Content from this work may be used under the terms of the CC BY 3.0 licence (© 2018). Any distribution of this work must maintain attribution to the author(s), title of the work, publisher, and DOI.

DISTRIBUTED CONTROL ARCHITECTURE FOR AN INTEGRATED ACCELERATOR AND EXPERIMENTAL SYSTEM[*]

D. J. Gibson[†], R. A. Marsh, B. Rusnak
Lawrence Livermore National Laboratory, Livermore, U.S.A.

Abstract

A neutron imaging demonstration system is under construction at LLNL, integrating 4 MeV and 7 MeV deuteron accelerators with gas-based neutron production target and the associated supply and return systems. This requires integrating a wide variety of control points from different rooms and floors of the Livermore accelerator facility at a single operator station. The control system adopted by the commercial vendor of the accelerators relies on the National Instruments cRIO platform, so that hardware system has been extended across all the beamline and experimental components. Here we present the unified, class-based framework that has been developed and implemented to connect the operator station through the deployed Real Time processors and FPGA interfaces to the hardware on the floor. Connection between the deployed processors and the operator workstations is via a standard TCP/IP network and relies on a publish/subscribe model for data distribution. This measurement and control framework has been designed to be extensible as additional control points are added, and to enable comprehensive, controllable logging of shot-correlated data at up to 300 Hz.

INTRODUCTION

To enable efficient imaging of thick, dense objects, fast neutrons are an ideal candidate because of their long scattering length compared to photons and charged particles. A lab-scale, fast-neutron imaging demonstrator is under construction [1], relying on a commercial 7 MeV D^+ accelerator and an experimental high-pressure, high-volume flow D_2 windowless gas cell, generating up to 10 MeV neutrons via the D(d,n)^{3}He reaction. While the accelerator is provided with dedicated controls, the beamline to deliver the D^+ beam to the target, the gas delivery and recovery systems, and the rotary valve are all custom and will require an integrated control architecture to operate.

In order to make ongoing expansion of this experimental system as simple as possible, efforts have been made to minimize the variety of control hardware and architectures used, as well as to standardize the data pathways to and from control points and monitoring points. Thus the National Instruments Compact Reconfigurable Input/Output (cRIO) hardware system has been selected as the baseline for all the controls, and the LabView programming environment for software development. This was driven by considerations such as the architecture of the commercial accelerator systems (which use cRIO hardware), the relative ease of use of the language for the team and new workers, and the robust support network available from the manufacturer, which will be important when the technology is deployed to a production environment. In this paper we present the basic communication model of the control system which underlies the development plan for the couple-dozen different control point types, and few-hundred individual control points.

SYSTEM ARCHITECTURE

The general model of the control architecture is shown in Fig. 1. The current design includes 4 real-time/FPGA combo chassis plus two additonal FPGA-expansion chassis. At the lowest communication level, custom software runs on each FPGA unit to read the physical inputs (pressure, temperature, and flow sensors, beam position and current monitors, etc.) and control the physical outputs (electric and pneumatic valves, magnet currents, trigger signals, etc.) based on the requests from higher level software. Critical machine protection interlocking is provided at this level (e.g. “turn off the quadrupole magnet if the water flow stops”). Command requests and most recent measurements are exchanged by reading and writing I/O Nodes in the FPGA software. For shot-correlated, high speed data where lossless transmission is required, First-In/First Out (FIFO) buffers are used, allowing the higher-level system to collect the data when it’s ready.

The next level up in the software chain is the embedded controller. This is a microprocessor running either a VxWorks or Linux-based real-time operating system, which allows for deterministic timing of control operations if needed. This layer brings data from the device into the LabView software environment to enable communication with the outside world. The controller could also function as an EPICS input-output controller [2] with appropriate configuration, should that become necessary. Additional machine protection (e.g. consolidating inputs from other subsystems), signal preprocessing (e.g. converting raw bit counts to voltages), or feedback loops are implemented at this level. A single real-time controller talks to one or more FPGA interfaces, and might also communicate with more sophisticated controls via other protocols (RS-232, TCP/IP, etc., used for gaussmeters, turbomolecular pumps, the accelerator controls, and other hardware with self-contained control functions).

