# nPoint LC400 Motor Record User Manual

Rev 1.0

2019-11-17

# Table of Contents

# Introduction

This document describes the function and operation of the NPoint LC400 motor record IOC. The LC400 series DSP controllers are designed to control nPoint nanopositioners. They are available in 1, 2, or 3 channel configurations. It  includes information about how to install and configure the IOC, details about the motor record driver operation and how to interact with the provided user interface.

# Abbreviated terms

| **Abbreviation** | **Description** |
|---|---|
| Asyn | EPICS asynchronous module for low level communication drivers |
| CS-Studio | Control System Studio |
| EPICS | Experimental Physics and Industrial Control System |
| FTDI | Future Technology Devices International |
| IOC | Input Output Controller |
| OPI | Operator Interface for Control System Studio |
| PID | Proportional Integral Derivative Controller |
| PV | EPICS Process Variable |
| R/W | Read and Write |
| USB | Universal Serial Bus |

Table 1: Abbreviated terms

# Theory of operation

The nPoint LC400 motor record IOC commands point-to-point movement at a user-defined acceleration and velocity by uploading a custom waveform with the generated trajectory. The commanded movement is a trapezoidal velocity profile with constant acceleration until the set speed is achieved, followed by a constant deceleration to zero velocity. The controller increments through the positions with a minimum delay of 24 µsec. The number of points for the generated waveform can be controlled through a PV and the maximum value is 83,333 points. Increasing the number of points will improve the movement resolution but it will also increase the time to start the movement as more data will need to be transferred. To improve the update rate, the positions and limits status will be updated during the movement at the configured polling rate but the auxiliary parameters such as PID values will only update when the positioners are idle.

# Requirements

These sections will describe the hardware and software requirements for deploying and using the LC400 motor record IOC.

## Hardware Requirements

- Standard computer with a USB port.
- An nPoint LC400 controller with firmware version >= 4.00.00.94 connected to the computer.

## Software Requirements

The software was developed using standard EPICS modules and libraries. It is not guaranteed but it might work with later versions than the ones listed here. These are the versions used for development:

- Debian Stretch (8)
- libftdi-dev version 0.20-2 (if using the FTDI mode)
- EPICS base version 3.15.3
- EPICS asyn version 4.33 or 4.37 (if using the FTDI mode)
- EPICS motor version 7-1
- CS-Studio 4.5.9

# FTDI Library for USB

The LC400 controller utilizes an FTDI chip for its USB communication. The FTDI drivers can provide both a legacy virtual port and a direct interface mode (referred by FTDI as “D2XX”). The LC400 motor IOC can work with both modes utilizing the asynSerial or the asynFTDI interface. While the legacy mode can be useful if the FTDI library cannot easily be added, the direct interface mode (FTDI) is recommended for the fastest communication speeds.

# Software Installation

These steps cover how to deploy the system on a Linux machine running Debian Stretch:

- Make sure you have the motor controller powered on and connected to the USB port.
- From a terminal application execute *lsusb* and look for the FT232 output and take note of the product and vendor ID. e.g.:

  *Bus 003 Device 002: ID* ***0403*** *:* ***6014*** *Future Technology Devices International, Ltd*

  *FT232H Single HS USB-UART/FIFO IC*

- The default access for the nPoint device is set in a way that only root or a sudoer user can utilize the device. To enable R/W access to all users, a new udev rule needs to be written and installed under */etc/udev/rules.d.*

  Create /etc/udev/rules.d/99-ftdi.rules with the following content, replacing the IDs if necessary:

  ```
  # FTDI rules
  ATTR{idVendor}=="0403", ATTR{idProduct}=="6014", MODE="660",
  GROUP="dialout"
  ```

- Restart the computer to apply the new udev rule.
- Download and follow the installation instructions for the following modules:
  - https://github.com/epics-base/
  - https://github.com/epics-modules/asyn/
  - https://github.com/epics-modules/motor
- If you are planning to use the FTDI interface, make sure you have *libftdi-dev* installed as well. e.g.:
  - *sudo apt-get install libftdi-dev*

- Download the nPoint motor controller IOC:
  - *https://github.com/epics-motor/motorNPoint*

## Motor nPoint software installation

Before proceeding with this section, make sure you have all the required modules compiled and installed in a known location. The nPoint motor IOC has the following structure:

| **Directory** | **Description** |
|---|---|
| configure | This is the configuration directory. The RELEASE file should be edited with the dependency modules and their locations. The CONFIG_SITE or CONFIG_SITE.local file should have FTDI=YES if you want to use the FTDI interface. |
| docs | Documentation files and release notes |
| iocs | The test application |
| nPointApp | Source files and database location |

- cd into the configure directory and edit the RELEASE file. Make sure to add the following lines with the correct module locations:

```
ASYN=/usr/local/lib/epics

MOTOR=/usr/local/lib/epics

EPICS_BASE=/usr/local/lib/epics
```

- Save the file and create CONFIG_SITE.local file with the following flag to add the direct interface mode:

```
FTDI=YES
```

- Save the changes and go back to the TOP level, defining the EPICS_HOST_ARCH variable as follows:

```
export EPICS_HOST_ARCH=linux-x86_64
```

- Type *make* to build the support module and test application making sure it completes without any errors.

