TAILORING THE HARDWARE TO YOUR CONTROL SYSTEM*

E. Björklund, S.A. Baily, Los Alamos National Laboratory, Los Alamos, NM 87545, U.S.A.

Abstract

In the very early days of computerized accelerator control systems the entire control system, from the operator interface to the front-end data acquisition hardware, was custom designed and built for that one machine. This was expensive, but the resulting product was a control system seamlessly integrated (mostly) with the machine it was to control. Later, the advent of standardized bus systems such as CAMAC, VME, and CANBUS, made it practical and attractive to purchase commercially available data acquisition and control hardware. This greatly simplified the design but required that the control system be tailored to accommodate the features and eccentricities of the available hardware. Today we have standardized control systems (Tango, EPICS, DOOCS) using commercial hardware on standardized busses. With the advent of FPGA technology and programmable automation controllers (PACs & PLCs) it now becomes possible to tailor commercial hardware to the needs of a standardized control system and the target machine.

In this paper, we will discuss our experiences with tailoring a commercial industrial I/O system to meet the needs of the EPICS control system and the LANSCE accelerator. We took the National Instruments Compact RIO platform, embedded an EPICS IOC in its processor, and used its FPGA backplane to create a "standardized" industrial I/O system (analog in/out, binary in/out, counters, and stepper motors) that meets the specific needs of the LANSCE accelerator.

BACKGROUND

The 800 MeV proton accelerator at the Los Alamos Neutron Science Center (LANSCE) was designed and built in the 1960's. The original design included a custom-built computer control system based on a custom-built data acquisition system that we called RICE ("Remote Information and Control Equipment") [1]. In the 1980's, with the advent of the CAMAC standard, we adapted our control system to use both CAMAC and RICE. Then, in the 1990's, we introduced both VME and EPICS into our control system. This required a lot of adaptation – old control systems to new hardware, new control systems to old hardware, and new and old control systems to each other.

While it is nice to keep up with new technology, unfortunately it did not mean that we got to eliminate any of the old technology. And so, in the 2000's we found ourselves supporting a control system that included three generations of hardware technology (RICE, CAMAC, and VME) and two generations of software technology (EPICS and the legacy control system).

Now, at last, we have the opportunity to start phasing out our old hardware and software. The only way to make this economically feasible, however, was to use hardware that could a) interface with the accelerator the way the accelerator equipment was designed, and b) provide a straightforward interface to the new software (EPICS). And so we began exploring the use of programmable hardware solutions.

THE NEW LANSCE INDUSTRIAL I/O SYSTEM

For our first test case we decided to replace the Industrial I/O (IIO) portion of one RICE module with a commercial programmable logic controller (PLC). We defined "Industrial I/O" to encompass the basic, non-time-critical, non-closed-loop, and non-safety-critical functions of the control system. The PLC system worked well, but we discovered that it was not fast enough for some of our IIO binary output channels. It also could not time ADC reads to avoid the noise induced on the system by the accelerating RF.

For our next iteration, we traded the PLC for a National Instruments Compact RIO system, which is about as environmentally rugged as a PLC, but can also be several orders of magnitude faster. In the Compact RIO system, I/O cards plug directly into an FPGA. The FPGA can be programmed using LabVIEW, which gets translated into VHDL and then into the FPGA bitmap.

The standard RICE-replacement system we constructed is an 8-slot Compact RIO system that features 64 binary input channels, 32 sinking binary output channels, 8 solid state relay binary output channels, 32 analog input channels, 16 analog output (DAC) channels, and 4 stepper motor channels. The analog inputs can be triggered in order to avoid RF noise. Variants of the standard system are possible, and may replace some functionality (e.g. stepper motors) with other functionality (e.g. counters). In most cases, one (IIO) chassis can service all the industrial I/O channels in one RICE module. Details of the specific implementation can be found in the companion paper [2].

ADAPTING THE COMPACT RIO TO RICE

The specific features of the RICE system that we wanted to emulate in our new IIO controller were:

- Timing the ADC reads to avoid RF-induced noise
- Multiple protocols for binary output commands.
- Knob-friendly stepper motors.
- Easy on-line reconfiguration

*Work supported by the US Department of Energy under contract DE-AC52-06NA25396

Binary Output Protocols

The LANSCE control system uses four different command protocols for binary output channels. These are:

- **<u>Command Only</u>:** The simplest protocol. Turn it on and it goes on. Turn it off and it goes off.
- **<u>Command With Latchback</u>:** The most complex protocol and the most common protocol used by RICE. Each command channel has an associated readback channel. The command value tracks to the value of the readback channel until a command is issued. The command value is held for a specified "hold time", allowing time for the readback to reflect the new command value before the command channel starts tracking it again. Latchback channels are useful when a device can be commanded from multiple sources.
- **<u>Momentary Normally Open</u>:** The command channel is normally low. Writing a 0 to the channel has no effect. Writing a 1 to the channel causes the device to change state. When a 1 is written, the command channel will be held high for the specified hold time, after which it reverts to low. A "momentary normally open" channel with a latchback is useful for implementing fault/reset logic.
- **<u>Momentary Normally Closed</u>:** The same protocol as "momentary normally open" except that the command output is inverted.

