Detection and reporting of faults in comunication between LabVIEW and EPICS in Cluster Jet control system

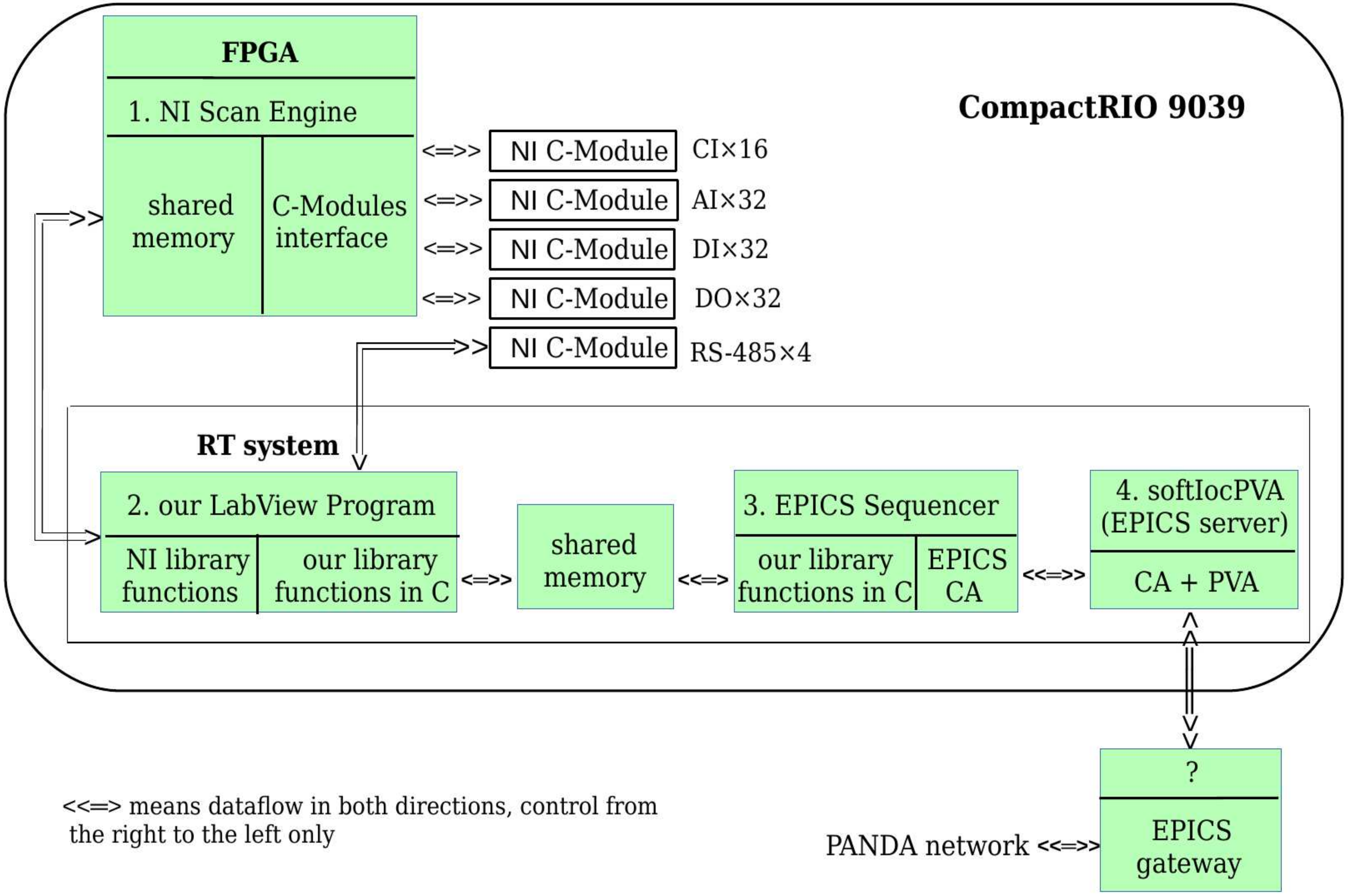
Jerzy Tarasiuk

NCBJ - Warsaw and Uni-Warsaw

The Cluster-Jet Target control system is to use 2 Compact RIO-s (short cRIO, RIO = Reconfigurable I/O; now we have one of them) NI-9039 type, with C-Modules constituting I/O interfaces; the cRIO has NI Linux Real-Time 6.0 with LabVIEW Real-Time 18.0.0, and NI Scan Engine (which "enables access to I/O channels by storing data in a global memory map and updating all values at a single rate" by NI's words, in fact it also signals errors in I/O operations - it was tested by removing a C-Module).

