attocube systems

OBSERVATORY SCIENCES

ECC100 Piezo Motion Controller

EPICS Developers Manual

Issue:	1
Date:	2nd February 2013

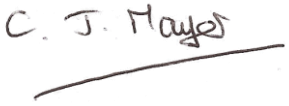
	NAME	DATE	SIGNATURE
Prepared by	Alan Greer, Observatory Sciences Ltd.	2 February 2013	
Checked by	Chris Mayer, Observatory Sciences Ltd.	7 February 2013	
Released by	Alan Greer, Observatory Sciences Ltd.	15 February 2013	

TABLE OF CONTENTS

1 Scope

This document describes the installation and use of EPICS device support code for the Attocube Systems' Piezo Motion Controller (ECC100). The ECC100 is a state-of-the-art motion controller, allowing the simultaneous operation of up to three positioners from attocube's industrial ECS Drive series. It can be controlled through an ethernet interface and this document describes the EPICS software written to control the unit via the ethernet interface.

2 Reference Documents

[RD1] IOC Application Developer's Guide, *Marty Kraimer et al.*
[RD2] EPICS R3.14 Channel Access Reference Manual, *Jeffrey O. Hill.*
[RD3] attocube systems' ECC100 Piezo Motion Controller User Manual, *attocube systems.*
[RD4] asynDriver: Asynchronous Driver Support, *Marty Kraimer, Eric Norum and Mark Rivers.*
[RD5] Motor Record and related software, *Tim Mooney, Joe Sullivan, Ron Sluiter.*

3 Introduction

The ECC100 EPICS software module contains device support code for longin and longout records to be used with an ECC100 motion controller (and associated boards). The device support code allows all command and status requests to be issued through these two records; longin records are used to request data and longout records are used to send commands (with data if necessary). The module contains two EPICS database template files, one for global control and one specifically designed to drive a stepper positioner. Also contained are edm screens that can be used to interact with a database generated by the template database files. A demonstration application is provided, set up to work with an ECC100 controller and three positioners.

EPICS motor record support is also included, providing a well-recognized interface to drive the stages using the standard EPICS motor record and GUI.

4 Requirements

This section details the hardware and software requirements for using the ECC100 EPICS device support code.

4.1 Hardware Requirements

The following hardware is required to run the device support code:

- Standard PC with an ethernet port.
- ECC100 present on the network. For operation with the EPICS device support code the ECC100 should be configured to have a static IP address. It is also required that the ECC100 device is setup not to send events.

4.2 Software Requirements

The following software is required to run the driver and device support code:

- Linux or Microsoft windows 32 bit operating system. The device support code has been written using general asyn and EPICS base function calls and methods and as a result will be fully functional on any system that can compile the required versions of EPICS base and the asynDriver module. The device support code was written on CentOS 5.7 and tested on CentOS 5.7 and Windows XP. A built executable was also run on Windows 7 64 bit.
- EPICS (version 3.14.12 or later)
- EPICS module ‘AsynDriver’ (version 4.17 or later)
- MSI extension for EPICS (version 5 or later). Required if building the supplied test application
- EDM to run engineering screens, if they are required.
- EPICS module ‘motor’ (version 6.7.1 or later) required to drive the axes using the standard motor record.

5 Installation

5.1 Installing Under Linux

This section will cover installation of the device support code for the ECC100 on a Linux operating system. Before performing the following installation, ensure that EPICS version 3.14.12 or later is installed and the asynDriver version 4.17 or later and the module motor version 6.7.1 or later have been installed.

Unpack the ECC100 tar file and cd into the top directory. In here you will see the following files and directories:

Directory	Description
ecc100App	This directory contains the device support code, a template database for a positioner and a template edm screen to use with the database template.
eccTest100App	This directory contains a test application. A substitutions file sets up a system configured for an ECC100 with three positioners.
ecc100MotorApp	This directory contains the motor record interface code.
configure	The configuration directory.
iocBoot	The boot directory for the test application. It contains the startup script that defines the IP address of the ECC100 (see below).
Makefile	
runGUI	Linux only. Bash script to start the edm application and open the test edl screen.

Before making the application it is necessary to add references to the EPICS installation, the asyn installation and the motor record installation. Cd into the configure directory and edit the RELEASE file. Update the lines that define the location of the asyn installation and the motor module to point to wherever these modules are installed on your system. For example

```
ASYN=/usr/software/epics/asyn-4.17
MOTOR=/usr/software/epics/motor-6.7.1
```

Also update the line that defines the location of your EPICS installation. For example

```
EPICS_BASE=/usr/software/epics/base-3.14.12
```

Save any changes and exit. Cd back up to the top level. Finally the host architecture must be defined before the build can commence. From a terminal enter the following line:

```
export EPICS_HOST_ARCH=linux-x86
```

There is a test application supplied with the device support code. If this is to be used then the startup script should be altered. Cd into the iocBoot/ioceccTest100 directory. Edit the st.cmd file and update the line that configures the asyn IP port with the IP address of the ECC100 unit on your network.

```
drvAsynIPPortConfigure("IP1","10.2.2.75:2101",0,0,0)
```

Now cd back to the top level and type make to build the device support code and the test application. The build should complete with no errors.