CAMAC and VME systems require a different card type for each of four binary output protocols. Even more card types are required if the hold times are implemented in hardware. By programming the protocol into the controller, we were able to implement all the binary command channels with only two card types.

Control Knobs and Stepper Motors

One of the unique features of our original control system is its heavy reliance on stepper motors as the primary analog output device and assignable control knobs as the primary analog output interface. This was a problem when we started to integrate EPICS into the control system because EPICS is not very good at implementing control knobs and particularly bad at controlling stepper motors with control knobs. Consequently, we spent a lot of time adapting our EPICS system to work well with control knobs [3].

A problem that can occur when knobbing a stepper motor is that the pulses can accumulate faster than the motor can turn, resulting in overshooting the intended target. We avoid this problem by limiting the output of the control knobs to only the number of pulses that can be accumulated in a fifth of a second. We also programmed the stepper motor controller to preempt the current pulse stream whenever a new pulse stream is received. In this way we guarantee that the device will stop moving when the knob stops turning.

Instead of using a stepper motor card, we chose to implement our stepper motor controller with a simple binary output card. This allowed us to program in the exact pulse width, speed and ramping characteristics that the accelerator equipment expected from RICE.

Stem Cells and Reconfiguration

In the RICE system, when you needed to change the protocol of a binary output channel, or give it a different readback, you simply moved a jumper or re-routed a wire. Typically this could be accomplished in a matter of minutes and did not disturb any of the other equipment controlled by that RICE module. To accomplish this same task with an FPGA, first the FGPA source code must be changed. Then (in the case of Compact RIO) the source code must be translated into VHDL. Then the bitmap needs to be reconstructed from the VHDL (a potentially lengthy process). Finally, the Compact RIO system must be taken off-line while the bitmap is re-flashed and the Compact RIO rebooted.

Reconfiguration occurs more frequently than one who is not accustomed to the workings of an accelerator laboratory might think. In order to minimize the amount of time it takes to reconfigure our FPGA systems, we adopted a "Stem Cell" approach. Under this approach, the FPGA code for a binary output channel (for example) resembles a biological stem cell. A binary output "stem cell" has the possibility of becoming any command type (command only, latch-back, momentary open, or momentary closed), using any binary input for its readback, and having any hold time between 0 and 65 seconds (in millisecond intervals). The "stem cell" does not take on a specific function until it receives instructions from a configuration process. The configuration process reads a configuration file and assigns the specified functions to the stem cells. Configuration runs at startup, but can be invoked again at anytime. Thus it is possible to completely reconfigure the action and behavior of a binary output channel "in vivo" without any interruption of service.

The chief drawback of the "stem cell" approach is that it requires more FPGA real estate per channel. When we first tried this approach on a Virtex 2 FPGA, we only had enough space to implement 11 binary output channels. After upgrading to a Virtex 5, however, we had more than enough room for 40 binary output channels and 64 binary input channels.

Binary outputs are not the only stem cells in our system. We can also dynamically configure the dynamic range and trigger of our analog input channels, the pulse rate and (to some extent) the ramp up rate of our stepper motor channels, and the integration time of our counter channels.

EMBEDDING THE CONTROL SYSTEM IN THE HARDWARE

Perhaps one of the most dramatic ways to tailor the hardware to your control system is to actually embed it within the hardware. Many commercial products such as PLCs, PACs, Serial Controllers, and beam diagnostics employ an embedded processor running a real-time or "near" real-time operating system. This raises the possibility of actually embedding the control system (or at least the "front end" part of the control system) in the device's processor and letting it interact directly with the vendor's code. One of the strengths of belonging to a control system collaboration is that vendors have been willing to entertain and even market this ability. We have already seen this with products from Instrumentation Technologies [4], Moxa [5], National Instruments [6], Yokogawa [7], and ZTEC Instruments [8].

The Compact RIO uses a power-PC running the vxWorks real-time operating system. National Instruments and CosyLab collaborated with us to install a complete EPICS I/O Controller (IOC) on the Compact RIO. The EPICS software runs concurrently with the National Instruments software and communicates with it using a shared memory interface concept originally pioneered at the Spallation Neutron Source [9].

