DEVICE CONFIGURATION HANDLER FOR ACCELERATOR CONTROL APPLICATIONS AT JEFFERSON LAB*

M. Bickley, P. Chevtsov, T. Larrieu, Jefferson Lab, Newport News, VA 23606, USA

Abstract

The accelerator control system at Jefferson Lab uses hundreds of physical devices with such popular instrument bus interfaces as Industry Pack (IPAC), GPIB, RS-232, etc. To properly handle all these components, control computers (IOCs) must be provided with the correct information about the unique memory addresses of the used interface cards, interrupt numbers (if any), data communication channels and protocols. In these conditions, the registration of a new control device in the control system is not an easy task for software developers. Because the device configuration is distributed, it requires the detailed knowledge about not only the new device but also the configuration of all other devices on the existing system. A configuration handler implemented at Jefferson Lab centralizes the information about all control devices making their registration user-friendly and very easy to use. It consists of a device driver framework and the device registration software developed on the basis of ORACLE database and freely available scripting tools (perl, php).

INTRODUCTION

With the use of superconducting RF technology, the CEBAF accelerator provides nuclear physics experiments at Jefferson Lab with electron beams of very high quality. The accelerator control system at Jefferson Lab is based on the Experimental Physics and Industrial Control System (EPICS) toolkit [1]. With its very powerful set of tools, EPICS allows easy system extensions at all control levels. The software presented in this paper has been designed primarily for an EPICS environment but can easily be ported to any control system.

Hundreds of physical devices at Jefferson Lab are controlled over such popular instrument bus interfaces as Industry Pack (IPAC), RS-232, and GPIB. The Industry Pack bus is a point to point bus from a carrier board to an IPAC module or card [2]. The IPAC card is passive on the bus and can only be accessed by the carrier board. IPAC provides a convenient and inexpensive way of implementing a wide range of control, I/O, analog and digital functions. Various GPIB and RS-232 interface modules are also available in the IPAC standard. This makes Industry Pack an attractive connection method for integrating physical devices into the accelerator control system. The device driver framework that has been created at Jefferson Lab provides a very efficient software solution for this integration.

*Supported by DOE Contract #DE-AC05-84ER40150.

DEVICE DRIVER FRAMEWORK

The device driver framework consists of IPAC, GPIB, and Common Serial driver libraries.

IPAC Driver Library

The IPAC driver library provides a standard software interface to different types of IPAC carrier boards. The library is based on the drvIpac software developed by A. Johnson [3]. It has a common IPAC interface module and a set of carrier board support modules. All control and data requests to the carrier boards go through the common interface module which in turn calls the carrier board support module written for the particular type of carrier board.

Each carrier board is activated or registered with the use of only one library call:

stat = initIpCarrierBoard(CBT, PARAM_STR)

where *stat* is the board registration status (traditionally, it is equal to 0 if the registration is OK or contains an error code in case of failure), *CBT* is a carrier board type, and *PARAM_STR* is a string containing board-specific initialization parameters which are very important for the IPAC Driver Library. This parameter string always contains the I/O base address of the board in the VME A16 address space. It may also have the information about the used VME A24 address space, interrupt numbers, etc. The order in which carrier boards are registered defines the carrier number for the control computer (IOC) to deal with, starting from zero for the first registered board. When the carrier board is activated, all its slots become available for communication with the IOC.

GPIB Driver Library

The GPIB driver library at Jefferson Lab was written with the use of the ideas implemented in the Message Passage Facility (MPF) of M. Kraimer [4]. The library consists of a general GPIB control module and an IPAC interface module. The general GPIB control module is completely responsible for the GPIB communication protocol. The IPAC interface module deals with GPIB communication cards. The communication card is activated with the use of the next library call:

stat = initGpibIpacLib(CBN, SLN, intNum)

Here *stat* is again the registration status, *CBN* is a carrier board number, *SLN* is the number of the slot (starting from 0 for slot "A") housing the card, and *intNum* is the interrupt vector number that is used by the card.

During the activation, the GPIB driver makes several checks to make sure that the specified carrier and slot are legal, the interface card is installed and has the valid manufacturer and model ID values in its PROM. It also calls all necessary GPIB initialization routines and connects the interrupt service routine to a particular interrupt vector number. The activated card provides the communication bus for controlling GPIB devices connected to it.