5.2 Installing Under Windows

This section will cover installation of the device support code for the ECC100 on a Windows operating system. Before performing the following installation, ensure that EPICS 3.14.12 or later is installed and the asynDriver version 4.17 or later has been installed. For the ECC100 test these were both installed using the mingw (Linux-like) environment and the gcc compiler along with Microsoft Visual C++ Express and Strawberry Perl. All instructions in this section assume the developer is also using a similar setup. For instructions please visit www.aps.anl.gov/epics/base/win32.php.

Download and unpack the ECC100 tar file. Cd into the top directory. In here you will see the following files and directories:

Directory	Description
ecc100App	This directory contains the device support code, a template database for a positioner and a template edm screen to use with the database template.
eccTest100App	This directory contains a test application. A substitutions file sets up a system configured for an ECC100 with three positioners.
ecc100MotorApp	This directory contains the motor record interface code.
configure	The configuration directory.
iocBoot	The boot directory for the test application. It contains the startup script that defines the IP address of the ECC100 (see below).
Makefile	
runGUI	Linux only. Bash script to start the edm application and open the test edl screen.

Before making the application it is necessary to add references to the EPICS and installation and the asyn installation. Cd into the configure directory and edit the RELEASE file. Update the lines that define the location of the asyn installation and the motor module to point to wherever these modules are installed on your system.

```
ASYN=C:/epics/asyn-4.17
MOTOR=C:/epics/motor-6.7.1
```

Also update the line that defines the location of the EPICS installation.

```
EPICS_BASE=C:/epics/base-3.14.12
```

Save any changes and exit. Cd back up to the top level. Finally the host architecture must be defined before the build can commence. From a terminal enter the following line:

```
export EPICS_HOST_ARCH=win32-x86
```

There is a test application supplied with the device support code. If this is to be used then the startup script should be altered. Cd into the iocBoot/ioceccTest100 directory. First edit the Makefile and ensure the ARCH variable is set to win32-x86, by default it will be setup for a linux build. Then edit the st.cmd file and update the line that configures the asyn IP port with the IP address and port number of the ECC100 unit on your network (see Section 7.1). The example below shows an ECC100 on a network with the IP address 10.2.2.71 listening on port 2101. The details of the ECC100 IP and port configuration can be found in [RD3].

```
drvAsynIPPortConfigure("IP1","10.2.2.71:2101",0,0,0)
```

Now cd back to the top level and type make to build the device support code and the test application. The build should complete with no errors.

6 Running the Test Application

Once the system has successfully compiled on either a Windows or Linux OS the test application should be executed to ensure a connection to the ECC100 is completed.

6.1 Running the EPICS Database

To run the IOC in either Windows or Linux cd into the iocecc100 directory and execute the st.cmd script. For windows the st.cmd script should be passed into the executable if it is to be run from a batch file. Below is an example of the output generated when the system is started on Linux.

```
#!../../bin/linux-x86/eccTest100
## You may have to change ecc100 to something else
## everywhere it appears in this file
< envPaths
epicsEnvSet("ARCH","linux-x86")
epicsEnvSet("IOC","iocecc100")
epicsEnvSet("TOP","/export/home/ajg/applications/epics/ecc100")
epicsEnvSet("ASYN","/export/home/ajg/applications/epics/asyn4-17")
epicsEnvSet("MOTOR","/export/home/ajg/applications/epics/motorR6-7-1")
epicsEnvSet("EPICS_BASE","/export/home/ajg/applications/epics/R3-14-12-
1/base")
cd /export/home/ajg/applications/epics/ecc100
## Register all support components
dbLoadDatabase "dbd/eccTest100.dbd"
eccTest100_registerRecordDeviceDriver pdbbase
## Create the Asyn IP port
drvAsynIPPortConfigure("IP1","10.2.2.75:2101",0,0,0)
## Create the ECC motor record layer
ecc100AsynMotorCreate("IP1","0","0","3")
Creating ECC100 motor driver on port IP1, address 0: card: 0, naxes: 3
drvAsynMotorConfigure("ECC1", "ecc100AsynMotor","0","3")
```

```
## Load record instances
dbLoadRecords "db/eccTest.db", "P=T1,PORT=IP1,DEV=ECC1"
cd /export/home/ajg/applications/epics/ecc100/iocBoot/iocecc100
iocInit
Starting iocInit
############################################################################
## EPICS R3.14.12.1 $Date: Tue 2011-04-26 15:36:19 -0500$
## EPICS Base built Jun 10 2011
############################################################################
iocRun: All initialization complete
epics>
```

Once the startup is complete and the records loaded the ECC100 can be controlled by setting various records present in the database. The main record name and device port macros are set during the startup procedure to allow for a quick integration of the test application into a facility. These can be changed by editing the st.cmd file.