One immediate advantage of embedding the control system is that data displayed on the operator screens now comes directly from the Compact RIO. With the PLC, the data had to go from the PLC to an EPICS IOC and then to the operator screen. Another advantage is that the Compact RIO is now able to take advantage of all the EPICS utilities such as archiving, bumpless reboot, access security, alarm handling, performance diagnostics, and our local software for making control knobs work well with EPICS. A final advantage is that the Compact RIO system may also access other local devices that have Ethernet interfaces – such as, for example, a PLC.

One final topic worth mentioning is that FPGAs are now coming equipped with hard and soft-core processors, making it possible to embed the control system right on the FPGA. Some examples can be found in [10], [11], and [12].

CONCLUSION

Not only has the Compact RIO IIO controller worked well as a replacement for RICE, but with some minor tweaks we found it also works well as a replacement for CAMAC. To date we have replaced two RICE modules and two CAMAC crates with our standard Compact RIO IIO systems. The three things that have contributed to this success have been 1) the ability to program the controllers, which allowed us to interface with existing accelerator equipment, 2) embedding the control system in the controller, which simplified the software interface, and 3) the ability to reconfigure the interface on-line.

ACKNOWLEDGEMENTS

The authors would like to thank David Bonal, Thierry Debelle and Ryan King of National Instruments for their invaluable support and assistance with the initial implementation of the Compact RIO IIO Controller. We would also like to thank Rok Šabjan of Cosylab for his initial work on getting EPICS to run on the Compact RIO as well as for the initial implementation of the shared memory library.

REFERENCES

[1] D.R. Machen, R. Gore and D. Weber, "A Compact Data Acquisition And Control Terminal For Particle Accelerators", PAC'69, IEEE Trans. Nucl. Sci. NS-16, p.883 (1969) ; http://www.JACoW.org

[2] S.A. Baily and E.Björklund, "Tailoring the Hardware to Your Control System", 5th NI Big Physics Symposium, Austin, August 2011, publication pending; https://decibel.ni.com/content/groups/big-physics?view=documents

[3] E. Björklund, "Toward A General Theory of Control Knobs," ICALEPCS'01, San Jose, November 2001, THAP071, p.608 (2001); http://www.JACoW.org

[4] C.Scafuri, V. Forchì, G. Gaio and N Leclercq, "Embedding a TANGO Device Into a Digital BPM", PCaPAC'06, Newport News, October 2006, p. 23; http://www.jlab.org/conferences/PCaPAC/PCaPAC2006_proceedings.pdf

[5] G.Y. Jiang and L.R. Shen, "An Embedded EPICS Controller Based on Ethernet/Serial Box," ICALEPCS'07, Knoxville, October 2007, WPPA06, p.328 (2007); http://www.JACoW.org

[6] E. Björklund, A. Veeramani and T. Debelle, "Using EPICS Enabled Industrial Hardware for Upgrading Control Systems," ICALEPCS'09, Kobe, October 2009, WEP078, p.555(2009); http://www.JACoW.org

[7] A. Uchiyama et. al., "Development of Embedded EPICS on F3RP61-2L," PCaPAC'08, Ljubljana, October 2008, WEX03, p.145 (2008); http://www.JACoW.org

[8] B.L. Shaw and C.D. Ziomek, "Off-The-Shelf EPICS Instrumentation for Remote Waveform Monitoring & Analysis," IPAC'10, Kyoto, May 2010, MOPE082, p.1173 (2010); http://www.JACoW.org

[9] D. Thompson and W. Blokland, "A Shared Memory Interface Between LabVIEW and EPICS," ICALEPCS'03, Gyeongju, October 2003, TU514, p.275 (2003); http://www.JACoW.org

[10] J. Weber, M. Chin, C. Timossi, and E. Williams, "Hardware and Software Development and Integration in an FPGA Embedded Processor Based Control System Module for the ALS," PAC'07, Albuquerque, June 2007, MOPAS031, p.503 (2007); http://www.JACoW.org

[11] A. Götz, J. Butanowicz, L. Slezak, C. Scafuri and G. Gaio, "Ubiquitous TANGO," ICALEPCS'07, Knoxville, October 2007, WPPA28, p.374 (2007); http://www.JACoW.org

[12] J. Odagiri et. al., "Fully Embedded EPICS-Based Control of Low Level RF System for SuperKEKB," IPAC'10, Kyoto, May 2010, WEPEB003, p.2686 (2010); http://www.JACoW.org