GENERIC DEVICE CONTROLLER FOR ACCELERATOR CONTROL SYSTEMS*

R. Mariotti, W. Buxton, R. Frankel, L. Hoff
AGS Department, Brookhaven National Laboratory
Associated Universities, Inc., Upton, New York 11973 USA

Introduction

Distributed intelligence for accelerator control systems has become possible as a result of advances in microprocessor technology. A system based on distributed intelligence is inherently versatile, readily expandable, and reduces both information flow across the system and software complexity in each unit.

Overview of the AGS Distributed Control System

A new distributed-intelligence control system has recently become operational at the AGS for transport, injection, and acceleration of heavy ions. A schematic representation of the basic architecture is shown in Figure 1. A brief description of the functionality of these physical devices follows.[1,2]

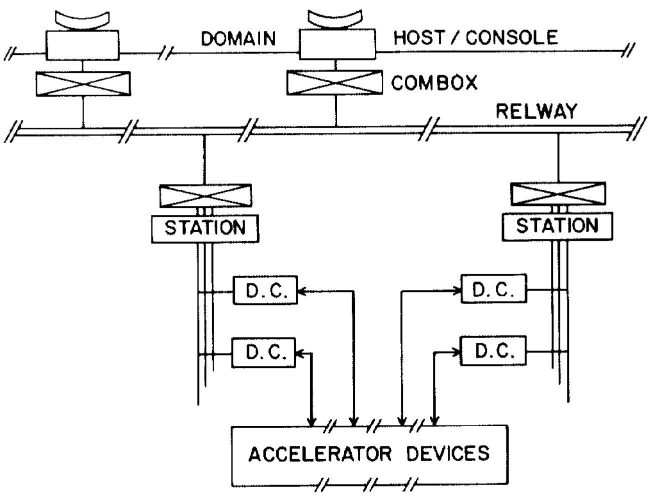


Fig. 1. Schematic representation of distributed control system.

Our system consists of three hierarchical layers: Host, Station, Device Controller.[3]

Hosts act as sources of commands and setpoints, and sinks of readbacks and status reports. Prominent features of the Hosts include their console function and their ability to provide a computational resource. Apollo 32-bit workstations are now employed as hosts/console computers.

A Station polls Device Controllers for information and formats this information into reports when requested by the Host; generates unsolicited alarm reports by use of "watchdog" tables downloaded from the Host; forwards commands and/or setpoints generated by the Host to the Device Controllers. The Station is cycled synchronously and has parametric limits on the period within a machine cycle when commands may be sent to, and information acquired from Device Controllers. The Station also appends or strips network level destination protocol headers.

Device Controllers have the function of acquiring information and/or controlling accelerator devices or equipment.

Every physical Device Controller has available to it information which must be arranged into data structures representing "logical devices". This fundamental data structure, resident in the Device Controller consists of command/status fields, setpoint fields, and readback fields.

According to the type of information, the data base contains "passive" information and "live" information. The passive or static information does not change in real time and consists of a series of tables (relations) which allow user modules and system utilities to access information on the network. Examples are names of logical devices, network addresses, allowed status, etc.

Data which changes in real time under operator control forms the "live" or dynamic data base and is resident in memory distributed in the network, i.e., in the Station and Device Controllers.

The Universal Device Controller Concept

The design of modern accelerator control systems is generally network based and has hierarchical layers. The top layer consists of the computers which support the operator consoles and provide high level programming facilities for general control operations. The middle layer consists of the microcomputer systems. Most device connections are made directly to this layer. the lowest layer consists of equipment which contain interface hardware, but may not have intelligence.

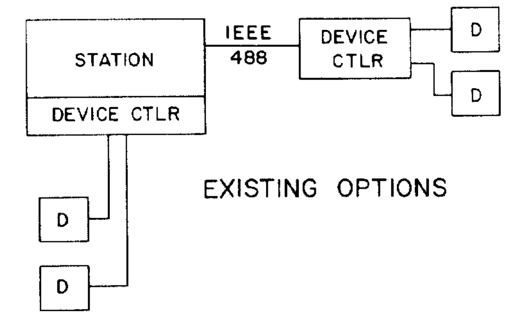
General tendencies in electronics lead to the distributed intelligence concept. We implemented that concept at the lowest level of the control system, with intelligent interfacing. An attempt has been made to integrate the devices for accelerator-specific interfacing into a standard microprocessor system, namely, the Universal Device Controller (UDC). The synchronization, timing mechanisms, and computer capabilities of the control system have been concentrated into the interface level (Figure 2).

The UDC Assemblies

The main goals for such a generic device controller are to provide: (1) local computing power; (2) flexibility to configure; (3) real time event handling.

(1) Signal adaptation, data conversion, and data translation are made possible by the UDC processing power. The controller must execute the lowest level

*Work performed under the auspices of the U.S. Department of Energy.