6.2 Running the EDM Screens

To allow fast testing some edl engineering screens have been added; these can be started on Linux by executing the script runGUI present in the top-level directory. The screens can be converted to other formats (http://www.aps.anl.gov/epics/) if required to run on Windows but this has not been done within the ECC100 module as the edl screens can be run on Linux and will connect to an IOC running on Windows.

Once started the user is presented with the main screen. From this screen individual stepper control screens can be opened as well as the global control screen. The test application contains records for three positioners and each is represented on this display.

Figure 1 Main edl screen

Clicking on the global button opens the global parameters screen.

6.3 Global EDM Screen

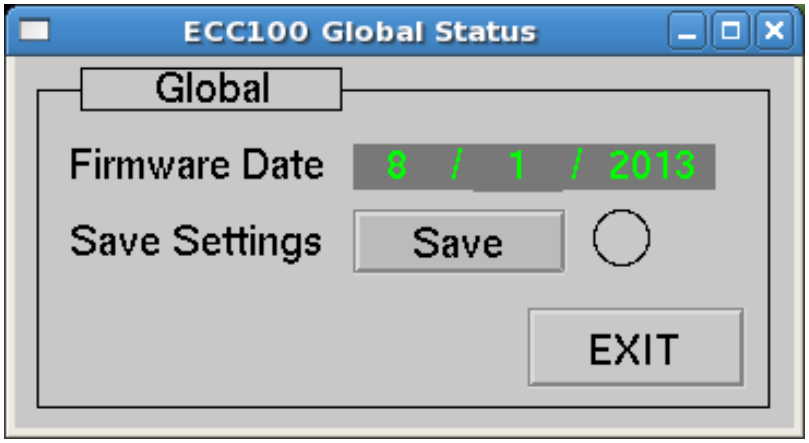


Figure 2 Global parameters screen

This screen contains the date of the current firmware version executing on the device. There is also a button that can be used to save any parameters currently entered on the device. The LED is lit whilst the save takes place. Note the save executes very quickly and so the LED might in practice not light up as the refresh rate is not fast enough.

6.4 Single Axis EDM Screen

From the main display clicking on one of the step module buttons opens the step display for the corresponding positioner.

