UCRL- 85669
PREPRINT

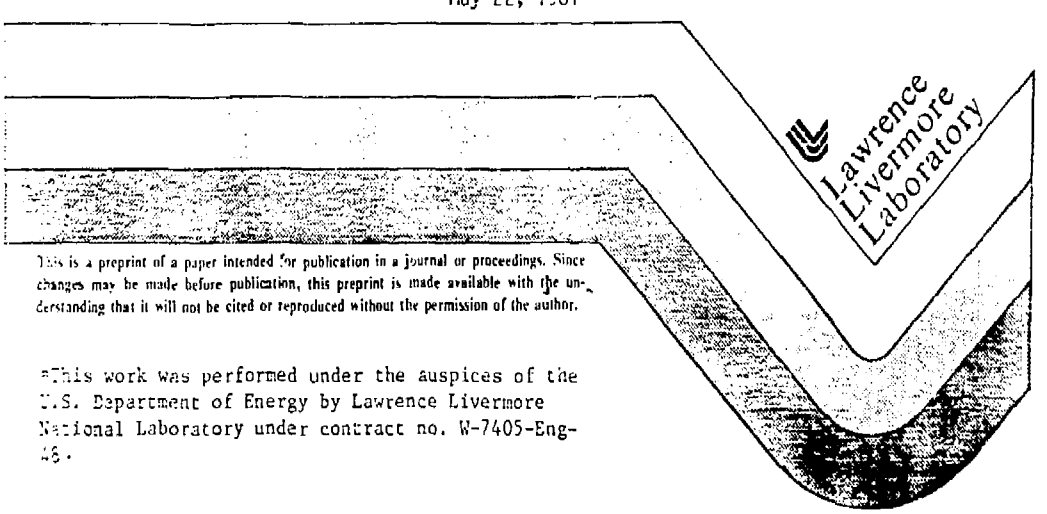
MASTER

The Software Design of a General Purpose Data Acquisition and Control Executive

William G. Labiak
Earl G. Minor

Topical Conference on Computerized Data
Acquisition in Particle and Nuclear Physics
Oak Ridge, Tennessee
May 28-30, 1981

May 22, 1981



This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract no. W-7405-Eng-48.

THE SOFTWARE DESIGN OF A GENERAL PURPOSE DATA ACQUISITION AND CONTROL EXECUTIVE

William Labiak and Earl Minor

Lawrence Livermore National Laboratory
P.O. Box 5511, L-535
Livermore, California

Abstract

The software design of an executive which performs general purpose data acquisition, monitoring, and control is presented. The executive runs on a memory-based mini or micro-computer and communicates with a disk-based computer where data analysis and display are done. The executive design stresses reliability and versatility, and has yielded software which can provide control and monitoring for widely different hardware systems. Applications of this software on two major fusion energy experiments at Lawrence Livermore National Laboratory will be described.

Introduction

The Mirror Fusion Test Facility (MFTF), under construction at Lawrence Livermore National Laboratory, is an experiment to advance the state of the art in fusion energy systems. Data acquisition and control for MFTF will be effected through a two-level hierarchical computer network. The top level of this network, called the Supervisor, is in charge of providing an interface for the operators and of doing required data analysis and archiving. The second level of the network, called the Local Controls, implements the protocol of the instrumentation busses (CAMAC and Modicon Programmable Controller), filters and abstracts instrumentation readings for the Supervisor, and performs those closed-loop control functions which have been dedicated to software.

The hardware for the Local Control level is made up of sixty DEC LSI-11/2 computers. These Local Control Computers (LCC's) are connected to the Supervisor via RS232 serial communication lines. The LCC's are interfaced to the experiment through CAMAC and through Modicon programmable controllers. Each LCC is assigned to one major subsystem, such as a neutral beam power supply, a superconducting magnet or a plasma diagnostic. A general purpose data acquisition and control program, called the Plasma Diagnostics and Local Control Executive (PLEX), was developed to be used in all LCC's. Using common software for all LCC's avoids the need to write special purpose software for each subsystem, and provides a uniform interface between the Supervisor and the LCC's.

PLEX was designed with the emphasis on reliability and versatility. Reliability is promoted by holding the use of interrupts to a minimum. The only interrupt process in PLEX is the communication link to the Supervisor; no CAMAC LAM's or other interrupt signals from the hardware are used. Instead the hardware is constantly monitored as a background task, and a simple scheduling algorithm insures that the sample rate is high enough to detect changes promptly. The lack of interrupts makes the design of PLEX simple, and when the inevitable bugs surface they are much easier to diagnose. Versatility is provided by the command language used to communicate with PLEX, and by a data structure called the Device Table. The basic element in the PLEX command language is called a primitive. There is a small number of primitives, and each primitive causes PLEX to perform a simple action. By stringing primitives together into commands PLEX can be instructed to perform complicated and specific manipulations of the hardware. The Device Table hides the hardware interface from the Supervisor. It associates all hardware addresses and access procedures with tags called Device ID's. The Supervisor controls the subsystem in terms of the Device ID's. This prevents changes in the hardware interface from affecting the way the Supervisor controls the subsystem.

