specified RICH data directory to the rewritable CDs.

When the application starts running it automatically backup all the files in the specified directories to the destination CD. When the application detects the CD is full, check the second CD drive and start to backup the files there.

The source directories and the destinations drives are specified in the RICH data backup configuration file.

When the application is minimized, it stays on the system tray and continue running in the background.

The application shows the status of the CD's and the progress of the saving process as well. More details about this process, setting of the ”config.ini”, configuration file and the complete information and a view of the user interface is given in [5].

4 DOLINA DEVICE DRIVER

A device driver provides a software interface to the hardware connected to a computer. It is a trusted part of the operating system. User application programs can access hardware in a well-defined manner, without having to worry about how the hardware must be controlled.

The DOLINA device driver is responsible of controlling the DOLINA card and makes this card accessible to the user application. It allows the user application to communicate with the DOLINA DSPs acquiring data.

The Dolina driver is a piece of software that becomes part of the operating system kernel when it is loaded. A driver makes one or more devices available to the user mode programmer, each representing physical or logical piece of hardware.

The Dolina device driver was written following the architecture of one standard monolithic device driver of the Microsoft Windows family of operating systems. You can find a description of the Windows driver structure and features in [10] and [9].

This driver provides the Kernel functions for the user interface functions of the operating system. These kernel driver functions allow the user application to communicate with the device hardware using an object handle. This is possible because in Windows a driver makes a device look like a file. A handle to the device can be opened. An application program can then issue read and write request to the driver, before the device handle is finally closed.

You can find the main functions provided by the Dolina device driver in 6. One complete description of the driver functions specifications and parameters is available in the Microsoft Developer Studio Manual [8] and [7].

5 CONCLUSIONS

The architecture and the set of programs that form the RICH Control System is installed and it was working with excellent performance in the last three years (2002-2004) of data tacking of the Compass Experiment at CERN (http://wwwcompass.cern.ch).

During the year 2005 one complete update of the system was implemented by the author. At the lower level the DOLINA Device Driver, that it was written initially to work on Windows NT, now is updated to work under Windows XP. The main applications were updated to follow the requirements of the new Operating system and same improvements were added as well. The RICH control computer was updated as was described before.

The first version of this package modules was developed under the direction of Prof. Alberto Colavita and tested in collaboration with all researchers and technical members of the Microprocessor Laboratory up to 2004 year. In particular Dr. Razaq Ijaduola played a fundamental role in this work as developer of DOLINA Device Driver under Windows NT operating system.

6 APENDIX: The DOLINA Device Driver Functions

We will describe here the main functions provided by the custom Dolina device driver:

1. HANDLE CreateFile(LPCTSTR lpFileName, DWORD dwDesiredAccess, DWORD dwShareMode, LPSECURITYATTRIBUTES lpSecurityAttributes, DWORD dwCreationDistribution, DWORD dwFlagsAndAttributes, HANDLE hTemplateFile);

   Parameters:

   lpFileName : Points to a null-terminated string that specifies the name of the object to create.

   dwDesiredAccess : Specifies the type of access to the object. An application can obtain read access, write access, read-write access, or device query access.

   dwShareMode : Set of bit flags that specifies how the object can be shared. If dwShareMode is 0, the object cannot be shared.

   lpSecurityAttributes: Pointer to a SECURITY_ATTRIBUTES structure that determines whether the returned handle can be inherited by child processes. If lpSecurityAttributes is NULL, the handle cannot be inherited.

   dwCreationDistribution: Specifies which action to take on files that exist, and which action to take when files do not exist.

   dwFlagsAndAttributes : Specifies the file attributes and flags for the file.

   hTemplateFile : Specifies a handle with GENERIC_READ access to a template file. The template file supplies file attributes and extended attributes for the file being created.

   Return Values

   If the function succeeds, the return value is an open handle to the specified file. In this case a object handle to the DOLINA device object that allow to reference the device.

2. BOOL ReadFile(HANDLE hFile, LPVOID lpBuffer, DWORD nNumberOfBytesToRead, LPDWORD lpNumberOfBytesRead, LPOVERLAPPED lpOverlapped);

   Parameters

   hFile : Identifies the file to be read. The file handle must have been created with GENERIC_READ access to the file.

   lpBuffer : Points to the buffer that receives the data read from the file.

nNumberOfBytesToRead : Specifies the number of bytes to be read from the file.

lpNumberOfBytesRead : Points to the number of bytes read.

lpOverlapped : Points to an OVERLAPPED structure. This structure is required if hFile was created with FILE_FLAG_OVERLAPPED. If hFile was opened with FILE_FLAG_OVERLAPPED, the lpOverlapped parameter must not be NULL. It must point to a valid OVERLAPPED structure. ReadFile does not return until the read operation has been completed.

