EPICS CHANNEL ACCESS IMPLEMENTATION IN LABVIEW

A. Zhukov, W. Blokland, R. Dickson, ORNL RAD, Oak Ridge, USA

Abstract

LabVIEW is becoming increasingly popular in Accelerator Control Systems. Interfacing of LabVIEW with EPICS could be done in several ways. We provide a native LabVIEW implementation of the CA protocol. Such an approach greatly simplifies the development of EPICS[1] controlled devices in LabVIEW.

INTRODUCTION

The Spallation Neutron Source (SNS) is a particle accelerator driven pulsed neutron source. The Experimental Physics and Industrial Control System (EPICS) is used as control system for the accelerator complex. There are more than 300 devices being controlled by Windows based PCs. These include: Beam Current Monitors, Beam Position Monitors, Wire Scanners and more. National Instruments' (NI) LabVIEW is used as the software development environment for these PCs. To integrate with EPICS clients, e.g. operator screens in the SNS control room, a communication interface between LabVIEW and EPICS is required.

DIFFERENT TYPES OF INTERFACES

Shared Memory

Currently our main method of connecting LabVIEW with EPICS is the EPICS Shared Memory Interface[2]. A DLL provides device support by sharing memory with the LabVIEW process. Thus we have 2 relatively independent processes running simultaneously: EPICS server and the LabVIEW runtime. This approach has obvious advantages:

- Having a standard full EPICS Input/Output Controller (IOC) server allows leveraging the power of records processing and the features that comes with it.
- All Channel Access (CA) clients work seamlessly because clients "see" a standard IOC.

This mechanism also has disadvantages:

- The implementation is OS specific. Although EPICS itself is ported to different platforms and LabVIEW has versions for different platforms this binding layer has to be implemented for every OS. While this is surely possible, a big effort is needed.
- One needs to support two mapped data structures: IOC records and LabVIEW variables.
- The installation and maintenance of versions of the EPICS Shared Memory IOC and LabVIEW libraries is required.
- Having two processes makes debugging more difficult than having a single process.

CA Server

Alternatively, instead of having a full IOC, one can use the CA server only. It won't have record processing power, but will publish data onto the network. All standard CA clients will be able to communicate with this. While there is a C++ version of the CA server, it is possible to implement a CA server in any language that supports TCP/IP and UDP/IP communication. A good example of this is the Java CA library developed by Cosylab[3]. Since LabVIEW has full functionality for TCP/IP we tried to implement the CA protocol specification internally in LabVIEW.

NI Datalogging and Supervisory Control Module

NI has developed several implementations of the EPICS CA server for different OSs. It is part of the Datalogging and Supervisory Control Module [4]. It follows the concept of shared memory method. So, for example, the Windows version requires installation of additional Windows service.

PURE LABVIEW CA SERVER

Requirements

Careful analysis of our EPICS connectivity needs led to the following requirements:

- We need a CA server, not a full IOC (since all logic is implemented in LabVIEW code rather than in EPICS record processing)
- The server should be able to host about 2000 PVs. We are not expecting a single system to multiplex with large numbers of detectors. Our most typical use case is one device per IOC (e.g. every BCM is controlled by dedicated PC).
- The data transfer rate we need is about 10 Mbit/s (maximum 50Mbit/s for video systems)
- The server has to be compatible with all platforms LabVIEW runs on: Windows, Linux, Mac OS, LabVIEW Real Time on CompactRIO and PCs, and maybe even on embedded systems.

We never use record processing so we can easily give up this functionality of IOC.

In order to evaluate possibility of creation of CA server in LabVIEW we set up a pilot project that doesn't fully comply with CA specifications but still allows us to estimate the performance of such approach. Here are main features of our pilot server:

- No CA Repeater (we probably won't implement it anyway because we will hardly ever have several IOCs running on one host)
- No beacons
- Support UDP channel search

- Support two CA types (FLOAT and LONG)
- Support DBR_STS and DBR_TIME
- Support put, get and monitor

Server Architecture

We used the LabVIEW internal threading mechanism as much as possible during the design of the server architecture. Rather than using classical threads we use simultaneously running VIs that use queues for sharing data. Of these data, there are two main types: Virtual Circuits (VC) and Records. The VC is the same concept as a VC in the standard CA library, i.e. the data supports a TCP client connection to a server. The record data is not a fully functional EPICS record, but does hold an up to date value and the state of the record. Every VC can be in either a communication state or processing state. This state information is maintained by the presence of the VC data in one of two different queues. So switching states effectively means placing the VC data in a different queue.