The Design of PLEX

A structure chart showing the basic design of the PLEX software is presented in Figure 1. Procedures are shown in boxes and the major data structures are in ovals. The arrows indicate procedure calls or reads and writes to data structures. Some of the boxes in Figure 1 actually represent several procedures in the final implementation.

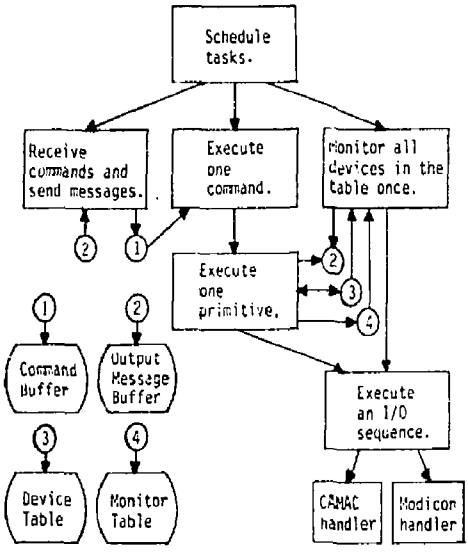


Figure 1. The Structure Chart for PLEX.

The main program schedules the three high level tasks of communication, command execution, and monitoring. The scheduling algorithm is to call the communications task, then to repeatedly call the command execution task until all commands in the buffer have been executed, and finally to call the monitoring task. This sequence is repeated continuously.

The communications task handles all message traffic over the serial line to the Supervisor. It first checks to see if any commands have been received. If a command has been received it is moved from an input buffer to the Command Buffer. The task then prepares to receive another command by initializing an interrupt-driven receive procedure. Next the communications task checks to see if there are messages to be

READ	DevID	*Read DevID and return it's value in a message.*
SET	DevID Value	Set DevID to value.
DUMP	DevID	Transfer the data buffer DevID from the hardware interface to an internal PLEX buffer.
SAVE	DevID	Transfer the data buffer DevID from an internal PLEX buffer to the Supervisor.
ANALOG	DevID MessageID Value Delta	Make an entry for DevID in the Monitor Table. Set the last significant value field to Value and the delta field to Delta. *All messages will have ID MessageID.*
IGNORE	DevID	Remove the entry for DevID from the Monitor Table.
CONFIGURE	DevID *parameters*	Define a device named DevID in the Device Table. *Parameters* is a variable length string required by the I/O sequence for the device. For instance, for a CAMAC device parameters would include the crate #, slot #, and sub-address.
WHEN	DevID Operator Value Timeout	Suspend execution of the command until the value of DevID and Value satisfy the condition specified by Operator (WHEN ATOD Greater Than 500). If the condition is not satisfied by Timeout continue the command with an error flag set.
ENTER	Value	Suspend execution of the command until it is restarted by a "START Value" primitive.
START	Value	*Start a command* which was previously suspended by an "ENTER Value" primitive.

Table 1. Frequently used PLEX primitives.

sent in the Output Message Buffer. If there are, a send is initiated by setting up an interrupt-driven send procedure. The communications are full duplex and fully interrupt-driven, and only need service when a message has completed transmission or reception. The send and receive procedures execute concurrently with other PLEX tasks.

The command execution task carries out the commands received by PLEX and reports on results. A command is made up of one or more primitives. A primitive is the most basic function which PLEX can be instructed to perform. *Each primitive* causes one simple action, such as to read a device, set a device to a value, or make an entry in the Device Table or the Monitor Table (see Table 1). A command, by stringing many of these simple primitives together, can cause PLEX to perform very complicated and specific actions. It is the commands which contain information specific to particular subsystems. Different sets of commands are used to run different subsystems, but all commands are made up of the same primitives, and are executed by the *same version of PLEX.*

After the last primitive in a command has been executed, the command is removed from the Command Buffer and a completion message is placed in the Output Message Buffer for subsequent transmission by the communications task. The completion message contains the command name or ID, the time the command completed, and several flags indicating whether any errors occured during command execution. Some primitives also generate messages, and these are transmitted before the completion message. Not all commands run to completion immediately; certain primitives suspend the execution of a command and leave it in the buffer for execution in a later cycle of the scheduler.