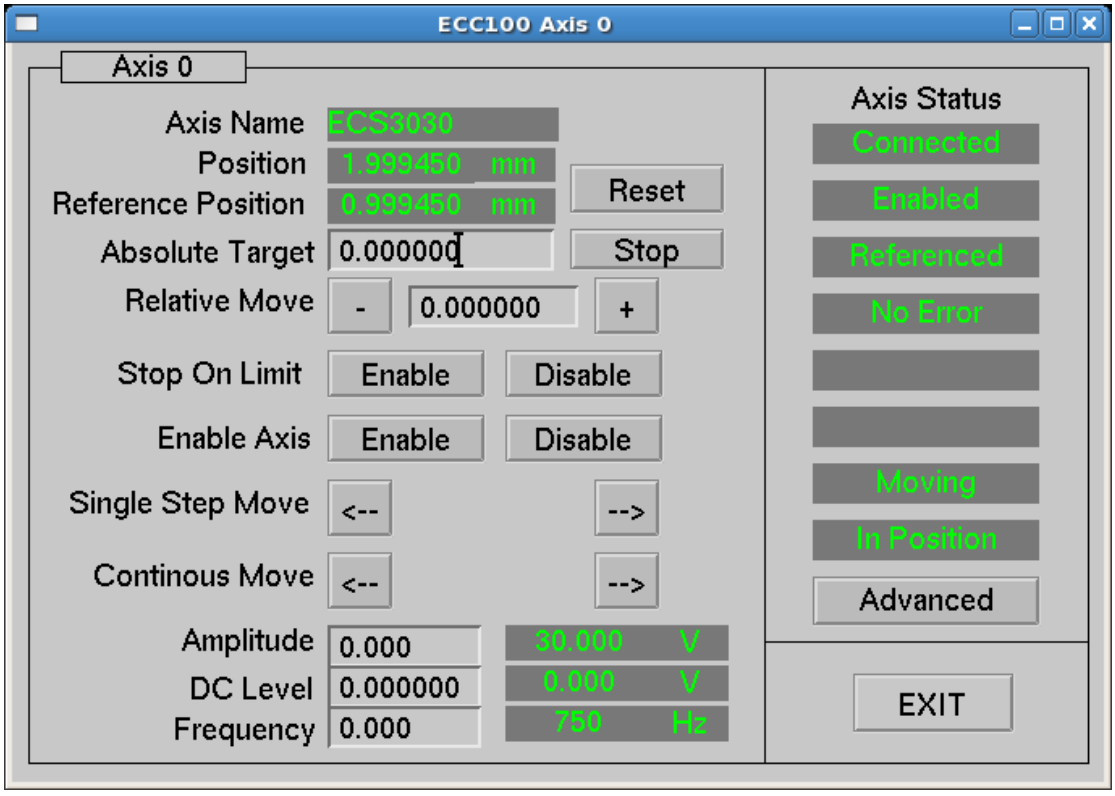


Figure 3 Single axis edm screen

The screen presents the following indicators and controls:

- Axis name. The currently selected positioner type is displayed in this indicator.
- Position. This indicator displays the current position of the positioner connected to the respective axis followed by the unit of this value.
- Reference. Each positioner has its individual reference position, a physical marker on the grating. The displayed value corresponds to the distance between the reference position and the currently set zero position. To get the reference position after starting the system, the positioner has to travel over the marker only once.
- Absolute Target. Entering a value here will result in the positioner moving to this value. The move is executed immediately.
- Relative Move. Entering a value in here sets up a relative movement step size. To move the positioner by either the +ve or –ve amounts click on the corresponding button.
- Stop On Limit. This forces the axis to drop out of move mode when it encounters a high or low limit. It can be enabled or disabled.
- Enable Axis. The positioner axis can be enabled or disabled from here.
- Reset. Resets the current position to zero. This results in a loss of the reference position.
- Stop. Takes the positioner out of active move towards the requested position.
- Single Step Move. Click either of these buttons to move the positioner by a single step.
- Continuous Move. Click and hold either of these buttons to move the positioner continuously. The positioner will move until the button is released.

- Amplitude. Value for the drive voltage of this Piezo. By changing this value the step size of the positioner can be varied.
- DC Level. The applied DC voltage to the Piezo.
- Frequency. Here the desired frequency can be entered with which the positioner should move, it is proportional to the speed.
- Axis Status. This consists of eight indicators:
    - 1. Green if the positioner is connected.
    - 2. Green if the axis is enabled.
    - 3. Green if the positioner is referenced.
    - 4. Green if there are no errors in the signal of the sensor.
    - 5. This contains red text ("High Limit") if the positioner reaches a high limit.
    - 6. This contains red text ("Low Limit") if the positioner reaches a low limit.
    - 7. The status of the positioner (moving or not moving).
    - 8. The in-position status of the positioner (in position or not in position).
- Advanced. Clicking on the Advanced button opens the axis advanced screen for the positioner (see section 6.5).
- Exit. Click on this to close the screen.

6.5 Axis Advanced EDM Screen

From the axis display clicking on the "Advanced" button opens the advanced options display for the corresponding positioner.