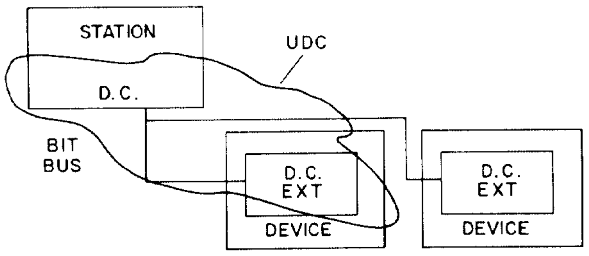


Fig. 2. The UDC Enhancement.

software. Application software and User Service Routines must be relieved of equipment-specific details which should be dealt with by the intelligent interface. The UDC must be able to control the attached equipment autonomously as long as the operation mode does not change. Test and diagnostic routines and a terminal interface should be provided at the UDC level in order to run them locally at that level.

(2) The main feature of this controller is its universality as programmable peripheral interface capability. The appropriate code downloaded from the central control computer will configure the controller according to the equipment it is to control. A menu-driven procedure will use static data base information contained in "device descriptor files" in order to initialize the controller and set up the actual operating conditions. As an example, the same standard hardware can be configured as a power supply controller or a vacuum controller and the operation parameters can be set up. The needed transfer function, written in a high level language will be coded in short and easily generated (command) files. The imbedded information fields form an ordered list consisting of commands, statuses, setpoint/readbacks, and actual operating conditions. A UDC resident interpreter will translate that information from the device definition files in commands to the device controlled. The term "universal" applies to the functional capability of the package which will be built as a compatible family of modules. It is also necessary to have a mechanism to be able to download programs in lieu of parameters.

(3) Real time synchronization is accomplished at this level. Many industrial process control systems provide programmed transfer functions which relate input to output states. The accelerator control and data acquistion system requires also the ability to generate and receive functional patterns based on time or events. The controller must perform the process synchronization autonomously, so that the demand for real time ability in the higher level computer is reduced. As an example, external events can activate or deactivate high speed DMA channels when we need to acquire data or generate functions; delay features can also be implemented. The connection between a channel and a specific event interrupt is determined by hardwired jumpers.

Figure 3 shows the prototype controller assembly. The dual processor structure consists of a Communication Processor and the Device CPU. The dual ported RAM allows them to communicate and exchange data. The communication processor ensures link-up with the station using a communication protocol. The Device CPU provides data acquisition, data processing, and equipment control.

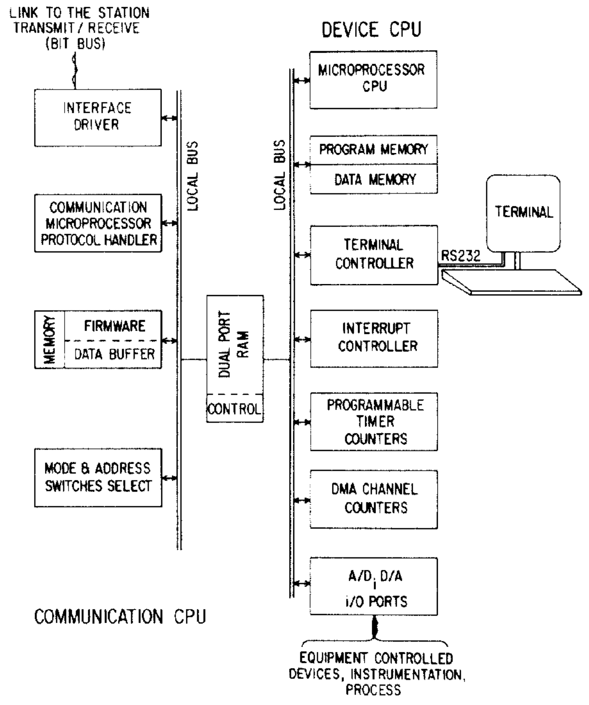


Fig. 3. The UDC Prototype Assembly.

It can be assumed that the UDC does not require a master-to-master interconnection and that a master-slave relation between a parent station and its UDC nodes is satisfactory.

A Software View; The UDC Software Tool Kit

Software tools should be provided for a device controller designer in order to make the job of writing and debugging the UDC software more efficient. The highest level tools would include a configured OS with all the required tasks and drivers as well as a method to load the code via the controls network. The low level tools provided would be macros and routines to interface with the standard report and the controls I/O hardware. A view of the UDC software follows.

There are three standard tasks in the UDC: the scan task, the receive-commands task and the send-standard-report task (Fig. 4). The receive-commands task and the standard-report task are the same in every UDC. For most UDCs there are also driver tasks which are used for theI/O to the devices being controlled. This driver might interface to a network

(MAP), a bus (IEEE 488), a port (RS 232, DMA), or memory. The drivers and their tasks are some of the tools provided for the device designer. The UDC user will be able to interface to the driver without knowing details of the I/O operation.

