THE FIRST DEVELOPMENT OF AN EPICS ACCELERATOR CONTROL SYSTEM FOR THE IDAHO ACCELERATOR CENTER

Anthony Andrews*, Y. Kim, C. Eckman, A. Hunt, and D. Wells
Department of Physics, Idaho State University, Pocatello, ID 83209, USA
K. Kim, SLAC National Accelerator Laboratory, Menlo Park, CA 94025, USA

Abstract

The Idaho Accelerator Center (IAC) of Idaho State University, has been operating nine low energy accelerators for nuclear physics applications and medical isotope production [1]. But almost all of those accelerators do not have modern computer based systems to control the various accelerator components remotely. To obtain stable accelerator operations with a good reproducibility, the EPICS accelerator control system has been adapted. After developing an EPICS control system for various components, the same control system will be applied to all other operating accelerators at the IAC. Since January 2011, an EPICS control system has been developed for a 16 MeV S-band linac by collaborating with SLAC controls department. This paper describes the first EPICS accelerator control system to control magnet power supplies of the S-band linac at the IAC.

INTRODUCTION

To upgrade the computer based accelerator control system at the IAC, various control systems using EPICS have been developing. As shown in Fig. 1, the ultimate goal of our project is a combination between the control system for a magnet power supply and a control system for a GigE CCD camera using MATLAB Channel Access (MCA). By combining these two control systems, a new tool will be available which can measure the beamsize on an OTR screen and current of a magnet power supply simultaneously. This will enable us to make automatic emittance measurements with EPICS, MCA, and MATLAB [2]. The control system described here is specifically for a TDK-Lambda ZUP magnet power supply [3]. To develop its EPICS control system, the power supply was connected to a MOXA terminal server with an RS485 interface [4]. Then the terminal server was connected to the network using an Ethernet cable for remote control. After that, a means to communicate with the power supply was developed by using two EPICS modules (ASYN and StreamDevice) [5]. Finally, a device support application was programmed and the current of the TDK-Lambda ZUP power supply was controlled by manipulating process variables (PVs) using a simple operator interface (OPI) panel. This paper describes the details of the first EPICS accelerator control system developed for the IAC.

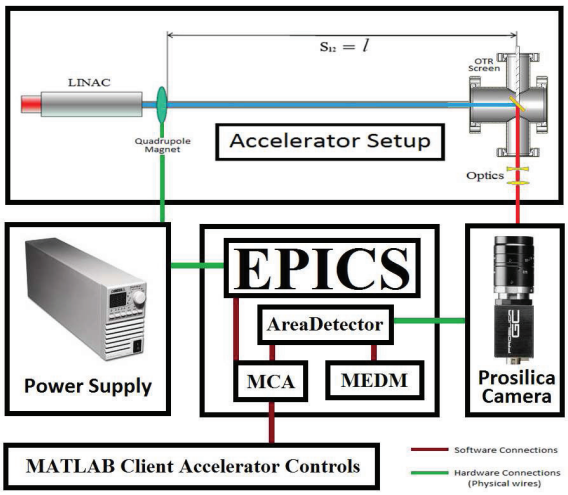

Figure 1: Schematic layout of an EPICS, MCA, and MATLAB based automatic emittance measurement system [2].

DEVICE SUPPORT APPLICATION

In order to control the magnet power supply, the first step is to download and install the correct software to communicate to the power supply, and then make a device support application which contains the files required to create PVs. To facilitate communication with devices like the TDK-Lambda power supply, the synApps software package including ASYN and StreamDevice was installed [5].

ASYN

ASYN is one of EPICS modules, which facilitates communication between the device, higher level programs, device support, and others. In other words, ASYN interfaces higher level programs and drivers in a client computer to the very basic code which is supplied with the device [6].

StreamDevice

StreamDevice is another EPICS module and a kind of specific device support for communication by sending and receiving strings [7]. StreamDevice is used to make protocols from commands, which only the device can understand. Records use these protocols to make PVs.

Creating the Application

Before creating PVs, first of all, a device support application must be created for them to live in. To create the application, the following commands were used [5]:

* Mail: andranth@isu.edu

Presented at the 3rd International Particle Accelerator Conference (IPAC 2012)
New Orleans, Louisiana, USA, May 20 - 25, 2012

Work supported in part by US Department of Energy under contract DE-AC02-76SF00515.