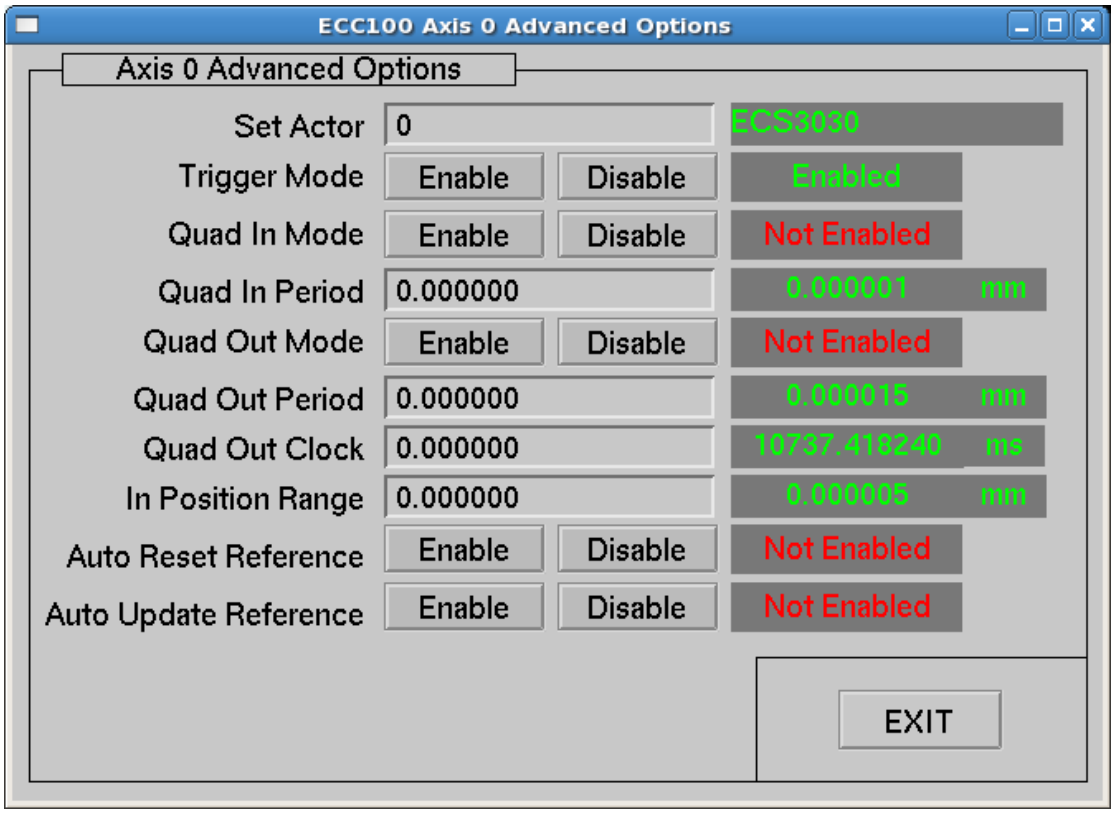


Figure 4 Axis advanced options edm screen.

The screen presents the following indicators and controls:

- Set Actor. The currently selected positioner can be set from this input. The type is set as an integer value but the corresponding name of the positioner is displayed next to the entry box.
- Trigger Mode. Controls the input trigger for steps. Enabled or disabled.
- Quad In Mode. Controls the AQuadB input for the set-point parameter. Enabled or disabled.
- Quad In Period. Controls the AQuadB input resolution for the set-point parameter. Units depend on the positioner type.
- Quad Out Mode. Controls the AQuadB output for position indication. Enabled or disabled.
- Quad Out Period. Controls the AQuadB output resolution for position indication. Units depend on the positioner type.
- Quad Out Clock. Controls the clock for AQuadB output. The clock can be set in multiples of 20ns with a minimum of 40ns.
- In Position Range. When this value is set and the positioner is moving towards a target then the in position flag will be set to true when the positioner moves within this range. This value is required for the EPICS motor record to work and so will be set to 10nm unless the value is already set from these records.
- Auto Reset Reference. When the positioner is referenced the reference position and current position are set to zero.
- Auto Update Reference. The positioner will be re-homed whenever it crosses the home mark. Enabled or disabled.

7 Using the Device Support Code

The simple test application described in section 6 presents all of the possible commands and status items available through the ethernet interface of the ECC100 motion controller. However, application developers can easily create their own databases of records; this section explains what is required to use the device support for the ECC100.

7.1 Configuration of EPICS Records and Application

The ECC100 uses only integer values when writing to and reading from its memory locations. Therefore only two record types are required to use the device support; they are the longin and longout record types. Longin records are used to request data from the controller and Longout records are used to issue commands.

To set up a record for use with the ECC100 ensure the devEcc100.dbd file has been included in the build. Set the DTYP field of the record to “ECC100”. Then the OUT or INP links need to be configured with the following information:

- Port Name. This must be the same name that is set in call into asyn in the startup script (see below). This is used by the asyn layer to ensure the record attempts communication with the correct controller.

- Signal Number. This is either the number of the axis for axis specific commands/requests, or zero for global commands/requests. It must be prefixed with the letter 'S'.
- Memory Address. This is a hexadecimal format number that represents the location in the ECC100 that should be written to/read from. A complete listing of the locations and their description can be found in Appendix A.

As an example, to read the current position of axis 0 you would use a Longin record and could set the INP link to

```
@$(PORT) S0 0x3000
```

$(PORT) would need to be replaced with the name of the port created through asyn registration.

Two template database files are included with the device support code and they present many examples of the use of Longin and Longout records to read and set the data inside the ECC100.

