Acromag for Slow Controls

Overview

The Acromag BusWorks XT system of terminals provides a cheap, easy-to-use set of slow (<16Hz) analog and digital ADC/DAC channels.

This page describes how to set up a full slow controls system. The short and sweet set up is:

1. create an Ubuntu 12.04 OS server machine
2. install EPICS modbus on that machine
3. configure the IP address and other settings of the Acromag terminals
4. configure the modbus IOC to interface to the terminals
5. set up the hardware on the network

Acromag BusWorks XT terminals

The Acromag BusWorks XT units [http://www.acromag.com/catalog/97/Ethernet,_Modbus___Profibus_I_O/Ethernet_I_O_Modules/Analog_I_O], illustrated below, are DIN-rail mounted individual terminals that provide ADC/DAC analog/digital capabilities. Multiple terminals can be daisy chained together on a single DIN-rail with a single +24V source powering the first terminal.

BUSWORKS XT
Modbus TCP/IP
EtherNet/IP conformance tested
PROFI NET
with PriorityChannel Technology
USB Configured

Units:

1. XT1111 [http://www.acromag.com/catalog/953/Ethernet,Modbus_Profibus_I_O/Ethernet_I_OModules/Digital_I_O/XT1110] 16-Channel Discrete I/O with Sinking Outputs, 0-32V DC Input, TTL thresholds
2. XT1221 [http://www.acromag.com/catalog/1022/Ethernet,Modbus_Profibus_I_O/Ethernet_I_OModules/Analog_I_O/XT1220] Modbus/TCP and i2o protocol, 8-channel differential voltage input module
3. XT1541 [http://www.acromag.com/catalog/1044/Ethernet,Modbus_Profibus_I_O/Ethernet_I_OModules/Analog_I_O/XT1540] Modbus/TCP and i2o protocol, 8-channel analog voltage output, 4-channel digital I/O module]]
4. 966EN-4006 [https://www.acromag.com/catalog/251/ethernet-modbus-profibus-io/ethernet-io-modules/analog-io/busworks-900en-series/966en] 6-Channel RTD/Resistance Input Ethernet Module

Manuals

- XT1111 16 channel I/O binary (sinking version)
- XT1112 16 channel I/O binary (sourcing version)
- XT1211 8 Ch Differential voltage input
- XT1541 8 Ch Aanlog output + 4 digital output
- 966en-4006 6 channel RTD input

Initial set up of Acromag terminals (Windows)

Each terminal must be configured using a configuration program that is only available in Windows.

1. Install configuration software on Windows machine (http://www.acromag.com/sites/default/files/9500465D.zip [http://www.acromag.com/sites/default/files/9500465D.zip])
2. Provide +24V of power to the terminal

3. Connect USB cable between the terminal and the computer
4. Launch the Configuration Software (12xx version for ADC, 15xx version for DAC)
5. Select the attached terminal and open a connection to it
6. Provide a fixed IP address to the terminal
7. Upload that to the terminal
8. For ADCs, go to the Input Configuration Tab.
9. For every channel set the input averaging to the desired level (max = 200).
10. Click “Send Input Config” to upload to the unit
11. Close the connection to the unit from the Device Tab
12. Say goodbye to Windows forever.

Configuring non-USB units over IP/Ethernet

The 966EN temperature sensor units are even easier to set up. On first boot they are configured to take the default IP 128.1.1.100. Simply reserve (or at least make sure its free on the local network) and then go the the units self hosted webpage http://128.1.1.100 [http://128.1.1.100] in your browser. If you have a mac this can be even easier, you can just connected directly to the ethernet of your computer and the mac should automagically form an adhoc network (turn off wifi to be sure you are accessing the Acromag card only).

Once on the configuration webpage:

1. Provide +24V of power to the terminal
2. Connect ethernet cable between the terminal and the computer
3. Click network configuration page, configure the unit for a new fixed IP address appropriate to the local network. Default factory credentials user=User, password = <pee><ay><ess><ess><2*u><oh><ah><d><zero><zero>
4. Reset the card, connect to the intended unit and access the settings again at http:<your-new-ip-address>. If you make a mistake, there is a reset switch on the front panel which goes back to 128.1.1.100 - Configure to the appropriate RTD resistors and I/O mappings on the Test Page and I/O Mapping tabs from the main webpage menu. The units can be configured to output temperature directly in counts 0 to 32000 corresponding to a defined temperature range. However there is some rounding involved that brings the best possible resolution to 0.1 C. A better approach is to request counts 0 to 32000 in units of resistance, which may be converted later to give resolution 0.05 C.
5. hidden and not in the configuration is the option to take the raw ADC counts. Registers for this may be acessed directly (see section below).
6. Make sure you choose three wire or two wire options appropriately
7. There is a calibration page, don't worry about this too much
8. Set a password if you must.