- makeSupport.pl -t streamSCPI devicename
- makeBaseApp.pl -t ioc devicename1
- makeBaseApp.pl -t ioc -i devicename1

The first command makes a support directory where the database files and protocol files are located. The second command creates a configure directory and an application directory. The third command creates an iocBoot directory, which contains the *st.cmd* file [5]. The "devicename" and "devicename1" are arbitrary names for the support directory, application directory, and the name of the directory, within iocBoot, where the *st.cmd* file is located. The convention here is to use some kind of name which is similar to the device, however it's a good idea to make the names of each of those directories different so that they won't be confused with each other. Protocol files, database files, and *st.cmd* files will be explained in more detail in later sections.

SCPI Commands

The commands of the TDK-Lambda ZUP power supply follow Standard Commands for Programmable Instruments (SCPI). One example of the commands which were used is:

- :CUR?;

This is a query command which asks the value of the current of the power supply [3]. Now these simple commands need to be converted into PVs, so the operator can easily read and write current values to the device.

Protocol Files

Stream Device uses protocol files to translate simple commands that the equipment understands (SCPI commands) into a format that EPICS records understand [7]. For example, the previous :CUR?; command would look like this:

- getI{ out ":CUR?"; in "AA%6f";}

In this case, getI becomes a protocol, which sends the query command :CUR?; "out" to the device and gets a response "in" from the device which has an AA followed by a pointing number with six floating digits. So, an example of a response could look like AA05.000, which corresponds to 5 A. Another example of a protocol is the following:

- setI{ out ":CUR%06.3f";}

This protocol is just a write operation sent by the operator. It is used to set the current of the power supply to be a six digit number with three decimal places. Something that is common to see at the top of a protocol file in the first two lines are terminators. A terminator is a symbol entered by the operator or an automatic response from the device which signals the end of an I/O operation [7].

OutTerminator The OutTerminator is a symbol that the operator puts on the end of a string to let the device know that the write operation is complete [7]. This is something that differs from device to device, but for our TDK-Lambda ZUP power supply, that character is a semi-colon. However, protocol files produce errors when too many semi-colons are placed next to each other. So the best way to handle that is by representing the terminator with a hexadecimal equivalent like this:

- OutTerminator=0x3B;

This represents the semi-colon character in hexadecimal.

InTerminator The InTerminator is the automatic response sent back from the device to the operator to let the operator know that the read operation is complete [7]. In our case, this was a line feed. The syntax for a line feed looks like this:

- InTerminator=LF;

A line feed occurs when the response from the device is displayed one line below the command, which is sent to the device.

Database File

The database file or files are where the records are located. The way that PVs are made is by putting the protocol name (ie getI) into the input or output fields in the record [5, 7]. Here's one of the records that was used to read the current:

```
record( ai, "$(P)$(R)Curr_R"){
field( DESC, "current read")
field( DTYP, "stream")
field( INP, "@NL2010.proto getI $(PORT) $(A)")
field( PREC, "6")
field( EGU, "Amps")
field( HOPR, "10")
field( LOPR, "0") }
```

One field of particular interest is the INP field. This field sends the command defined as the protocol "getI" located in the protocol file "NL2010.proto" to the port and address of the device. The $(PORT), $(A), $(P), and $(R) are all defined in the *st.cmd* file.

st.cmd File

This is an executable file that establishes a connection to the device by using it's IP address, portName, and address. This file also loads all of the records that are in the database files. The way that one specifies what the IP address of the device depends on the hardware interface of the device and how one communicates with the device. For the TDK-Lambda ZUP power supply, which was interfaced to a MOXA terminal server with an RS485 cable, telnet was used before the EPICS channel access (CA) connection. To specify this setup, the following line

was entered in the *st.cmd* file.

drvAsynIPPortConfigure(”P3”,”134.50.A.B:4003”,0,0,0)

Here P3 is a portName, which is an arbitrary identifier used to define $(PORT) from records as an IP address (134.50.A.B) and a port (4003) of the MOXA terminal server [5]. In our case, P3 was used because our power supply was connected to the third port of the MOXA terminal server with an RS485 cable. The rest (0,0,0) specifies *priority, noAutoConnect,* and *noProcessEos* [5, 6]. To specify a database in the *st.cmd* file, a line similar to the following thing was used.

dbLoadRecords(”db/.db”,”P=$(P),R=$(R),PORT=,A=”)

Here the .db is the database file that contains all of the needed records. The macros P and R are defined in the *st.cmd* file as well. They specify what the full name of a record will be. For example P could be something like “NL2010” and R could be “Test”. The PORT is the same as the portName. The address (A=) for our set-up was zero because the test bench which was used for this project had only one power supply [5].