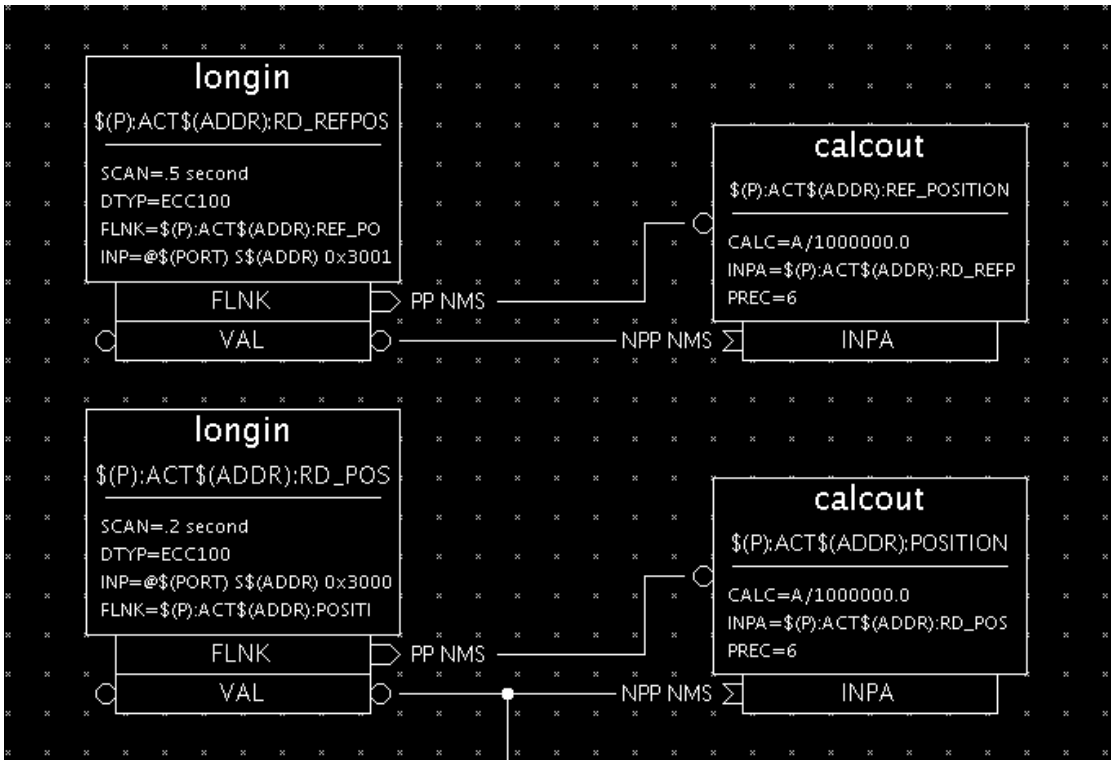


Figure 5 Reading the position and reference position from the ECC100

As well as setting up the necessary records there are some libraries that need to be added into the source Makefile of the application (found in the directory <appName>/src/). Figure 5 shows the complete source Makefile used for the test application supplied with the ECC100 device support code.

```
TOP=../..

include $(TOP)/configure/CONFIG
#----------------------------------------
#  ADD MACRO DEFINITIONS AFTER THIS LINE
#============================

#============================
# build an ioc application
PROD_IOC = eccTest100

DBD += eccTest100.dbd
# eccTest100.dbd will be made up from these files:
eccTest100_DBD += base.dbd
eccTest100_DBD += asyn.dbd
eccTest100_DBD += motorRecord.dbd
eccTest100_DBD += motorSupport.dbd
eccTest100_DBD += drvAsynIPPort.dbd
eccTest100_DBD += devEcc100.dbd
eccTest100_DBD += ecc100AsynMotor.dbd

# <name>_registerRecordDeviceDriver.cpp will be created from
<name>.dbd
eccTest100_SRCS += eccTest100_registerRecordDeviceDriver.cpp
eccTest100_SRCS_DEFAULT += eccTest100Main.cpp
eccTest100_SRCS_vxWorks += -nil-

#The following adds support from base/src/vxWorks
eccTest100_OBJS_vxWorks += $(EPICS_BASE_BIN)/vxComLibrary

# Include the following libraries in the build
eccTest100_LIBS += asyn
eccTest100_LIBS += motor
eccTest100_LIBS += ecc100
eccTest100_LIBS += ecc100AsynMotor
eccTest100_LIBS += $(EPICS_BASE_IOC_LIBS)

#===========================

include $(TOP)/configure/RULES
#----------------------------------------
#  ADD RULES AFTER THIS LINE
```

Figure 6 Source Makefile for ECC100 test application

Finally, an asyn IP port must be configured at startup to connect to the ECC100 device; for this the IP address and port number of the device must be known. For the test application the startup script is shown below.

```
#!../../bin/linux-x86/eccTest100

## You may have to change ecc100 to something else
## everywhere it appears in this file

< envPaths
```

```
cd ${TOP}

## Register all support components
dbLoadDatabase "dbd/eccTest100.dbd"
eccTest100_registerRecordDeviceDriver pdbbase

## Create the Asyn IP port
drvAsynIPPortConfigure("IP1","10.2.2.75:2101",0,0,0)

## Create the ECC motor record layer
ecc100AsynMotorCreate("IP1","0","0","3")
drvAsynMotorConfigure("ECC1", "ecc100AsynMotor","0","3")

## Load record instances
dbLoadRecords "db/eccTest.db", "P=T1,PORT=IP1,DEV=ECC1"

cd ${TOP}/iocBoot/${IOC}
iocInit
```

Figure 7 Startup script for the ECC100 test application

The drvAsynIPPortConfigure function call sets up the asyn port. For detailed information see the asyn manual ([RD4]). The first argument supplied is the name of the port; this defines the port name that should be used in all longin and longout record INP and OUT fields. The second argument is a string that contains the IP an address and port number to connect to. A colon separates these. The next argument is the priority; a value of 0 indicates the use of epicsThreadPriorityMedium. The next argument is the “auto connect” argument and a value of 0 here ensures the asyn layer automatically connects to the port. The last argument is the noProcessEos argument. These arguments can of course be configured as required.