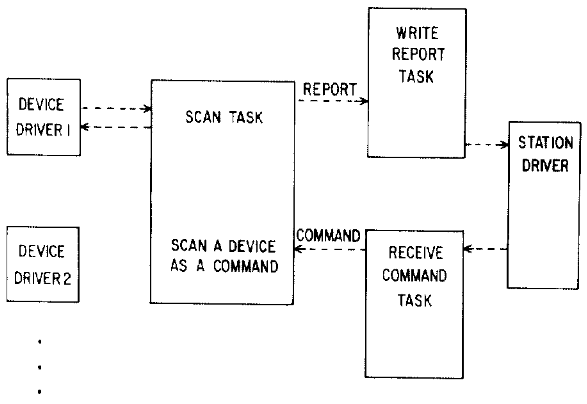


Fig. 4. Standard Tasks in the UDC Software.

The scan task has a form that is common to all device controllers. The details of the routines that are called from the scan task must be defined by the device designer. The scan task has an initialization section which sets up the I/O hardware (ADCs, DACs, I/O ports, timers DMAs) for the particular application. After the initialization section, the scan task goes into a loop forever. The function of the main loop is to scan each device and fill in the standard report. This main loop also takes care of the buffer switching among two or three standard report buffers. After each device is serviced there is a routine which looks to see if a command has been sent to the device controller. If there is a command in the queue, the command is processed.

The receive command task, which is the same for all controllers, just receives commands from the station and puts them in a queue. There is a routine in the scan task which can look into the command queue to see if there are commands in the queue.

The send-report task sends a report to the station when the device controller is made a talker. It sends the last buffer filled in by the scan task.

The standard-report data structure, init-hardware, and scan-a-device are unique for each UDC type, but have a common form. They are built by the device designer using the low level UDC software tools.

The low level UDC software tools would make it easy for the device designer to do the following: (1) set up the I/O hardware, (2) define the standard report data structure, (3) define the actions to take in filling in the standard report for each device in the scan, and (4) define the actions to take when processing a command for a device.

The translation between the I/O hardware readings and the format of the standard report and the translation from commands to the I/O hardware is the heart of the device controller. All of the UDC software tools come into play in assisting the device designer in writing the code to do the translation functions. Data structures, names, and I/O routines are available to make the translation code easy to write and read.

We picture the UDC running a multitasking OS much like RMX 88 which is now in use in the station and device controllers.

Status

For the prototype design of the UDC, a simple 8085 microprocessor has been used. The peripheral circuits include I/O ports, DMA channels, timers/counters, 32K of program memory, 16K of data memory, an interrupt controller, and a terminal controller with a RS232 interface. The PROM resident monitor and a XYBASIC interpreter create a friendly environment to write tests, and to exercise the controller and the interface from the terminal. The same RS232 interface is also used as a port to download/upload programs.

For the link and communication CPU, the Intel's BITBUS protocol and components have been used. The BITBUS uses SDLC standard and interconnect segments up to 300 meters in length at 375 Kbits/sec and 1200 meters in length at 62.5 Kbits/sec. From the user point of view, the twisted pair link to the UDC behaves and is completely compatible with the Intel's iRCB 44/10 Remote Controller Board. Using Intel's 8044 Controller and the iRMX 51 Executive in preconfigured firmware, the actual design of that part of the UDC is emulating Intel's product.

A key component in the UDC system is the Dual Ported RAM that ensures the data communication between the Device CPU and the Communication Processor.

The UDC prototype was intended to be only a testing and debugging system. The assemblies for the AGS will most probably use the Motorola 68000 microprocessor family or Intel 80186 in a VME system. Aside from the BITBUS for the communication protocol, there are two or three candidates: the MAP system, token-ring approach based on the TM380 chip set, or the MIL 1553 data link.

The original idea in the design of the UDC software was to progress toward the resident interpreter in the UDC that would be programmed in some new language. Once the UDC Software Toolkit is well developed, such an interpreter can be built. The main problem is to define a distributed data base that is universal to all accelerator subsystems and easy to understand.

References

1. R. Frankel, Tailoring a CSMA Local Network to Research Needs, Data Communications 9, 145-154 (1984).
2. J. Skelly, T. Clifford, R. Frankel, A Broadband Accelerator Control Network, IEEE Trans. Nucl Sci., NS-30, 2155 (1983).
3. A. Stevens, T. Clifford, R. Frankel, Distribution of Computer Functionality for Accelerator Control at the Brookhaven AGS, presented at the 1985 High Energy Physics Conf. on Particle Accelerators, Vancouver, B.C., May 13-16, 1985 (in press).