Common Serial Driver Library

The common serial driver library deals with serial (RS-232) ports used for the connections with control devices [5]. It has a serial hardware support module and a serial port control block. The serial hardware support module handles various RS-232 communication hardware components including IPAC serial interface cards. Each serial port is served by a separate control task that exchanges messages with the EPICS database records referencing this port. The serial port control task reads and writes data into and out of the serial port. It also handles the hardware operation timeouts.

Each serial port on any IPAC card is activated with the use of the next library call:

stat = ipacSerialPortConfig(CBN, SLN, SPN, parms)

where *stat* is (as always) the registration status, *CBN* is the carrier board number, *SLN* is the slot number occupied by the card, *SPN* is the port number on the card and *parms* is the list of the basic serial communication parameters: the used baud rate, data word size, number of stop bits, parity, etc.

The serial port activation procedure is very similar to the registration of GPIB communication lines described above. The activated port becomes available for communication with a serial device connected to it.

DEVICE REGISTRATION SOFTWARE

One of the main advantages of the device driver framework described in the previous section is that it typically does not require any software coding for connecting a new physical device to the control system. All that needs to be done to set up EPICS control for this new device is to create the EPICS database that takes care of the device command protocol and run it. From the other side, to properly activate and handle device communication channels, the control computers must be provided with the correct information about unique memory addresses of the used interface boards, interrupt numbers, etc. In these conditions, the device registration bookkeeping is not an easy task for control system software developers. Because the device configuration is distributed, it requires the detailed knowledge about not only the new device but also the configuration of all other devices on the existing system. A configuration handler implemented at Jefferson Lab centralizes the information about all control devices making their registration user-friendly and very easy to use.

The idea of the device configuration handler is transparent. As mentioned above, each device communication channel is activated by a very limited number (one or two) of the corresponding driver library calls. For each IOC and for each particular type of the used instrument bus, all activation calls are combined into one communication bus configuration (or device configuration) file. All these files reside in one directory in the control computer file system. At start up time, specially designed VxWorks shell scripts download the required communication bus configuration files into the IOC and activate the used communication channels.

Furthermore, information about different carrier boards and device interface modules used at Jefferson Lab is stored in an ORACLE database. When it is necessary to add a new device to the control system, user-friendly graphical forms can be used to allow selection from the list of available device interface modules and then to enter the parameters specific for the chosen module type. After the forms have been filled out, the required communication bus configuration files are generated automatically. It is no longer necessary to edit device configuration files by hand.

By maintaining information in the database for each IOC and all its associated control devices, it is possible to ensure that new additions will not introduce conflicts. For example, when a new serial interface module is added to a carrier board, the automated process makes sure that the configuration file doesn't register it in the same slot with any of the existing serial modules. Some web based graphical forms for the registration of serial devices are shown in Fig.1-2.

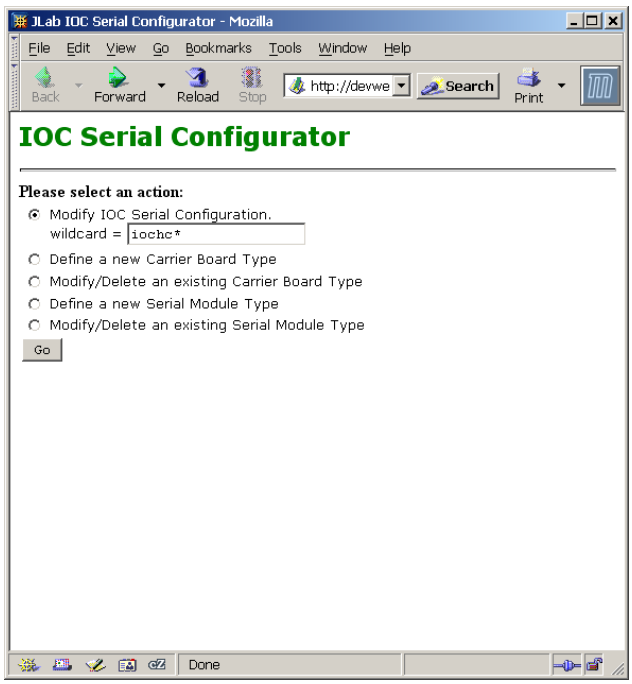

Figure 1: Basic serial device registration graphical form.