8 Motor Record Support

This section deals with using the motor record to drive individual axes on the ECC 100. When creating a new application ensure the asyn, motor and ecc100 modules are all included in the configure/RELEASE file. The test application startup script already makes the necessary motor record specific calls.

```
drvAsynIPPortConfigure("IP1","10.2.2.71:2101","0","0","0")
ecc100AsynMotorCreate("IP1","0","0","3")
drvAsynMotorConfigure("ECC1", "ecc100AsynMotor","0","3")
```

The first line configures the asyn layer to communicate with the ECC 100 over an ethernet connection. This requires a name for the connection, followed by the IP address and port number of the ECC hardware. After this there are some configuration parameters that can be left at zero. For a full description of this function call see the asyn driver manual [RD4].
The second line creates the ECC 100 specific device driver. Its first argument must be the same name supplied to the previous function call. The address and card number follow this. Finally the number of axes should be supplied to this call. In the example above the device has been created with three axes.
The third and final call required sets up the motor record device support layer for the motor record. The first parameter can be any name assigned to this device (the name is used in the motor record itself, see below). The next parameter must be "ecc100AsynMotor". This ensures the motor device support uses the ECC 100 driver functions to attempt communications with the axes. The final two parameters supplied to this function call are the card number and the number of axes.

8.1 Motor Record Database Configuration

To add motor records to a database in the application simply add an entry to a suitable substitutions file that makes use of the motor record template. An database template file called eccMotor.template is provided with the application. An example of a suitable substitution for the motor record is provided below:

```
file ../../../db/eccMotor.template {
pattern
{P,   M,   DESC, DTYP, DIR, VELO, VBAS, ACCL, BDST, BVEL,
BACC, PORT, ADDR, MRES, PREC, EGU, DHLM, DLLM, INIT}
{T1:,  M1,  Desc, asynMotor, 0,   600.0,  50.0,  1.0,  0.0,
0.0,  0.0,  ECC1,  0,    0.001,  3,    um,  5000.0,  -5000.0,
0}
{T1:,  M2,  Desc, asynMotor, 0,   600.0,  50.0,  1.0,  0.0,
0.0,  0.0,  ECC1,  1,    0.001,  3,    um,  5000.0,  -5000.0,
0}
{T1:,  M3,  Desc, asynMotor, 0,   600.0,  50.0,  1.0,  0.0,
0.0,  0.0,  ECC1,  2,    0.001,  3,    um,  5000.0,  -5000.0,
0}
}
```

The example substitution file above resulted in the following motor record

```
record(motor,"T1:M1")
{
      field(DESC,"Desc")
      field(DTYP,"asynMotor")
      field(DIR,"0")
      field(VELO,"300.0")
      field(VBAS,"50.0")
      field(ACCL,"1.0")
      field(BDST,"0.0")
      field(BVEL,"0.0")
      field(BACC,"0.0")
      field(OUT,"@asyn(ECC1,0)")
      field(MRES,"0.001")
      field(PREC,"3")
      field(EGU,"um")
      field(DHLM,"5000.0")
      field(DLLM,"-5000.0")
      field(INIT,"0")
      field(TWV,"1")
}
```

The most important field is the "OUT" field. This must be set to point to the ECC 100 motor support. This is achieved by setting the value to '@asyn(ECC1,0)'. The ECC1 should match the name defined in the startup script and the number after this represents the axis number (from 0 to n-1 axes).

Other motor record configuration is beyond the scope of this document: for further details, consult the documentation [RD5].

8.2 Using the Motor Record with the ECC100

Specific features available when using the ECC100 with the motor record include the following:

- **Velocity**. Changing the velocity will result in a change to the frequency of the axis.
- **Jog**. The motor can be jogged at a constant velocity in either direction.
- **Move**. Absolute and relative moves can be made.
- **Referencing**. The ECC100 positioners automatically reference when they first cross their reference marks. The motor record reference mode can be invoked but it only performs a move. It is recommended to always ensure the axis is homed before moving using the motor record as when the reference position is crossed the position of the motor record might change resulting in an unexpected final position. The motor record attempts to stop the lower level axis from moving when a reference is first discovered but this should not be relied upon.
- **Units**. The motor record described above is set up to use µm as the default unit. If the units were changed within the ECC controller then the motor record would need to be updated accordingly (EGU, PREC, MRES).