Control of the real-time operating software from the supervisory system relies largely on the LabView shared variable architecture, which is used to display current measured val-

[*] This work performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344.
[†] gibson23@llnl.gov

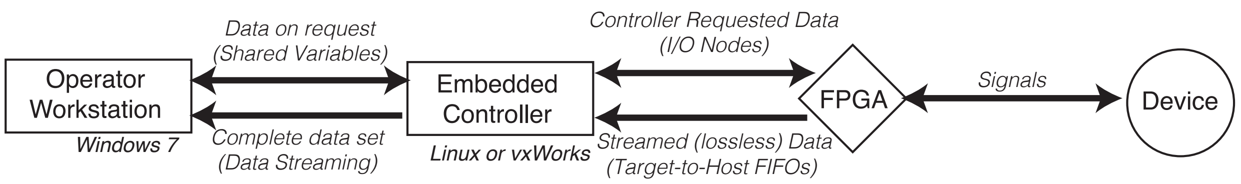
Operator Workstation
Windows 7
Data on request (Shared Variables)
Complete data set (Data Streaming)
Embedded Controller
Linux or vxWorks
Controller Requested Data (I/O Nodes)
Streamed (lossless) Data (Target-to-Host FIFOs)
FPGA
Signals
Device

Figure 1: Summary of command flow mechanisms of the neutron imaging control system architecture.

ues and to send command requests to the field devices. The architecture also provides the Data Streaming System, where the supervisor can register to receive broadcast messaged from the real-time host, allowing a stream of data to be plotted, logged, or otherwise processed without missing any valves.

At the highest level of the software lay the operator workstations and supervisory system. This interface provides the visual feedback to the operator of the current system status, allows for command entry, and is also responsible for data archiving. In general, there is a single operator workstation; however, there is also a redundant "local" control station for the gas compressor system collocated with that hardware. There is also the possibility that for operational reasons in the future a secondary workstation near the accelerator might be desirable. Thus all the controls are designed to be replicated across workstations so devices can be actuated from any location.

DATA STREAMING SYSTEM

In addition to reporting current control values to the operator, the system is also capable of logging data values either periodically or on every trigger. The real-time software runs a data streaming engine over the TCP/IP network. Whenever new data is acquired, the system, in addition to publishing the value to the shared variable engine, will broadcast the value to any registered listeners. This allows lossless data flow to a remote machine that can then archive the data. Built in buffering capabilities enable data collection at up to 300 Hz, delivered to the supervisory system when it is able to receive it. An abstract model of this streaming service is shown in Fig. 2.

Abstract Classes

LabView offers an object-oriented architecture known as the "Actor Framework" to enable easy segmentation of operations into semi-independent tasks. This provides the base code necessary to reliably start up multiple tasks that can run unsupervised, as well as a queue-based message handling scheme to allow inter-task communications. This architecture is the basis for the data distribution system.

The data distribution system relies on 4 abstract classes of actors, which are subclassed to handle specific cases. This allows, for example, the developer to select among different message transmission methods (simple TCP messaging, the zero-MQ protocol [3], or even a service that behaves like a full EPICS implementation) without having to change the hardware interface code. These abstract classes are:

- BroadcastingActor: A class that is used to spin up processes that actually generate data that needs to be transmitted. These subclasses are assigned to a transmitter and automatically know what to do with their data. This can either do all the interfacing with a control element directly, or it can use a ControlHandler subclass to easily interface with multiple control points.

- Transmitter: A class that collects data from the BroadcastingActors and sends it to Receivers that have subscribed to that data. This is subclassed to define the method of data transmission. There is generally one broadcaster per real-time system.

- Receiver: A mate class to the Transmitter that sends subscription requests to the associated transmitter and collects the data to pass on to a local Processor. There is one receiver per processor and per broadcaster. Like the Transmitter, this is subclassed to deal with the specific mechanism of data transfer between the systems.

- Processor: The class that actually does something with the data (e.g. logging it to a file or plotting in a chart). The Processor is created and assigned a Receiver to collect the data from the Transmitter.

The system also relies on two additional abstract classes for support:

- Datagram: A class that actually carries the data and metadata (e.g. names, timestamps, shot counts, etc.) from the BroadcastingActor to the Processor. These are the elements these streming classing can handle, and subclasses are created to deal with specific data.

- ControlHandler: A class that a broadcasting actor can use to handle simple interfaces with FPGA hardware. It knows how to interface between the shared variables and the FPGA commands, and how to generate datagrams. An example would be the PolingActor subclass of the BroadcastingActor, which takes an array of ControlHandlers and periodically asks each of them to update their values.

Content from this work may be used under the terms of the CC BY 3.0 licence (© 2018). Any distribution of this work must maintain attribution to the author(s), title of the work, publisher, and DOI.

ISBN: 978-3-95450-184-7 doi:10.18429/JACoW-IPAC2018-WEPAL043

Content from this work may be used under the terms of the CC BY 3.0 licence (© 2018). Any distribution of this work must maintain attribution to the author(s), title of the work, publisher, and DOI.