Return Values

If the function succeeds, the return value is nonzero.

If the return value is nonzero and the number of bytes read is zero, the file pointer was beyond the current end of the file at the time of the read operation. If the function fails, the return value is zero. To get extended error information, call GetLastError.

Remarks

The ReadFile function reads data from a file, starting at the position indicated by the file pointer. After the read operation has been completed, the file pointer is adjusted by the number of bytes actually read, unless the file handle is created with the overlapped attribute.

Using the DOLINA Device functions gets data from the dual port memory and returns this data in the lpBuffer to the user application. If the file handle is created for overlapped input and output (I/O), the application must adjust the position of the file pointer after the read operation.

3. BOOL WriteFile(HANDLE hFile, LPCVOID lpBuffer, DWORD nNumberOfBytesToWrite, LPDWORD lpNumberOfBytesWritten, LPOVERLAPPED lpOverlapped);

   Parameters

   hFile : Identifies the file to be written to. The file handle must have been created with GENERIC_WRITE access to the file.

   lpBuffer : Points to the buffer containing the data to be written to the file.

   nNumberOfBytesToWrite : Specifies the number of bytes to write to the file.

   lpNumberOfBytesWritten: Points to the number of bytes written by this function call. WriteFile sets this value to zero before doing any work or error checking.

lpOverlapped : Points to an OVERLAPPED structure. This structure is required if hFile was opened with FILE_FLAG_OVERLAPPED.

If hFile was opened with FILE_FLAG_OVERLAPPED, the lpOverlapped parameter must not be NULL. It must point to a valid OVERLAPPED structure. If hFile was not opened with FILE_FLAG_OVERLAPPED and lpOverlapped is not NULL, the write operation starts at the offset specified in the OVERLAPPED structure and WriteFile does not return until the write operation has been completed.

Return Values

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero. To get extended error information, call GetLastError.

Remarks

The WriteFile function writes data to a file and is designed for both synchronous and asynchronous operation. The function starts writing data to the file at the position indicated by the file pointer. Using the DOLINA Device functions the user application putting (the content of the lpBuffer) data into the dual port memory belonging to specific DSP. After the write operation has been completed, the file pointer is adjusted by the number of bytes actually written, except when the file is opened with FILE_FLAG_OVERLAPPED. If the file handle was created for overlapped input and output (I/O), the application must adjust the position of the file pointer after the write operation is finished.

4. BOOL CloseHandle(HANDLE hObject);

   Use the CloseHandle function to close an object handle returned by CreateFile. CloseHandle invalidates the specified object handle, decrements the objects handle count, and performs object retention checks. Once the last handle to an object is closed, the object is removed from the operating system.

There are other driver kernel functions to be used in specific task to control the board. Is possible to find the complete list and definitions of the functions in the RICH Control System Web site.

References

[1] Baum *et al*, THE COMPASS RICH-1 READ-OUT SYSTEM, Instr. Meth. A 433**1**, 426, 1-41 (2003).

[2] E. Albrecht *et al*, Status and characterization of COMPASS RICH-1 and references therein, Nuclear Instruments and Methods in Physics Research (NIM-A) journal A 553, 215 (2005).

[3] V. Diaz, Rich Control System Web Site, http://www.cern.ch/veronica.diaz, (2005).

[4] E. Albrecht *et al*, COMPASS RICH 1, Nuclear Instruments and Methods in Physics Research (NIM-A) journal A 502, 112 (2003).

[5] V. Diaz , Rich Control System Manual, Rich Group-INFN Trieste (2005).

[6] Microsof, Microsoft Developer Studio Integrated Development Environment (IDE), http://msdn.microsoft.com/virtuallabs/ (2005).

[7] Microsoft Corporation, Microsoft VisualC++ 6.0 Reference Library, ISBN 1-57231-865-1, Microsoft Press (1998).

[8] Deitel, Deitel, Nieto and Strassberger, Visual 6 IDE one Introduction to MFC, Microsoft Press (1998).

[9] D.D. Burn, Getting Started with the Windows Driver Development Environment, Reliable Technologies-Microsoft.com/whdc/driver/kernel (April 2005).

[10] C. Cant, Writing Windows WDM Device Drivers, R&D Books (2000).

[11] COMPASS Collaboration, Proceedings of IEEE Realtime 2003 Conference, IEEE Tran. Nucl. Sci. **1** (2003).