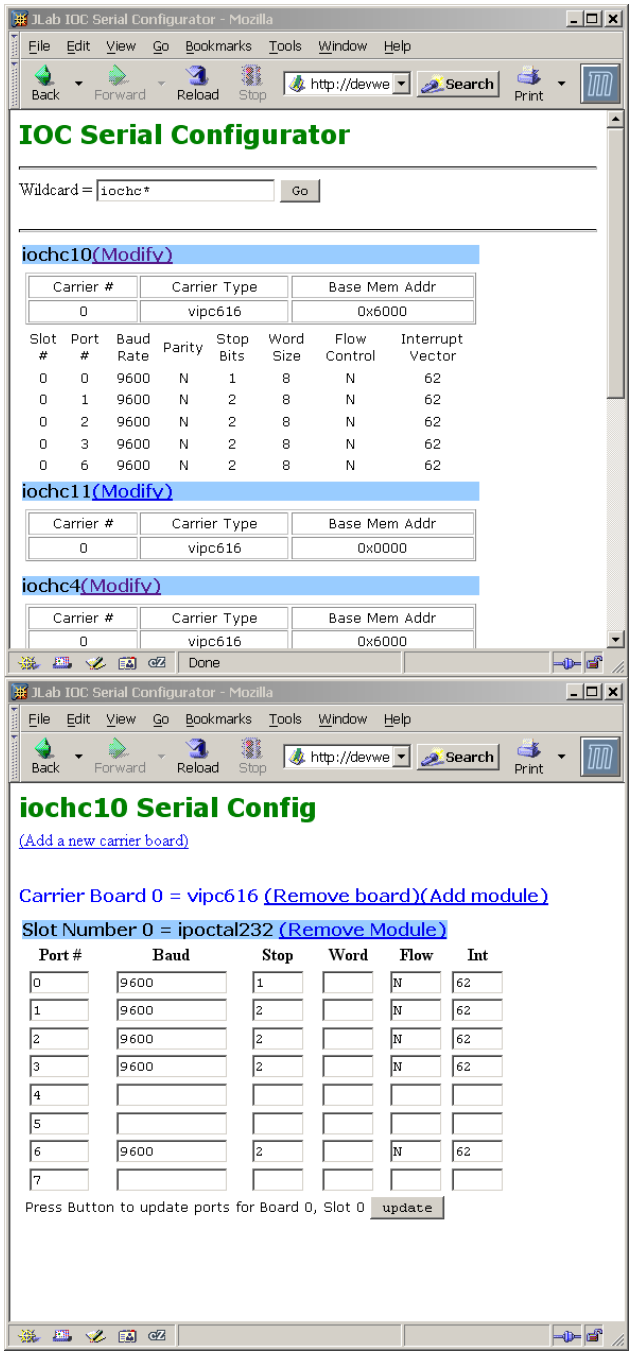

Figure 2: Graphical forms used to register serial devices controlled by a particular IOC.

CONCLUSIONS

The device configuration handler at Jefferson Lab significantly simplifies the work on the control applications for physical devices. It does not require software coding for connecting a new device to the control system. The information about all devices and their communication parameters is kept in the ORACLE database. This eliminates any possible errors in these parameters. It also makes it possible to automate the creation and modification of the communication bus configuration files which are used by the device driver framework on the basis of ORACLE and freely available scripting tools (perl, php).

ACKNOWLEDGMENTS

The authors are very thankful to K.White for her support of this work.

REFERENCES

[1] B. Dalesio et al., “The Experimental Physics and Industrial Control System Architecture: past, present and future”, NIM, A 352 (1994), p. 179-184.

[2] M. Timmerman, “Comparison of Different Mezzanine Buses”, Real-Time Magazine, 97-1, p. 77-81.

[3] A. Johnson, “EPICS Industry Pack Module”, www.aps.anl.gov/asd/people/anj/ipac/

[4] M. Kraimer, “Message Passing Facility”, www.aps.anl.gov/asd/people/mrk/epics/modules/bus/mpf

[5] P. Chevtsov, S. Schaffner, “Information-Control Software for Handling Serial Devices in an EPICS Environment”, ICALEPCS 2001, San-Jose, CA, USA, 2001.