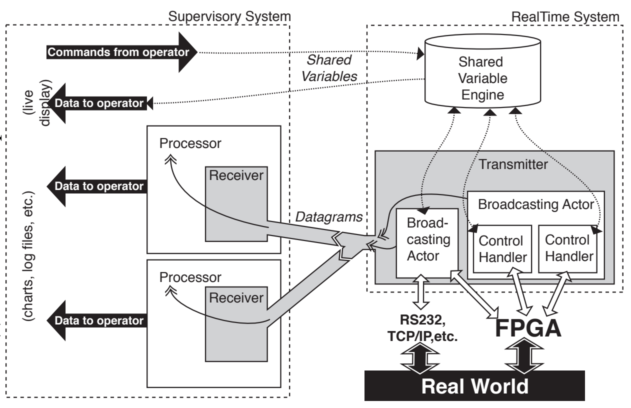

Figure 2: Heirarchy and data flow of the custom data streaming service developed for system controls.

Transmitting and Receiving

Generally, Transmitter and Receiver are subclassed in pairs for each communication protocol to be used. However, single subclasses will sometimes be used for communication to a system that is transmitting data by a defined protocol outside the control system architecture (e.g. receiving data streams from the accelerator, which are defined by the manufacturer). A receiver is configured with a list of datagram channels to register for and given to a processor to be launched. It is expected that the processor will only receive the datagrams corresponding to registered control channels, but how the broadcaster and receiver implement that will vary with the protocols used; the filtering of data can occur at either the broadcaster or the receiver, depending on whether network bandwidth or broadcaster CPU resources are more scarce.

Most data exchange in system is done using National Instruments STM (Simple Messaging) capability. This suite of VIs encapsulates TCP connections with a small amount of metadata and allows for a variety of message types to be exchanged between devices with minimal overhead. It also provides basic connection management functionality which makes designing a transmitter straightforward. The STMReceiver connects to a STMTransmitter and requests the desired control channels. An EPICS-like "search" service, to find a control channel without needing to know which device owns it, is planned for a future upgrade.

Special Data Handling

Most datagrams are just that: packets of data to be stored, plotted, or processed. However, there are two special types of data that require different handling: error messages and metadata.

Error Messages Many processes can encounter erroneous conditions: devices connected via serial or Ethernet ports may be powered off or otherwise non-responsive, poor code could put devices in unexpected states, or the system may have difficultly loading and running the FPGA code. During development, reporting these errors is important for debugging. During operation, the operator should be informed of any unexpected responses. Thus any code with error-handling functionality transmits that error data via an ErrorDatagram. A dedicated operator panel collects all these error messages and displays them, one for each control channel. New errors are highlighted in red to catch the operators attention. As long as the same error code keeps getting generated on the same device (e.g. a gaussmeter keeps timing out because it's off), the error remains in the display. Once that error stops being generated, the error indicator changes to black or drops off the list at the operators discretion. All errors are also logged to a file for later review.

MetaDatagrams In addition to the experimental data they deliver, some control points have a set of metadata that might be useful in evaluating or understanding the data at a later time. This includes device serial numbers, calibration factors, filter settings, scale ranges, etc. This metadata might be gathered by querying the device in question or might be manually recorded as sensors are installed in the system.

Like regular data, metadata is distributed in the form of datagrams, using the GenericMetaDatagram subclass, which provides key/value pairs of metadata in a 2D string array. This datagram is sent to the transmitter via a special "SetMetadata" message, which tells the transmitter to keep a local copy of the datagram in addition to forwarding it to subscribers. When a new receiver subscribed to that data channel, the Transmitter will send the stored meta-datagram as the first message so the receiver has a copy.

CONCLUSION

The framework developed here allows for integrated control and readout of a wide variety of devices and sensors, gather data of various types at a variety of rates, including shot-correlated performance parameters. Custom subclasses of BroadcastingActors and ControlHandlers are being developed to interface with each control loop in the system. With this readily extensible framework in place, adding a new piece of hardware requires only writing the interface code and wiring up the device, and data collection then happens automatically.

REFERENCES

[1] B. Rusnak *et al.*, "Advancement of an accelerator-driven high-brightness source for fast neutron imaging" in *Proc. of 8th Int. Part. Accel. Conf. (IPAC '17)*, Copenhagen, Denmark, May. 2017, paper WEOBB3, pp. 2533–2536.

[2] A. Veeramani, T. F. Debell, W. Blokland, R. Dickson, and A. P. Zhukov, "Options for Interfacing EPICS to COTS Hardware Through LabVIEW"' in *Proc. 12th Int. Conf. on Accel. and Large Exp. Phys. Control Syst. (ICALEPCS '09)*, Kobe, Japan, October 2009, paper THD004, pp. 913–915.

[3] Distributed Messaging - zeroMQ, http://zeromq.org/