The LabVIEW Real-Time includes "EPICS Server I/O Server", which provides some EPICS Server functionality, but far from being suitable for our purposes. I presented some of its disadvantages on PANDA CM 3/2019 (as well as our plan to use standard EPICS server and NI's EPICS Client only), some more were told us about by Florian Feldbauer, and were described by Dirk Zimoch in 2013 on EPICS tech-talk.

Yet another thing (from Florian's dissertation) triggered a change in our plans: SNL (State Notation Language) and EPICS Sequencer - we planned designing something similar in the LabVIEW, but now we can use the original.



<<=> means dataflow in both directions, control from the right to the left only

Compared to the previous plan it has added the EPICS Sequencer, and a shared memory interface between LabVIEW program and the Sequencer. In order to propagate errors from LabVIEW to EPICS need store there not only the data, but also a kind of status.

To detect a communication failure the status must contain a positive "OK" information, not an "Error" information only, as in the latter case "OK" is the same as "connection lost".

Our idea is: the LabVIEW side stores 0 as status value after storing a valid data; the Sequencer stores a non-0 after it processed the set of data and waits for another one. Further, the Sequencer may use the value to count the time it waits.

This applies to the global communication status; every value has its own status, and before setting the global status to 0 the LabVIEW side sets these individual statuses: if a data is OK, it stores 0; on an error, it adds the global status value
- this way any data item status tells how long the Sequencer was waiting for the data since it received some; 0 = "fresh".

For writing a data from the Sequencer to the hardware there is a command/status for every data item: a non-0 requests the data to be written, a 0 tell the writing was done.

The Sequencer examines these statuses and if none of these shows an unacceptable delay it sends some value to a "timeout" record on the EPICS server (currently, it is 12 (seconds) in our stub program); it is a "calc" type record scanned every second, which subtracts 1 from its value, and has LOW=0 - when the value reaches 0, it triggers a warning.

Also, the Sequencer uses fresh data only to send any data to the EPICS - if it isn't fresh, its EPICS timestamp remains unchanged and it shows there is a problem with a data item.

Yet another thing is planned (and a space for it is reserved in the shared memory): both LabVIEW and Sequencer are to put current time (each has a separate variable for it) and they may compare these times - a significant difference will mean a stall of one of these programs and the other can restart it.

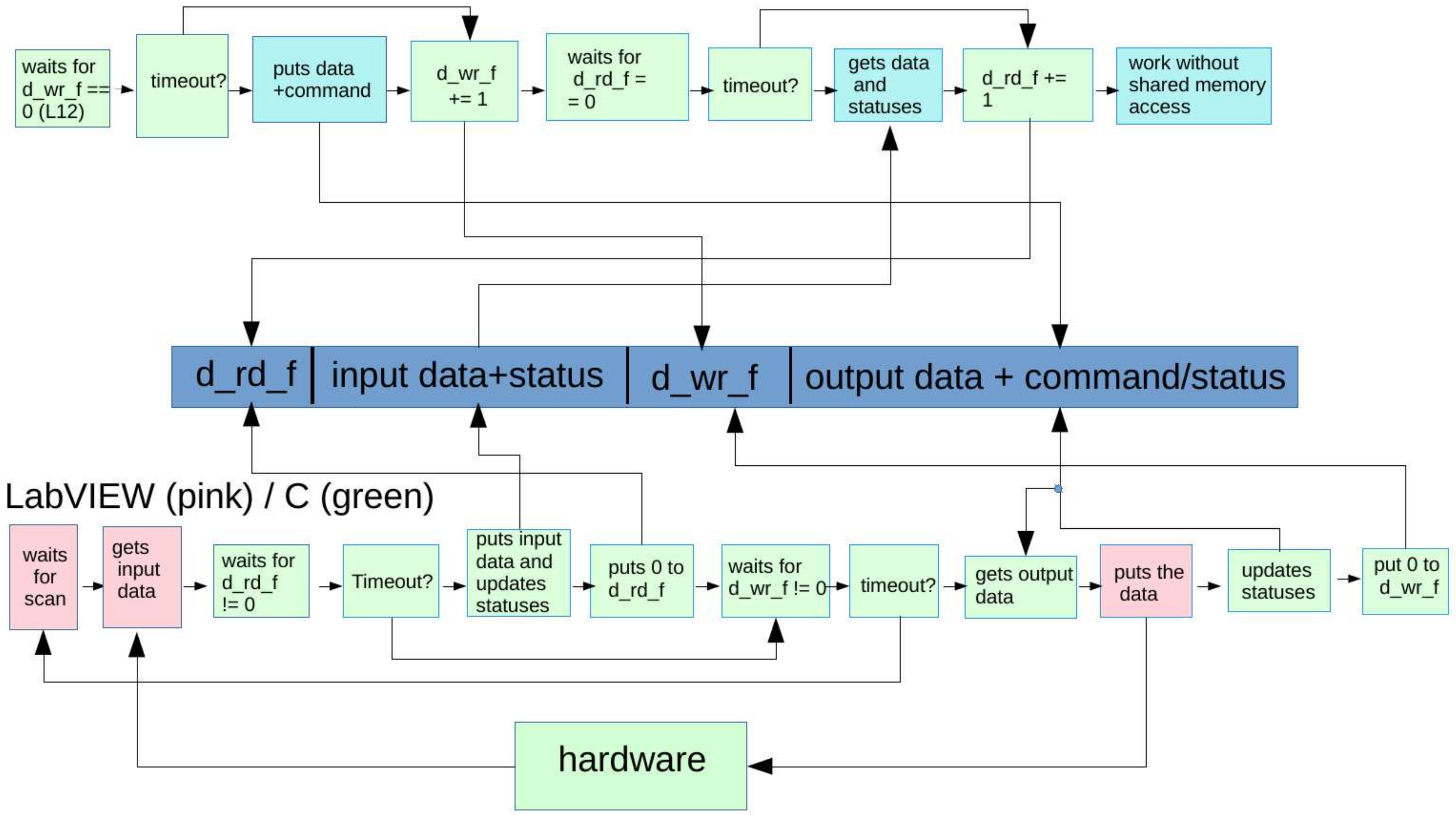
The LabVIEW part can handle Analog Input and Output (AI and AO) signals (data type = Double or 8-byte Real), and Digital Input and Output (DI and DO, data type = Boolean stored in byte); the shared memory contains 6 arrays for the data (output data needs separate arrays for output and for input, as it can be read back to the Sequencer); and it is accomplished by 6 status arrays (control/status for output data). If there are more modules providing the same kind of signals (currently we have 2 DO modules), they are put in the same array, and the order of the data is determined by a configuration file telling also setup values for some modules (NI-9205 has a lot of settings).