Server machine (super-simple setup)

Install the latest stable release of Debian [https://www.debian.org/releases/] (preferable) or Ubuntu [https://www.ubuntu.com/download/desktop]. The desktop version is fine for our uses.

You should configure your box with:

- a fixed IP address and Subnet mask appropriate to the lab network
- A standard username (“controls”) and password
- Install tmux, git, vim etc using apt-get
- Mounted network directories for your user code and other useful things (if you want)

modbus installation

Application installation

DO NOT USE THE PREVIOUS HACKY METHOD OF COPYING BINARIES FROM RANDOM LIGO FOLDERS

Building from scratch is easy(ish) and will deliver something maintainable. The binaries LIGO has are poorly maintained and brittle.

Another option is to run the IOC process inside a container. This an be good for testing or deployments where you want to leave the host system pristine or unbroken from a build from scratch installation.

Building from scratch

There are three packages needed to get modbus over TCP/IP (acromags) working with and EPICS IOC process: the EPICS base [https://epics.anl.gov/base/index.php] and two modules that extend its function asyn [https://epics.anl.gov/modules/soft/asyn/] and modbus [http://cars9.uchicago.edu/software/epics/modbus.html]. The latest modbus package and its required modules [http://cars9.uchicago.edu/software/epics/modbus.html] must be downloaded from the official source. It is best to use the latest stable release of modbus and match with the recommend base and asyn.

To install you must download the source, install build tools, set some environment variables correctly and then compile. Depending on the speed of the machine this might take 15-60 minutes. Instructions are below

First update apt-get. You will either need to be root or use sudo, here we use sudo:

```
$ sudo apt-get update
$ sudo apt-get upgrade
$ sudo apt-get update
```

Now install the usual basic build tools

```
$ sudo apt-get --yes install curl g++ make libperl-dev libreadline-dev wget
```

Now we need to add some environment variables so make process knows where to look for and put things. Add the following to your .bashrc (this is located in your user home folder).

```
export EPICS_HOST_ARCH=$(/opt/epics/base/startup/EpicsHostArch)
export EPICS_ROOT=/opt/epics
export EPICS_BASE=${EPICS_ROOT}/base
export EPICS_BASE_BIN=${EPICS_BASE}/bin/${EPICS_HOST_ARCH}
export EPICS_BASE_LIB=${EPICS_BASE}/lib/${EPICS_HOST_ARCH}
```

Its best to put this at the top of the .bashrc so people can find it.

After saving restart the terminal. The export commands should make the variables available in shell, you can check that they are there and correct by running

```
$ env | grep EPICS
```

You should see all the above variables listed on the command line.

Download the source files from the Argonne National Laboratory website [https://epics.anl.gov] for epics. Here we use the recommended stable combination of modbus-R2-10-1 asyn4-32 base-3.15.5 (as of October 2018). You can use wget to get the files directly

```
$ cd ~/Downloads
$ wget https://epics.anl.gov/download/base/base-3.15.5.tar.gz
$ wget https://www.aps.anl.gov/epics/download/modules/asyn4-32.tar.gz
$ wget https://github.com/epics-modules/modbus/archive/R2-10-1.tar.gz -O modbus-R2-10-1.tar.gz
```

Now make a directories to build epics and its modules into:

```
$ mkdir /opt/epics
$ mkdir /opt/epics/modules
```

and untar the downloaded files into these directories

```
$ tar xvzf base-3.15.5.tar.gz -C /opt/epics/
$ tar xvzf asyn4-32.tar.gz -C /opt/epics/modules/
$ tar xvzf modbus-R2-10-1.tar.gz -C /opt/epics/modules/
```

You can delete the original tar files after this point if you like.