There are two types of "worker" VIs: Communication VIs and Processing VIs. These will run as separate tasks if LabVIEW uses parallel execution. Communication VIs get a VC from the communication queue and check if there is new data available in the socket. If yes, it reads the data and places the VC into the processing queue. The processing VI gets VCs from processing queue and processes it by updating the record data and placing the VC back onto the communication queue. A typical configuration would launch several (three as on Fig. 1) of the communication VIs and one processing VI. This effectively allows reading simultaneously by different TCP connections, so if one connection is having trouble it will not affect the workflow. It is possible to launch as many communication VIs as one wishes (as long as the OS gives LabVIEW the resources), but it stops making sense if the number of "workers" is higher than the number of connections (VCs) to the server.

In addition to these two "threads" one has to launch a TCP listening VI and two UDP related VIs. The first listens for TCP connections and if a connection is made it creates a new VC and places it into the communication queue. The UDP VIs listen for UDP search messages and reply to them. Receiving and sending are performed by two different VIs.

Demo Program

This demonstration program initializes the CA server, creates FLOAT waveform, FLOAT scalar and LONG scalar records. Then it starts the server that initializes all the queues and starts the VIs described in previous section.

After that it enters a loop that generates a sine wave with random noise and publishes the waveform and its integral.

Figure 2 shows a diagram of this program. One can see that the end user part is quite straightforward and self-explanatory. Records are created within LabVIEW and do not require any EPICS knowledge to configure. This allows a developer completely unfamiliar with EPICS to quickly publish data from his/her LabVIEW program onto the network.

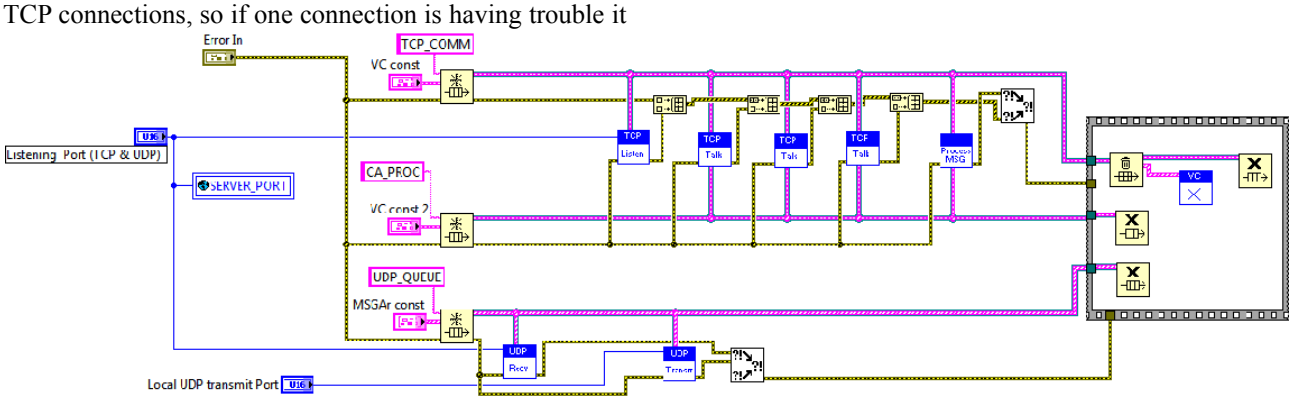

Figure 1: CA server configured with three communication VIs ("TCP Talk") and one processing VI ("Process MSG").