Besides these arrays, the communication uses two variables, named d_rd_f and d_wr_f, for synchronization (both programs are doing their job in their own pace - LabVIEW synchronized with the Scan Engine), and the shared memory contains also array offsets (12 total) and sizes (4).

On the next page there is a schema of their coordination.

Sequencer (cyan) / C (green)

LabVIEW (pink) / C (green)



The rules for the synchronization are:

- d_rd_f rules read (hardware -> Sequencer) transfers;
- d_wr_f rules write (Sequencer -> hardware) transfers;
- the Sequencer may only increment d_rd_f/d_wr_f values;
- the LabVIEW part may only clear these values;
- when d_rd_f/d_wr_f is 0, the Sequencer has access to all data and command/status elements in the shared memory for the respective direction; when finished, it should set non-0 there to allow access for the LabVIEW part;
- when d_rd_f/d_wr_f is non-0, the LabVIEW part has the access; it should set 0 there when it finishes;
- a sum of the d_rd_f/d_wr_f and an individual data item status tells how old data is.

The Sequencer has access to the status information and it is not to use any old data, so an I/O error detected by the LabVIEW part results in old timestamp in EPICS; also, when all is functional, it updates the "timeout" record - so stopping the Sequencer will trigger a warning/alarm.

An I/O error may be caused by stopping the LabVIEW process, or by pulling out a C-Module - both were tested and they resulted in a warning and later alarm in the EPICS.

The softIocPVA used was from EPICS base version 7.0.4; the Sequencer was of version 2.2.7; an example Sequencer program generated by makeBaseApp.pl was adapted for the purpose of these tests (everything except the SNL program was removed, a shared memory interface was added, as well as forwarding to EPICS all the input data which was read without an error) - it is a minimal stub for testing the error forwarding.

Besides analog and digital data, the shared memory can forward data from/to serial ports, using circular buffers; the LabVIEW part has an advantage of configuration file (in Windows .INI format) support, detection and configuration of serial ports.

Limitations: we cannot test results of FPGA malfunction, as the LabVIEW seems to have no command or function to simulate it, althought pulling out a C-Module has shown that some error propagation from the FPGA to LabVIEW RT exists.

If stopping of the FPGA (e.g. due to a clock failure) does not cause an error to be returned by the Scanned Variable Read or Scanned Variable Write function, nor it causes a detectable malfunction of scan synchronization, such an error stil can be detected by comparing consecutive data values from an analog input - due to noise, they usually vary, showing the data is really read from a hardware, not from a saved old value.

However, detecting a broken wire between the C-Module and the signal source is not possible by a purely software method.

And, from the EPICS side, there is needed a client monitoring a variable that would show a loss of communication. More, some server variable must contain a kind of clock, and be monitored to detect possible loss of network communication.

THANK YOU FOR YOUR ATTENTION