Its best if you also symlink these directories so simple (unversioned) filenames. This makes navigating easier and makes the final build version agnostic.

```
$ ln -s /opt/epics/base-3.15.5 /opt/epics/base \
$ ln -s /opt/epics/modules/asyn4-32 /opt/epics/modules/asyn \
$ ln -s /opt/epics/modules/modbus-R2-10-1 /opt/epics/modules/modbus \
```

You also need to tweak the configuration of the asyn and modbus modules so that they know where and how the EPICS base was built. For asyn navigate the the file called /opt/epics/modules/asyn/configure/RELEASE and comment out the lines for IPAC and SNCSEQ. Then change SUPPORT=* **to /opt/epics/modules (or whatever your build dir is). Also change EPICS_BASE**=* to /opt/epics/base. The RELEASE file in the asyn module configure director should look something like

```
#RELEASE Location of external products

SUPPORT=/opt/epics/modules
-include $(TOP)/../configure/SUPPORT.$(EPICS_HOST_ARCH)

#  IPAC is only necessary if support for Greensprings IP488 is required
#  IPAC release V2-7 or later is required.
#IPAC=$(SUPPORT)/ipac-2-14

# SEQ is required for testIPServer
#SNCSEQ=$(SUPPORT)/seq-2-2-4

#  EPICS_BASE 3.14.6 or later is required
EPICS_BASE=/opt/epics/base
-include $(TOP)/../configure/EPICS_BASE.$(EPICS_HOST_ARCH)
```

You will need to make similar changes in the RELEASE confige file in the modbus modules directory (/opt/epics/modules/modbus/configure/RELEASE in our case). The final file should looks something like

```
#RELEASE Location of external products
# Run "gnumake clean uninstall install" in the application
# top directory each time this file is changed.
#
# NOTE: The build does not check dependancies on files
# external to this application. Thus you should run
# "gnumake clean uninstall install" in the top directory
# each time EPICS_BASE, SNCSEQ, or any other external
# module defined in the RELEASE file is rebuilt.

TEMPLATE_TOP=$(EPICS_BASE)/templates/makeBaseApp/top

SUPPORT=/opt/epics/modules
-include $(TOP)/../configure/SUPPORT.$(EPICS_HOST_ARCH)

ASYN=$(SUPPORT)/asyn

# If you don't want to install into $(TOP) then
# define INSTALL_LOCATION_APP here
#INSTALL_LOCATION_APP=<fullpathname>

# EPICS_BASE usually appears last so other apps can override stuff:
EPICS_BASE=/opt/epics/base
-include $(TOP)/../configure/EPICS_BASE.$(EPICS_HOST_ARCH)

#Capfast users may need the following definitions
#CAPFAST_TEMPLATES=
#SCH2EDIF_PATH=
```

Adapt anything if you have a custom build or the newer stable release is sightly different.

Ok, its time to build. First build the base

```
$ cd /opt/epics/base
$ make
```

If this goes well they you can also build the asyn module

```
$ cd /opt/epics/modules/asyn
$ make
```

Finally build the modbus module

```
$ cd /opt/epics/modules/modbus
$ make
```

If this all build properly then you should be good to go. You will need to test using another machine with EPICS tools installed. See configuration and testing section for how to actually start a modbus EPICS IOC process.

A good final step is to clean up a bit

```
$ apt-get clean
```

Make an alias

Also, you might like to add an easy cmdline alias for simple one line start of the modbus server:

```
alias startAcromag='~/modbus/bin/linux-x86_64/modbusApp  ~/modbus/iocBoot/iocTest/acromag.cmd'
```

This line can be added to .bashrc or .bash_aliases and will make launching a breeze.

Running from pre-built Docker

Docker [https://www.docker.com/] is a modern containerized platform that makes it easy to build isolated packaged builds that are operating system agnostic. This can be a very fast way to get a Modbus IOC EPICS process up and running quickly for testing. You can either start with a prebuilt recipe called a DOCKERFILE that contains explicit build instructions or download a pre-compiled image from Docker Hub [https://hub.docker.com/]. An EPICS IOC that supports modbus can be setup in just a few lines of the command prompt on any server, test-rig or laptop for testing or a more permanent deployment.

To use the docker option, first install Docker using these instructions [https://docs.docker.com/install/linux/docker-ce/debian/]

Now to test pull a prebuild docker image that has all the modbus stuff already installed:

```
$ docker pull andrewwade/modbusepicsdocker
```

And run a test instance with the default channel

```
$ docker run -dt --name testiocdocker -p 5064:5064 -p 5065:5065 -p 5064:5064/udp -p 5065:5065/udp andrewwade/modbusepicsdocker
```

If you run a caget command on a machine on the local network you should see the channel and a value of 1.2345

```
$ caget C3:EXP-modbusDocker
C3:EXP-modbusDocker       1.2345
```

Setting up an acromag server is just as easy. You need two sets of files [https://github.com/fincle/modbusEPICSDocker/tree/master/modbus], the IOCStart.cmd script file that runs at start and a some .db files to define your channel names and their math.