The figure below shows an example of a motor record running with an ECC 100 controller with a single axis.

Figure 8 Motor record control screen example.

Appendix A. Commands And Status Memory Locations

Below is a table of all commands and status locations within the ECC 100. Any listed below that do not specify units are simple on/off commands. Most simple commands operate when the value is supplied as 1, and some commands turn off when the value is supplied as 0. For example, a continuous move starts when its value is set to 1, and will only stop when the value is set to 0.

Location	Global	Description
0x000A	X	Synchronisation request. Interrupts the controller outputs for 2000ms to allow resynchronisation.
0x0145	X	Controls sending of events. 1 to turn events on and 0 to turn off.
0x3038	X	Read-only. Firmware version. 32 bit wide number which is the build date of the firmware coded in hex. 0xYYYYMMDD.
0x3030		Axis enabled.
0x3012		Amplitude of the axis in mV.
0x3013		Frequency of the axis in mHz.
0x3041		Control of fixed DC voltage. 1 is enabled. 0 is disabled.
0x3043		Fixed voltage output in µV.
0x302D		Resets actual position to zero and marks the reference position as invalid.
0x3000		Retrieves the current position. Either nm or µº.
0x3001		Retrieves the reference position. Either nm or µº.
0x3014		Selects the actor to be used on the selected axis. Value from 0 to 255.
0x3003		Name of selected actor, characters 16 - 19.
0x3004		Name of selected actor, characters 12 - 15.
0x3005		Name of selected actor, characters 8 - 11.
0x3006		Name of selected actor, characters 4 - 7.
0x3007		Name of selected actor, characters 0 - 3.
0x3033		Actor type. Linear, Gonio or Rotator.
0x3011		Target position for the approach function. Either nm or µº.
0x3027		Controls the approach of the actor to the target position. 1 is enable approach (move) and 0 is disable approach (stop).
0x3023		Triggers a single step in the forward direction. Value must be 1, write only.
0x3024		Triggers a single step in the backward direction. Value must be 1, write only.
0x3025		Controls continuous movement in forward direction. Any movement in the opposite direction is stopped. 1 to start movement, 0 to stop movement.
0x3026		Controls continuous movement in backward direction. Any movement in the opposite direction is stopped. 1 to start movement, 0 to stop movement.
0x3028	X	Save user parameters to persistent flash memory in controller.

		Parameters that are saved are amplitude, frequency and actor selection of each axis. Value must be 1.
0x302F	X	Retrieves the status of a flash operation. 1 indicates flash in progress, 0 otherwise.
0x302C		Retrieves the status of the reference position. It may be valid or invalid. (1 or 0).
0x302E		Retrieves the moving status. Moving means the actor is actively driven by the output stage either for approaching or continuous/single stepping.1 indicates moving and 0 indicates not moving.
0x3031		Retrieves the error status. 1 indicates a sensor malfunction. 0 indicates no problems.
0x3002		Retrieves the connected status. 1 indicates the actor is electrically connected to the controller. 0 indicates the actor is not connected.
0x3036		OEM feature. Defines the range around the target position in which the flag target status is set active. The value is either in nm or µº.
0x3037		OEM feature. Retrieves the target status. Indicates whether the actual position is within the specified target range. Required for use with the EPICS motor record.
0x3039		Pro feature. Retrieves the status of the end of travel (limit) in the forward direction.
0x303A		Pro feature. Retrieves the status of the end of travel (limit) in the backward direction.
0x304A		Pro feature. Defines the behaviour of the output on a limit. If enabled, the output of the axis will be deactivated when a limit is detected.
0x3042		Controls the input trigger for steps. 1 to enable, 0 to disable.
0x3044		Pro feature. Controls the AQuadB input for setpoint parameter. 1 to enable, 0 to disable.
0x3045		Pro feature. Controls the AQuadB input resolution. The value is either in nm or µº.
0x3046		Pro feature. Controls the AQuadB output for setpoint parameter. 1 to enable, 0 to disable.
0x3047		Pro feature. Controls the AQuadB output resolution. The value is either in nm or µº.
0x3048		Pro feature. Controls the clock for AQuadB output. Clock value is in multiples of 20ns with a minimum of 2 (40ns).
0x3034		OEM feature. Positioner is re-homed every time it crosses the reference mark. 1 to enable, 0 to disable.
0x3035		OEM feature. When the positioner is homed the relative position is set to zero, resulting in a reference mark position of 0. 1 to enable, 0 to disable.