Status of activities at NCBJ on the slow-control for the PANDA cluster-jet target

Jerzy Tarasiuk
NCBJ / University of Warsaw

A few word about myself and my experience

Since I am a newcomer to the Detector Slow-Control Group, let me tell you a few words about me and my programming experience.

After Dr. Andrzej Trzciński died in January 2019 I was asked to join the NCBJ team and continue Andrzej's work on the PANDA Cluster-Jet Target Slow Control.

I work in the Physics Department, University of Warszawa since 1980. Since 1986 as a programmer, although I did a lot of programming before that date, helping people in the Department whenever they needed help in something exceeding their capabilities - from 1986 I am formally employed as a programmer there.

I studied there in 1975-1980, and I have over 40-year experience in computer programming, as I wrote my first computer program in Fortran during student internships (and the program was included into a larger program used for scientific computations then).

I was also a passionate of electronics since my school years.

And since then I had a continuous experience with computer programs - since the beginning of 1979 I worked (as a volunteer) in CYBER-73 (CDC-6400) system analyst group at CYFRONET (Świerk), starting eith a work on fixing mainframe-terminal communication problems there. Later I used my knowledge on the communication protocol to write a CYBER terminal emulator (Intercom Mode II) program for a PC.

As I work in Physics Department, much of my programming work was for physics experiments - some of examples I remember the best now are:

* data acquisition for Cold Fusion test done in the Department;
* monitoring forces and displacement in the ZEUS detector at DESY;
* monitoring ion drift speed variations in a drift chamber at CERN;
* control of etching of thin (few um) silicon wafers at ŚLCJ UW; with a few version of the apparatus. I planned computer interfaces and wrote programs to operate them.

The second and the last had complexity near that of the Beam Dump: the last had 16 analog inputs and 64 digital outputs for simultaneous control of 16 devices; the controlling computer was HP T5720 terminal.

I was writing in assembly for many architectures: CPU, PPU and BC on CYBER, 6502, 6809 (and 68k), 8088 (and later), Cortex...
and in higher-level languages: Fortran, Pascal, Lisp, C, Tcl/Tk.

LabVIEW is new for me, I started learning it in February 2019.

I had to spend some time to acquire knowledge what LabVIEW can do, and what I am to do to make it doing what I need.

The Beam Dump control program - design assumptions

To my best knowledge, the program is required to:
* run on cRIO alone (including auto-start on reboot/power-on)
* provide status information via EPICS (being an EPICS server)
* get operator commands via EPICS (being an EPICS client);
- therefore these are the basic assumptions of my design.

Another: I want to avoid - as far as it is possible without requiring significantly more work - encoding into the LabVIEW program information about hardware configuration, like which valve is connected to which wires - any mistake made when writing a program using such a technique is hard to notice, and any change requires changing the program itself.

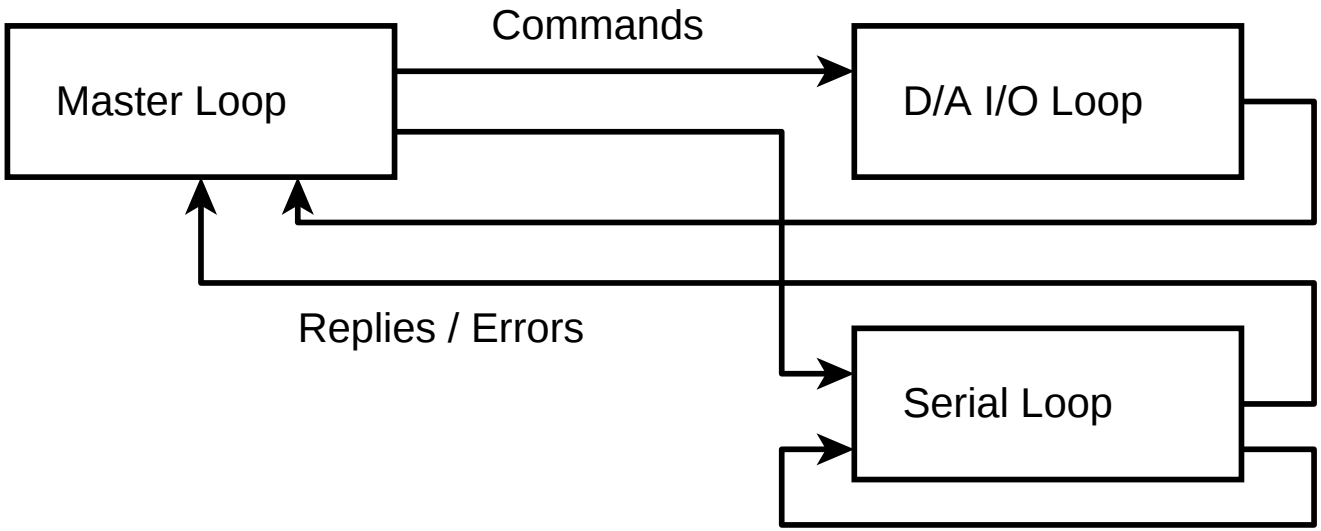
Instead, I am putting such an information in a text file - or more precisely, a spreadsheet in tab-separated-values format.
This way the connection information is separated from the program, allowing its independent verification and correction.

The file constitutes a hardware configuration definition.