Some example files are given in the git modbus folder [https://github.com/fincle/modbusEPICSDocker/tree/master/]. For an Acomag XT1221 ADC card uncomment the XT1221 block and change the IP address to that of your acromag card's. You might also want to change the "ADC_Reg_XXXX" name to something descriptive of your application.

Next add a line like

```
dbLoadDatabase("/home/modbus/defaultXT1221.db")
```

to the bottom to call a .db file with all the details of your channels.

Using the default defaultXT1221.db provided, just rename channels and labels as needed.

To spin it up you need to bind mount these files into the /home/modbus directory of the docker container at startup.

```
$ docker run -dt --name MyDockerContainerName -v /exact/path/to/IOCStart.cmd:/home/modbus/IOCStart.cmd -v  /exact/path/to/db/files/dir:/home/modbus -p 5064:5064 -p 506
```

or, if you built locally, the name of your local image. Here the ports used by the IOC process to answer EPICS channel requests are explicitly given as 5064 and 5065 mappings from inside to outside the docker. Another (simpler) way to make the host network interface accessable to the inside of the docker is to use a host network bridge.

```
$ docker run -dt --name MyDockerContainerName -v /exact/path/to/IOCStart.cmd:/home/modbus/IOCStart.cmd -v  /exact/path/to/db/files/dir:/home/modbus --network host andr
```

This only works on linux computers.

If the Acromag card is available on the network and you've configured it right you should be able to see the channels broadcast on the network.

Further detailed instructions can be found in ATF:2249 [https://nodus.ligo.caltech.edu:8081/ATF/2249] with more hints in TCS:222 [https://nodus.ligo.caltech.edu:8081/TCS_Lab/222]

modbus IOC configuration

To run a modbus IOC, you need to (a) create a .cmd file to configure the modbus IOC and (b) create a database file of properly configured channels to access the channels.

These files should be stored in one of the user directories on the network that is mounted on the Ubuntu machine.

myiocconfig.cmd example

The following text illustrates how to create a configuration file for a modbus IOC. Also, see Keith Thorne's explanation on the DCC T1400200 [https://dcc.ligo.org/LIGO-T1400200].

```
epicsEnvSet("IOC", "myiocconfig")
epicsEnvSet("ARCH","linux-x86_64")
epicsEnvSet("TOP","${EPICS_MODULES}/modbus")
epicsEnvSet("MDBTOP","${EPICS_MODULES}/modbus")
dbLoadDatabase("$(MDBTOP)/dbd/modbus.dbd")
modbus_registerRecordDeviceDriver(pdbbase)

# Use the following commands for TCP/IP
# drvAsynIPPortConfigure(const char *portName,
#                        const char *hostInfo,
#                        unsigned int priority,
#                        int noAutoConnect,
#                        int noProcessEos);
drvAsynIPPortConfigure("c3test1","10.0.1.42:502",0,0,1)

# modbusInterposeConfig(const char *portName,
#                       modbusLinkType linkType,
#                       int timeoutMsec,
#                       int writeDelayMsec)
modbusInterposeConfig("c3test1",0,5000,0)

#drvModbusAsynConfigure(portName, (used by channel in DB file to reference this port)
#                       tcpPortName,
#                       slaveAddress,
#                       modbusFunction, (read = 4, write = 6)
#                       modbusStartAddress, (ADCs are numbered 0-7, DACs are numbered 1-8 - brilliant, huh?)
#                       modbusLength, (length in dataType units - basically the number of channels?).
#                       dataType, (16-bit signed integers = 4)
#                       pollMsec, (how frequently to request a value in [ms] - beware of making this too small with averaging on)
#                       plcType);
drvModbusAsynConfigure("ADC_Reg","c3test1",0,4,0,8,4,32,"Acromag")

drvAsynIPPortConfigure("c3test2","10.0.1.41:502",0,0,1)
modbusInterposeConfig("c3test2",0,5000,0)
drvModbusAsynConfigure("DAC_Reg","c3test2",0,6,1,8,4,0,"Acromag")
# set up the binary outputs using function code "5"
drvModbusAsynConfigure("BIO_Reg","c3test2",0,5,0,4,0,0,"Acromag")
dbLoadDatabase("./IOCTEST.db")

iocInit
```

The tricky part is getting the correct values into the three commands: drvAsynIPPortConfigure, modbusInterposeConfig and drvModbusAsynConfigure.

