DAQ application using open source tools for Plasma heating experiment

by Rameshkumar Joshi

Institute for Plasma Research, India
Integrated DAQ architecture

Programmable Logic Controller

MODBUS-TCP communication

Computer having CSS OPI, EPICS, Python, C

Communication over Http

Client having Web-browser

Pymodbus

PyEpics, Cython, Numpy, Scipy, Matplotlib

MODBUS TCP

Field I/o

Signal Conditioning

PLC

EPICS Client

CSS Studio
There are multiple DAQ systems running simultaneously in the Institute. They must exchange data during experiment. Experimental Physics and Industrial Control Systems (EPICS) is provide the infrastructure by which each DAQ can exchange data as per requirement.

User Interface has been designed using Java based Control System Studio (CSS) which can glue with the EPICS network variable to plot as well as display values.

The system is designed and implemented the real time programmable logic controllers (PLC) data to monitoring and control using EPICS.

The Interface between PLC and computer is based on Ethernet. Using MODBUS/TCP communication data have been exchanged between PLC and Computer. For MODBUS communication Pymodbus library has been used as implementation platform.

Logical control has been designed using Python. The calibration and algorithm implementation is designed using C. The C functionality can be called using cython interface with Python.

Calibration of the instrumentation data has been implemented by curve fitting using Numpy, Scipy and Matplotlib.
The infrastructure layer: EPICS

• The infrastructure layer is implemented with **EPICS** (Experimental Physics and Industrial Control System)

• EPICS is
  – an open-source control system toolkit
  – used in hundreds of large and small experimental physics projects world-wide: light sources, high energy physics, fusion (KSTAR, NSTX), telescopes
  – maintained and further developed by a world-wide community of users (including ITER)

• The same infrastructure for the CODAC servers and for the plant system controllers to ensure a **uniform standard interface**.
CSS and Python

Python action script on action button click event handling
## Used tools and libraries

<table>
<thead>
<tr>
<th>Name</th>
<th>File</th>
<th>Download</th>
</tr>
</thead>
<tbody>
<tr>
<td>PYTHON</td>
<td>Python-2.7.6.tar.xz</td>
<td><a href="http://www.python.org/getit/">http://www.python.org/getit/</a></td>
</tr>
<tr>
<td>EDM</td>
<td>edm-1-12-71.tgz</td>
<td><a href="http://ics-web.sns.ornl.gov/edm/">http://ics-web.sns.ornl.gov/edm/</a></td>
</tr>
<tr>
<td>CSS</td>
<td>CSS-3.0</td>
<td><a href="http://cs-studio.sourceforge.net/">http://cs-studio.sourceforge.net/</a></td>
</tr>
</tbody>
</table>

### PV Access from a Shell

EPICS Base provides several utilities for reading and modifying Process Variables from command shell. Here `caget` and `caput` are demonstrated by accessing our softIoc.

```
$ caget calc:a calc:b calc:sum
calc:a 0
calc:b 0
calc:sum 0
$ caput calc:a 2
Old : calc:a 0
New : calc:a 2
$ caget calc:a calc:b calc:sum
calc:a 2
calc:b 0
calc:sum 2
$ caput calc:b 3
Old : calc:b 0
New : calc:b 3
$ caget calc:a calc:b calc:sum
calc:a 2
calc:b 3
calc:sum 5
```

### Database Creation:

```python
import ctypes
import time
from epics import caget, caput
tmy_test_lib = ctypes.cdll.LoadLibrary('/home/admin/CSS-Workspaces/Default/icrh/libds.so')
print "now initialization is %d" % tmy_test_lib.DsInit()
print "now connection is %d" % tmy_test_lib.DsConnect()
while True:
tmy_test_lib.DsRead()
    f3 = tmy_test_lib.DsReadAnalogIp(2)*20/4095
    caput("ICRH:KW2_PLATE_VOLTAGE",f3)
    time.sleep(.100)
```

```python
import cftime
import import time
from epics import caget, caput
tmy_test_lib = ctypes.cdll.LoadLibrary('/home/admin/CSS-Workspaces/Default/icrh/libds.so')
print "now initialization is %d" % tmy_test_lib.DsInit()
print "now connection is %d" % tmy_test_lib.DsConnect()
while True:
tmy_test_lib.DsRead()
    f3 = tmy_test_lib.DsReadAnalogIp(2)*20/4095
    caput("ICRH:KW2_PLATE_VOLTAGE",f3)
    time.sleep(.100)
```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <sys/wait.h>
#include <signal.h>
#include <tcl.h>
#include "nrutil.h"

#define MYPORT 4461  
#define NO_OF_SAMPLES 2  
#define ACQ_LIMIT 100000

int DsReadAnalogIp(int chno)
{
    int status;
    int value;
    int sine_bit = 1;
    unsigned short int b;
    b = disp_packet.data[chno] & 0x00ff;
    disp_packet.data[chno] = disp_packet.data[chno] & 0xff00;
    disp_packet.data[chno] = disp_packet.data[chno] >> 8;
    b = b << 8;
    disp_packet.data[chno] = disp_packet.data[chno] + b;

    /* convert into double */
    if ( disp_packet.data[chno] > 0x7ff0 )
    {
        disp_packet.data[chno] = 0xfff0 - disp_packet.data[chno];
        sine_bit = -1;
    }
    return value;
}
import numpy as np
import matplotlib.pyplot as plt
import scipy as sp
from scipy.interpolate import interp1d
from scipy import interpolate

with open('kw20_plate_c.calib', 'r') as f2:
    lines = f2.readlines()
data = [line.split() for line in lines]
data2 = np.asfarray(data)
    x1 = data2[:,0]
y1 = data2[:,1]

print len(x1)
new_length=len(x1)
new_x = np.linspace(x1.min(), x1.max(), new_length)
#new_y = sp.interpolate.interp1d(x, y, kind='cubic')(new_x)
#new_y = interp1d(x1, y1, kind='cubic')(new_x)

print x1.min()
print x1.max()
print new_x
print new_y
print interp1d(x1, y1, kind='cubic')(0.88)
print interp1d(x1, y1, kind='cubic')(2.88)
print interp1d(x1, y1, kind='cubic')(4.88)
new_y = interp1d(x1, y1, kind='cubic')(new_x)
plt.plot(x1, y1,'r')
plt.plot(new_x, new_y,'g')
plt.show()
Experiments and Results

OPI user interface screen for DAC software
Experiments and Results

1.5 MW monitoring and control screen for DAC software
Experimental shot panel for DAC software
EPICS and Extended modules have been compiled with Linux 32 bit machine as host computer.

1000 process variables have been created with unique PV name which have been exported with softIOC command provided by EPICS base.

PyEPICS and python script program has been used to run recursively at specified time delay for data monitoring and acquisition.

Cython package gives ability to make C library as shared object which functionality can be called using Python program.

Wireshark and system monitor utilities have been used to check system parameters and validation with benchmarking.

CSS OPI (user interface) can be run from any client computer which gets parameter and display data on periodic time scale.
References

3. Delta AH-500 PLC module reference:
7. Pyepics Python Package, cars9.uchicago.edu/software/python/pyepics3/
15. X. Chen, K. Kasemir, "BOY, a modern graphical operator interface editor and runtime". Proceedings of 2011 Particle Accelerator Conference, New York, NY, USA.
20. http://m2m.eclipse.org/
Thanks