The monitor task keeps a constant watch on selected channels in the hardware interface and reports on changes. It also provides a means for data filtering by only reporting changes that are greater than certain values stored in the Monitor Table. The channels to be monitored and their significant values of change are sent to PLEX in commands. Monitoring is described in detail in a later section.

Devices

One of the most important jobs for PLEX is to hide the complicated hardware interface from the Supervisor. This is done by a data structure called the Device Table and a library of I/O procedures called sequence codes.

The structure of an entry in the Device Table is shown in Figure 2. The entry consists of a fixed length header containing pointers into an array of variable length parameter strings. One pointer is used for reading or input from the hardware interface and the other pointer is used for writing or output. The header also contains an identifier called a Device ID. Primitives access the hardware interface in terms of Device ID's. For example, the primitive "READ CHAN1" causes the Device Table to be searched for the ID "CHAN1". The read pointer is then traced to the read sequence code, which is simply the number of an input procedure in the I/O procedure library.

The input procedure is called, and it fetches the parameters following the read sequence code and uses them to address the hardware interface and perform the input. Some I/O procedures are very complicated, requiring several reads and writes to be done through the hardware interface handlers. These handlers are low level drivers which do reads and writes in the protocol of the CAMAC and Modicon instrumentation systems.

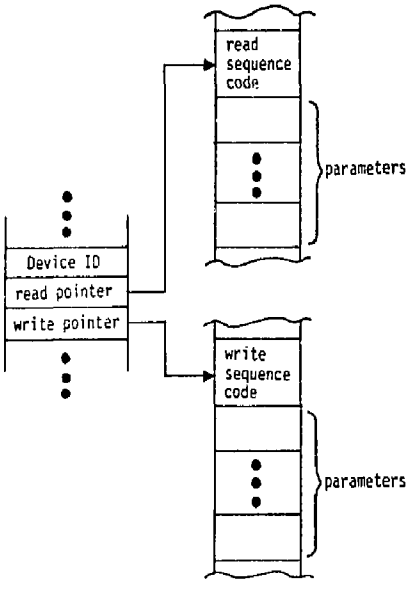


Figure 2. An entry in the Device Table.

The Device Table is filled and modified by primitives while PLEX runs. Adding more devices of an existing type simply requires making entries in the Device Table. Adding new types of devices simply requires adding procedures to the I/O library. This design affords a high degree of control over code development. Since the early days of implementation and debug the only code added to PLEX has been new procedures in the I/O library. This has allowed PLEX to evolve to meet changing system requirements without affecting the basic integrity of the executive.

Monitoring

Almost all hardware subsystems have some channels which need to be constantly watched for changes. PLEX meets this requirement by supplying a function called monitoring. A table of Device ID's for devices to be monitored is maintained in PLEX. Additions are made to the table by the ANALOG and DISCRETE primitives and deletions are made by the IGNORE primitive. The monitor task reads each device listed in the table and compares it's value against a previous value stored in the table. If the two values differ by more than Delta, also stored in the table, a message is sent. Otherwise the monitor task proceeds to the next entry in the table.

Figure 3 is an example of how monitoring works for a typical case. The ANALOG primitive makes an entry in the Monitor Table for device ATODCHAN, which has already been defined in the Device Table. The "last significant value" field of the entry is initialized to 100, and the "Delta" field to 10. Any monitor messages coming from ATODCHAN will have a message ID of 1 (a). Every time the monitor task runs ATODCHAN is read, and it's value is compared to the last significant value of 100. If the two values differ by more than Delta a message containing the Device ID and the new value is queued for transmission (b). Also, the last significant value field in the Monitor Table is set equal to the new value (c). Monitors established with the DISCRETE primitive are similar with the exception that for them Delta automatically defaults to one. This causes any change to generate a monitor message, and is used for devices which have stable values such as contact closure and indicator light sensors.

Monitoring provides a mechanism for constantly watching and reporting on the hardware without using interrupts. By varying Delta the low level random

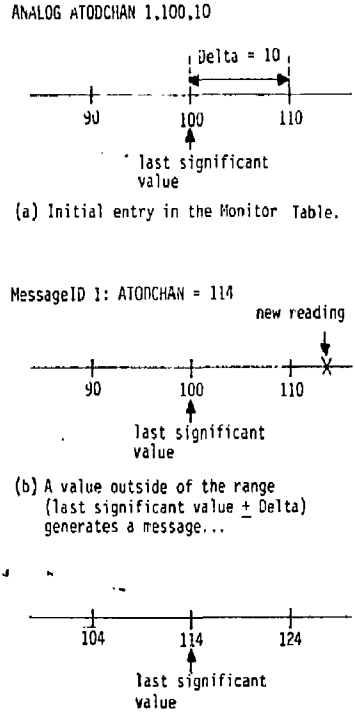


Figure 3. An example of monitoring.

noise from analog channels can be filtered out. Combined with the WHEN primitive, monitoring also provides PLEX with a way to take automatic action on the basis of events in the hardware. This will be described in the next section.