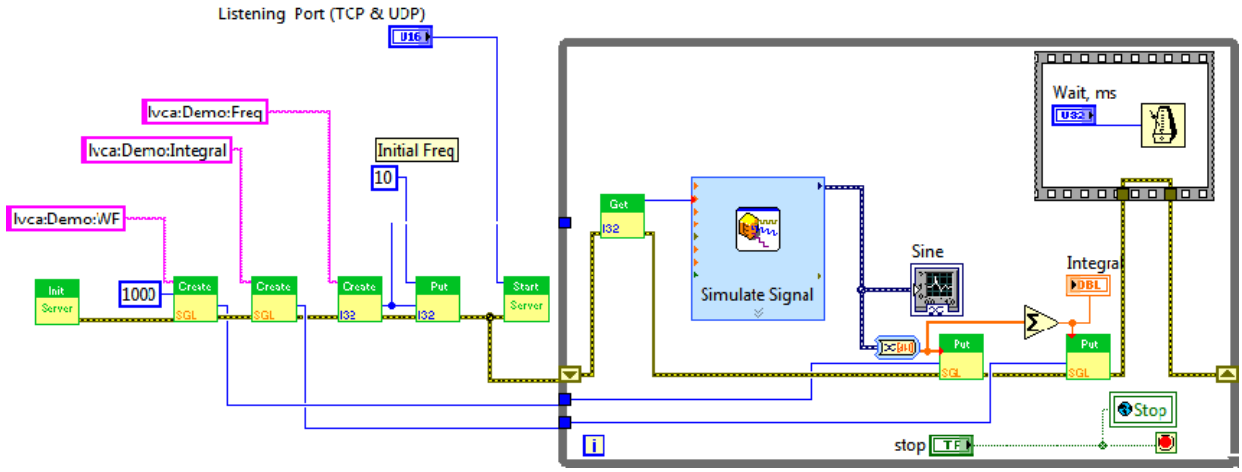

Figure 2: EDM screen (right side) displaying data from LabVIEW server (left side).

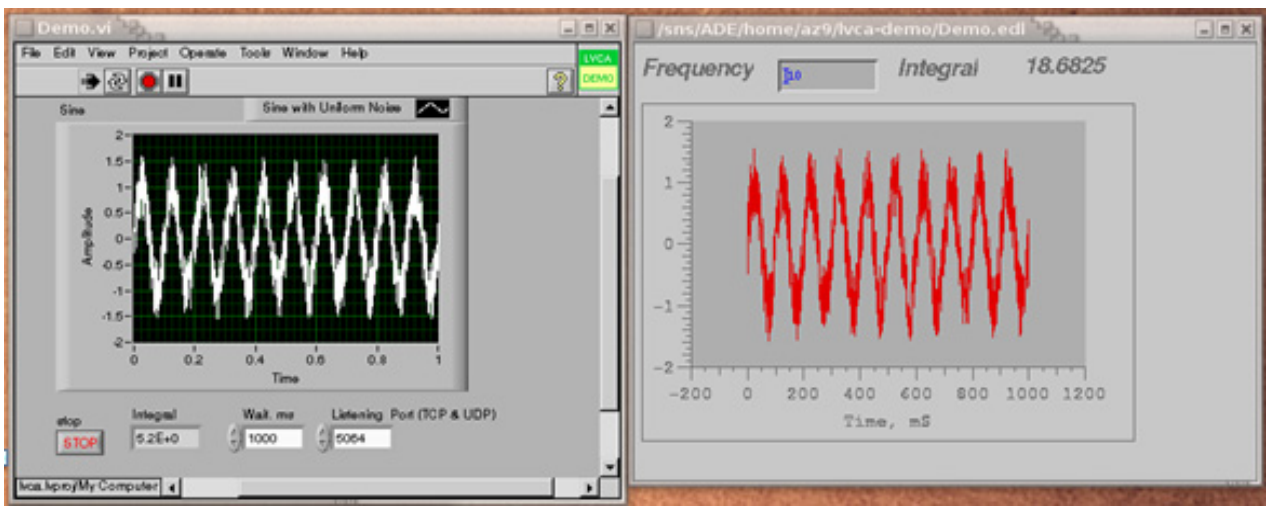

Figure 3: EDM screen (right side) displaying data from LabVIEW server (left side).

The standard tools like (EDM) will be able to get this data and display it – Fig. 3 illustrates EDM screen getting data from CA server.

RESULTS AND FUTURE DEVELOPMENT

We were able to implement this CA server using LabVIEW internal VIs for CA protocol communication. The maximum data throughput was about 160Mbit/s which exceeds our needs. The server was tested on Windows, Linux, Mac OS X, CompactRIO. We did not test LabVIEW Real Time on the PC, but we expect no issues there. We made an attempt to put it into embedded system running an Analog Devices DSP processor, but encountered a LabVIEW bug that was fixed in LabVIEW 2009.

The current version is not a full implementation of CA protocol, but already can be useful.

Our future plans include full implementation of the CA protocol and development of a native LabVIEW CA client.

ACKNOWLEDGEMENTS

ORNL/SNS is managed by UT-Battelle, LLC, for the U.S. Department of Energy under contract DE-AC05-00OR22725.

REFERENCES

[1] Experimental Physics and Industrial Control System http://www.aps.anl.gov/epics/.

[2] D. Thompson and W. Blokland, “A Shared Memory Interface between LabVIEW and EPICS”, ICALEPCS 2003, pp275-277. Gyeongju, Korea, Oct 13-17 2003.

[3] Channel Access for Java http://cosylab.com/solutions/particle_accelerators/Channel_Access_for_Java.

[4] National Instruments Big Physics http://www.ni.com/physics/.