Temperature sensing card (966EN-4006)

For the 966EN temperature sensing unit there are two sets of registers that may be accessed. The first is counts (0-32000) in units calibrated in the card configuration (see Card Configuration). These may be temperature (i.e. value = 100C/32000*counts), alterativlty you can just get raw resistance values (i.e. 500 ohm/32000*counts) These are in registers 30012 to 30017. Raw A/D count values may be accessed in registers 30018 to 30023 (these can be harder to work with, the manual doesn't give much detail of their calibration). However, when calling the address in the .cmd config file, you must subtract one from the starting register. That is, to get the calibrated output of the card you set the modbusStartAddress= 11 and not 12 for calibrated units and 17 for the raw count.

An appropriate set of commands for a 966EN-4006 acromag card at IP 10.0.0.43 would be

```
drvAsynIPPortConfigure("c3test3","10.0.0.43:502",0,0,1)
modbusInterposeConfig("c3test3",0,5000,0)
drvModbusAsynConfigure("TEMP_Reg","c3test3",0,4,11,6,4,32,"Acromag") # register 40011 is channel 0 in direct units of resistance
```

This accesses a card at 10.0.0.43 (port 502). It has a 5000 time out delay. It is set to read (=4) and starts at addresss *300***11** for a total of 6 channels. Be sure not to set the polling rate too high with averaging on.

IOCTEST.db file

```
record(ai, "C3:ACROMAG_INPUT0")
{
        field(SCAN, ".1 second")
        field(FLNK, "C3:ACROMAG_INPUT0_C.PROC")
        field(DTYP, "asynInt32")
        field(INP, "@asynMask(ADC_Reg 0 -16)MODBUS_DATA")
}
record(ai, "C3:ACROMAG_INPUT1")
{
        field(SCAN, ".1 second")
        field(FLNK, "C3:ACROMAG_INPUT1_C.PROC")
        field(DTYP, "asynInt32")
        field(INP, "@asynMask(ADC_Reg 1 -16)MODBUS_DATA")
}
record(ao, "C3:ACROMAG_OUTPUT0")
{
        field(DTYP, "asynInt32")
        field(OUT, "@asynMask(DAC_Reg, 0, -16)MODBUS_DATA")
        field(LINR,"NO CONVERSION")
        field(EGUL, "-163.84")
        field(EGUF, "163.835")
        field(HOPR, "163.835")
        field(LOPR, "-163.84")
        field(PREC, "2")
        field(SCAN, ".1 second")
}
record(bo, "C3:ACROMAG_BO0")
{
        field(DTYP,"asynUInt32Digital")
        field(OUT,"@asynMask(BIO_Reg, 0, 0x1)")
        field(ZNAM,"zero")
        field(ONAM,"one")
        field(SCAN, ".1 second")
}
```

Hardware set up

The terminals need to be set up on a DIN-rail, powered with 24V and one in a row must be connected to the network switch.

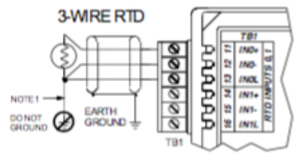
note on binary outputs:

- binary outputs require a separate supply voltage DEXC for the digital outputs. This can be between 6V and 32V.
- The 10K pulldown resistor in the BI/O units can cause problems with TTL logic if not properly accounted for. See here [http://nodus.ligo.caltech.edu:8080/PSL_Lab/1573]

for details.

Hardware config of 966EN-4006 (6 chan RTD module)

The acromag temperature sensor unit may be configured for either 2-wire or 3-wire resistive temperature sensing modes on each of its six channels. The unit provides a fixed 1 mA of supply current across the resistor and then reads out the resistance on a 0 to 0.5 V ADC. The three wire option (preferable) uses a network of three matched wires to cancel the resistive variations of the leads. This leaves only changes due to the resistance and temperature. The wiring is illustrated below. Two of the leads are connected to the drain of the resistor and the unit passes a current through that loop to determine its resistance. That potential value is assumed to be the same as the network in series with the RTD resistor: this fact is used to cancel the offset and variations of the transmission leads.