For a valve, or forepump, the configuration contains (in one record):
* a valve/pump name (used to identify it in communication)
* DO signals to be used for "on"/"off" actions and their polarities
* DI signals to be used to detect its state by the program, and
  their polarities (if one signal tells both "on"/"off" state,
  it is specified twice with different polarities);
* pulse time (0 if the device is controlled by signal level)
* busy time (it is a time for the device state to stabilize; before
  it ends, its state signals are unreliable and are to be ignored)
* EPICS variable name(s) (if its state is to be reported)
- all these in keyword/value form, with keywords:
  valve/pump (one of these two), on_sts, off_sts, err_sts,
  on_cmd, off_cmd, pulse, busy, epics.

A valve defined this way may show the following "control" states:
* closed/stable -or- open/stable - need a command to change it
* opening/pulse -or- closing/pulse - lasts pulse_time, than
  the DO line returns to its base state and the next state is
* opening/busy -or- closing/busy - lasts busy_time, and when
  the time ends its state is verified to match the requested one.
* opening/fault -or- closing/fault - these indicate an error.

The overall program structure



The Master Loop controls the two others - sends command to them, receives their replies and (possibly asynchronously) error signals.

These other loops control devices connected to cRIO:

* D/A I/O Loop controls devices connected to DI (digital-in), DO (digital-out), AI (voltage-in), CI (current-in) pins of cRIO modules;
* Serial Loop controls devices connected to a serial port (one - if more serial ports is used, there will be more serial loops).

The entire program uses cRIO Scan Mode (no FPGA programming).

In the D/A I/O Loop elementary actions (reading line state, setting output line high/low) are (for the program) almost immediate (their effects are slightly delayed: outputs until the next scan, inputs are from the last preceding scan) - and actions can be parallelized - an action affecting one device do not stop action affecting other. The loop counts time for delays (every device has separate timer) to output pulses of defined time (time of the "pulse" state), and wait before examining the device's status after a command ("busy" state) - but doing this all "in parallel", in a quick internal loop.

In the Serial Loop any action affecting a device forbids any other device actions until the first action gets a complete reply from the device, or times out. It has one timer - counting the time-out for the device it awaits reply for.

Both these loops frequently (with a period shorter than a pulse time or getting a reply from a serial device) examine master's commands to reply to them in a short time (although, execution of a command requiring a device action can be delayed in any of these loops - even a valve open/close pulse is forbidden in its pulse/busy states).

Also, both loops are acquiring device status periodically, each in its own pace: the D/A I/O Loop can read everything in much less than a millisecond, but since it reads scanned values, the read result can change at a scan time only; the Serial Loop needs much more time to get all statuses, as sending commands and receiving replies for all important values (pressures, turbopump speeds and temperatures) needs about a second alone (assuming every device replies immediately). Andrzej Trzciński used 2-second spacing between serial commands and 90-second period for looping over all queries sent to these devices.

And they both are capable (if a device configuration specifies it) to put these statuses they receive in local EPICS server PV-s.

The Serial Loop puts requests to itself - just these reading statuses - and processes them unless it has other request from the Master Loop. Its configuration specifies a list of these requests and the commands the Master can send - how they are translated to device requests.

While the D/A I/O Loop is more than enough quick, the Serial Loop 90 seconds time seems to be slow, even for the Slow Control purposes.

I wrote a serial input (device->program) in a way that every received byte is put in a buffer and contents of the buffer are examined to see if it contains a complete reply - if yes, it is processed at once, and the serial port is free for another device action.

However, a lack of device reply will cause a long delay anyway.

The ICPDAS i-7521 devices affect the serial communication limitations in many ways. While they are commonly assumed to be RS485<->RS232 converters, they also can buffer data and transmit it using different baud rate; if the device reply arrives too late, they store the reply and allow it to be later fetched by a host command.

However, they introduce extra delays, since they wait for end-of-record before retransmitting it. Also, a Center Three readout needs sending ENQ char - if an i-7521 is configured to used LF-ended record, it will send ENQ LF instead ENQ - I do not know, will the Center Three ignore the LF? It is possible to send ENQ followed by a command to the Center Three - then the LF will be added after the command where it can be.
Or the Center Three can be configured to use time-out as end-of-record.

The Master Loop

While both device loops control device accesses and actions affecting their devices, this loop focuses on more general Beam Dump states.

The following requirements are for the Master Loop:

* it sends initialization commands (read from a file) to device loops
* it has its own configuration, defining possible Beam Dump states
* every state must have configured, as its entry actions, commands to be send to device loops for all devices
* state may have configured a timeout and its state transition
* state may have configured lists of device statuses and conditions they are to meet for entering another state and for staying within this state (if none of these is met, an error state is entered)
* it publishes its state via EPICS server
* it monitors EPICS variable (as a client) to get operator's commands
* it receives error signals from device loops and takes appropriate action (if a valve open or close command fails, it may be retried few times; a failure to perform the requested action, and at least some other errors cause a transition to the error state)
* it saves on a disk last state entered and current operator command to be able to restore its state after cRIO restart (but a change of the operator command must cause transition to initial state)

What has to be tested?

* i-7521 timing - what are its transmission delays, can an immediate device answer to be buffered and forwarded when host requests it - to be done in Warszawa
* device protocols - I do not know if Andrzej Trzciński tested them with real devices, as these test programs I found seem to connect to Raspberry Pi using slightly different protocol than the device needs - possibly these test programs could connect to real devices with help of an FPGA program (accessed via TCP/IP protocol), but for now I cannot confirm it; also, possibly these programs were based on other programs which were already tested and found correct
* device access timings - in what time a device replies via serial communication, in what time its status changes (for digital I/O).

Thank you!