MEDM OPI

Motif Editor and Display Manager (MEDM) is an EPICS extension which provides a way to create simple OPI’s [8]. One of these OPI’s to control the current of the power supply is shown in Fig. 2. Here the current of a quadrupole magnet power supply was automatically scanned by a MATLAB program to increase the current from 0 A to 2 A with a speed of 0.2 A in 0.5 seconds, while ten images were automatically taken by a Prosilica GigE GC1290 CCD camera in each scan step. This is an example of the EPICS, MCA, and MATLAB based automatic emittance measurement system [2]. If the operator wants to control the current manually, they could simply change the current by moving the slide bar on the bottom left of Fig. 2. The left meter above the current controller shows the exact set-value of the power supply current, and the right meter above the current controller shows the read-back of the power supply current. The top-left strip chart shows the change in set and read-back values of the power supply current, while the top-right strip chart shows the change in the read-back value of the power supply voltage with time. Since the power supply was operating in the constant current mode, only the read-back value of the voltage was displayed on the strip chart on the right hand side. However, a controller for the voltage was included in case the operator wants to operate the power supply in the constant voltage mode instead of the constant current mode.

CONCLUSION

Our goal for this project was to make an EPICS control system for a TDK-Lambda ZUP power supply. This was

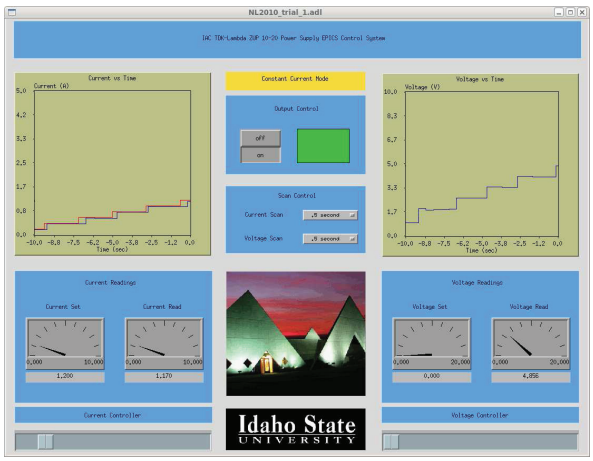

Figure 2: An MEDM OPI pannel to control and read the current of the TDK-Lambda power supply.

accomplished by using ASYN and StreamDevice to communicate with the power supply and convert SCPI commands into PVs. Then we created an MEDM OPI to control and monitor those PVs. We successfully developed the first EPICS control system at the IAC to control a TDK-Lamda ZUP power supply for a low energy accelerator. In the near future, the control system of the IAC accelerators can be upgraded with our newly developed EPICS accelerator control system. In addition, a simple demonstration of the EPICS, MCA, and MATLAB based automatic emittance measurement system was successfully done. I would like to thank Dr. Eric Norum who provided invaluable advice, help with troubleshooting various errors, and patiently explained many EPICS concepts through email correspondence.

REFERENCES

[1] http://www.iac.isu.edu

[2] C. Eckman *et al*., in *Proc. IPAC2012*, New Orleans LA, USA.

[3] http://www.tdk-lambda.com/products/sps/ps_adj/zup/indexe.html#

[4] http://www.moxa.com/product/nport_6650.htm

[5] E.Norum, “HowToDoSerial(StreamDevice). Argonne National Laboratory, http://www.aps.anl.gov/epics/modules/soft/asyn/HowToDoSerial_StreamDevice.html

[6] E.Norum, *et al*., “asynDriver: Asynchronous Driver Support”. Argonne National Laboratory, http://www.aps.anl.gov/epics/modules/soft/asyn/R4-18/asynDriver.html

[7] D. Zimoch, “EPICS StreamDevice .” StreamDevice. Paul Shriner Institut. http://epics.web.psi.ch/software/streamdevice/doc/index.html

[8] K.Evans, Jr.,“MEDM Reference Manual”. Argonne National Laboratory, http://www.aps.anl.gov/epics/EpicsDocumentation/ExtensionsManuals/MEDM/MEDM.html