The key things to get right are:

- Select a thermistor with a resistance that is within range of the ADC (0.5 V) for the given 1 mA current for the temperature operating range. A typical choice would be a 100 Ohm Pt (alpha=1.385), also suggested are Ni 120 Ohm and Cu 10 Ohm;
- Choose identically matched wires of the same length leading to the thermistor;
- Connect them to the thermistor as illustrated above and keep the wires bunddled together so their variations are all in common mode;
- Ground all the remaining IND+ to the IND- and INDL terminals with a resistor (maybe something low ~10-100 Ohm) for best results.

Terminal mappings are in the section below Terminal Mappings

Running it all

Run a tmux session:

```
controls@hartmannFG:~$ tmux

controls@hartmannFG:~$ cd <ioc.cmd directory>
controls@hartmannFG:</ioc dir>g$ ${EPICS_MODULES}/modbus/bin/${EPICS_HOST_ARCH}/modbusApp myiocconfig.cmd
ctrl+b, d
```

'Ctrl+b' followed by 'd' detaches from the tmux session.

Check you can access the channels.

```
caget C3:ACROMAG_INPUT1
caput C3:ACROMAG_OUTPUT0 3000
```

Channel calibrations

Still working on this. The DAC channels are configured to:

- -30,000 counts → -10V
- +30,000 counts → +10V

The following channel calibration for DAC channels was found to be accurate.

```
C = 3003.5 [C/V] * V_req -20.8285 [C]
```

This code block also illustrates a possible routing of an digital value to the output in correctly calibrated units. That is, the voltage requested in C3:PSL-RCAV_COMMON_GAIN_V is the actual voltage that will appear at the output pin on the XT1541.

```
record(ai, "C3:PSL-RCAV_COMMON_GAIN_V")
{
        field(SCAN, ".1 second")
}
record(calcout, "C3:PSL-RCAV_COMMON_GAIN_C")
{
        field(SCAN, ".1 second")
        field(OUT, "C3:ACROMAG_OUTPUT0")
        field(CALC,"A*3003.5-20.8285")
        field(INPA,"C3:PSL-RCAV_COMMON_GAIN_V")
}
record(ao, "C3:ACROMAG_OUTPUT0")
{
        field(DTYP, "asynInt32")
        field(OUT, "@asynMask(DAC_Reg, 0, -16)MODBUS_DATA")
        field(LINR,"NO CONVERSION")
        field(EGUL, "-163.84")
        field(EGUF, "163.835")
        field(HOPR, "163.835")
        field(LOPR, "-163.84")
        field(PREC, "2")
        field(SCAN, ".1 second")
}
```

The temperature sensing card 966EN-4006 can output values in counts 0 to 32000 calibrated to resistance 0 to 500 Ohms. From there a soft calcout record can be used to convert into units of temperature with knowledge of the RTD device attached.

Adding slow channels to frames

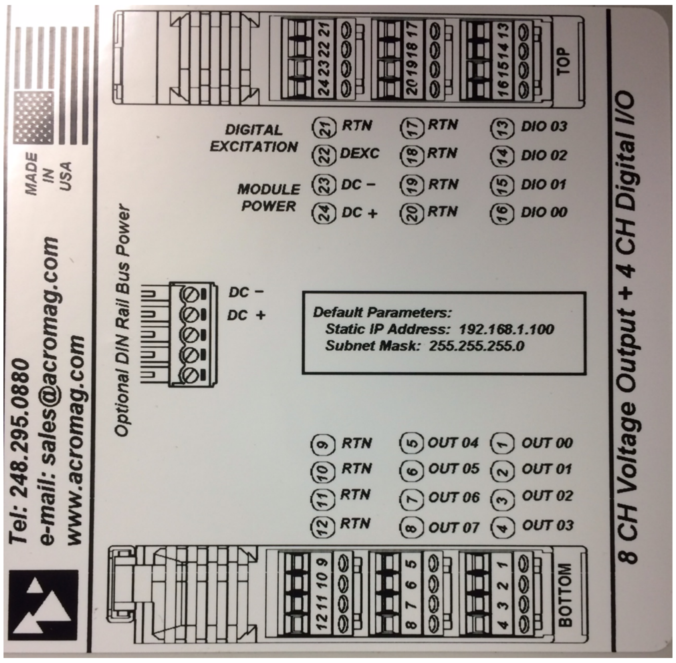
1. Edit the slow channels file in /cvs/cds/caltech/chans/daq, e.g. C2ATF_EPICS.ini