Applications

PLEX was designed to be used in the Mirror Fusion Test Facility, but it's first application was on another LLNL fusion energy experiment, the Tandem Mirror Experiment. This application called for a small stand-alone data acquisition and control system to run a new diagnostic. The system was required to set up CAMAC based instrumentation before each shot and to unload two 4K buffers from a data logger after each shot. The shots were 50ms in duration and were to occur approximately every three minutes. The instrumentation requiring control included the data logger, four amplifiers with variable gain and offset, a variable rate clock, and relays to switch the amplifier front ends between the sensors and a calibrated reference voltage. The data acquisition system consisted of an LSI-11/2 running PLEX and connected via a 19.2 KBaud link to a disk-based LSI-11/23 with a graphics terminal. The usefulness of a general purpose software component such as PLEX quickly became apparent in this application. To a large extent the acquisition and control part of the job was already done, and the programming effort could be concentrated on archiving and displaying the data once it reached the development system. What control programming was required could be carried out in the high level "command language" of PLEX primitives. It was also necessary to add one procedure to the PLEX I/O library for testing the "logger full" flag of the data logger.

The data acquisition requirements of the system were met by the PLEX command shown in Figure 4. The first ENTER primitive causes execution to be suspended (see Table 1), so once the command is transmitted it must be activated by the primitive START 10. The SET primitive resets the data logger so that it will begin taking data when it gets a start trigger. The WHEN primitive then suspends command execution until LOGGER, the "logger full" flag, equals 1, indicating that the logger is finished taking data (other commands, monitoring, and communications continue). Every time a shot is fired the facility provides a trigger pulse, and this is used to start the data logger. When the logger is full the WHEN condition is satisfied and PLEX proceeds to transfer the data buffers to the development system. The final START and ENTER primitives cause the command to be rerun the next time through the schedular loop. The data logger is reset and the command then waits for the next shot. In this way the command loops, sending the data to the development system after each shot, then resetting the logger and waiting for the next shot.

```
ENTER     10
SET       LOGGRESET 1
WHEN      LOGGER EQ 1 32000
DUMP      BUFFER1
SAVE      BUFFER1
DUMP      BUFFER2
SAVE      BUFFER2
START     10
ENTER     10
```

Figure 4. A PLEX command to perform data acquisition.

The remaining control in the system was accomplished through a program which allows PLEX commands to be entered in ASCII and transmitted over the serial link. This program was written as a tool to debug PLEX, and it was possible to modify it slightly and link it into the applications software on the development system. It enables the user to enter simple commands to adjust gain and offset in the amplifiers, set the sample rate in the clock, and switch the calibration signal in and out.

PLEX has subsequently begun to play it's intended role as the Local Control software for the Mirror Fusion Test Facility. The applications on MFTF are much larger and more complicated than the application just described. The helium liquefaction subsystem contains over 600 channels of I/O, grouped in PLEX into approximately 170 devices. The super-conducting magnet subsystem requires the monitoring of 120 devices, 60 of which must also be stored in an internal ring buffer to provide a recent history in case the magnet goes "normal". There are several other major subsystems in the facility, each of them different. PLEX is helping the Supervisor Computer Group to provide a unified control and data acquisition system for all of these subsystems. By presenting the same interface for all subsystems it allows tools and methods developed for one to be transferred to the others. PLEX is also reducing the amount of effort which the Local Controls Group must devote to software. The number of LSI-11s which a single programmer can maintain is proving to be much higher for Local Controls than has historically been observed in LLNL projects. There are two software engineers in charge of bringing up all of the Local Control Computers for the group. Their effort is mainly directed toward writing new I/O procedures for the PLEX library and to providing device definitions (CONFIGURE primitives) to the Supervisor Computer Group.

Conclusion

The task of providing software for the many MFTF Local Control Computers has been accomplished by writing a single general purpose executive. The executive is versatile enough to meet a wide variety of requirements and reliable enough to be easily maintainable. The basic executive design, and the techniques employed for hiding hardware details and for monitoring, are not dependant upon the architecture of MFTF and could be used in other data acquisition and control systems.

Acknowledgements

The authors would like to acknowledge the work of Dr. R. S. Langer in the design and implementation of PLEX. Dr. Langer was a member of the design team from the first design sessions through the application of PLEX on the Tandem Mirror Experiment. Many of his ideas are strongly reflected in the final design.