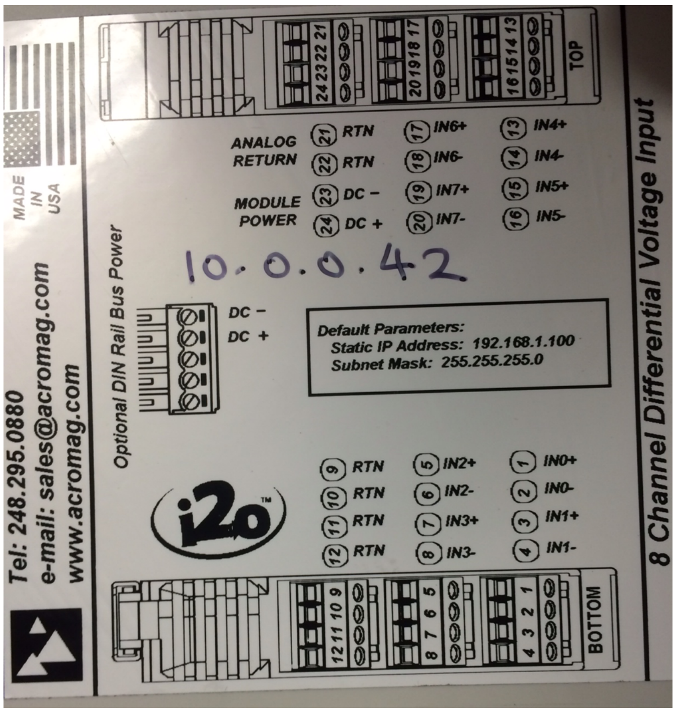
2. Make sure that file is called in /cvs/cds/caltech/target/fb(1?)/master
3. restart the daqd process on FB0 https://nodus.ligo.caltech.edu:30889/ATFWiki/doku.php?id=main:resources:computing:data_acq:add_chans [https://nodus.ligo.caltech.edu:30889/ATFWiki/doku.php?id=main:resources:computing:data_acq:add_chans]

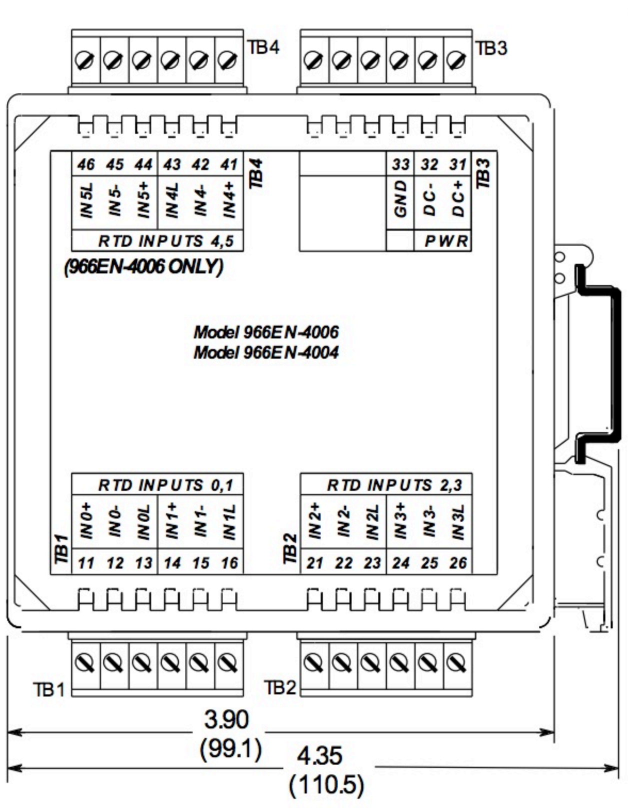
Recovering from a terminal power/communication outage

If the units lose power the IOC should keep running. Once they regain power the I/O connections should resume after about 10-30s.

Terminal maps







Further reading

Some further documentation

Driver Support for Modbus Protocol under EPICS [http://cars.uchicago.edu/software/epics/modbusDoc.html#Creating_a_modbus_port_driver]

Templates for all different types of possible channels [https://github.com/epics-modules/modbus/tree/master/modbusApp/Db]