# Configuring the application

After the system has compiled successfully, navigate to *iocs/nPointIOC/iocNPoint* directory and modify the following files:

- *LC400.substitutions* – Contains the PV prefixes and software limits for each motor record channel.
- *LC400.cmd* – USB port configuration for either legacy (drvAsynSerialPort) or FTDI direct access mode (drvAsynFTDIPort), remove the comments for the method you'd like to use making sure it points to the correct hardware location. Edit the dbLoadRecords lines with the desired prefixes and addresses. Make sure that the LC400CreateController function has the right amount of channels and the correct asyn port.
- *st.cmd* – This is the file used to start the application, verify that the first line has the correct binary location and that the LC400.cmd line is not commented.

## LC400 startup commands

- `LC400CreateController`
  This command creates the controller object, and connects it to the low-level asyn port that interfaces to the device.

```
# LC400CreateController(
#       port name,
#       serial port,
#       num axes,
#       moving poll period (ms),
#       idle poll period (ms))
LC400CreateController("LC400", "LC400ftdi", 3, 100, 1000)
```

- `LC400ConfigAxis`

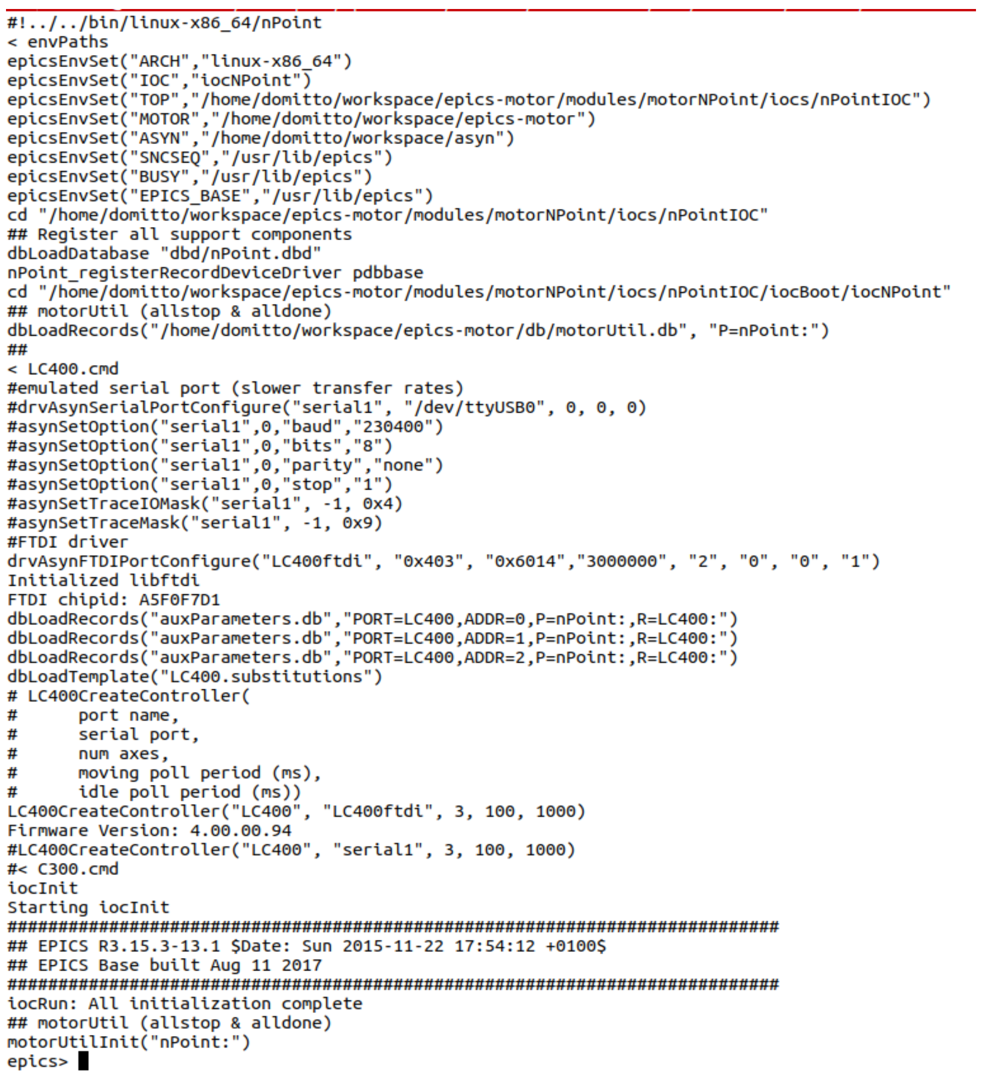
  This command is optional.

  It sets software values in the driver to simulate physical limit switch signals and defines the range that each axis is allowed to operate over. The EPICS motor record soft limits can be used as user-configurable software limits. The tolerance parameter is used to provide hysteresis at the limit position to prevent the measurement noise from making the limit status cycle rapidly between the on and off states.

```
#LC400ConfigAxis(
#       port name,
#       axis number,
#       positive hard limit (in µm, relative to controller
readback position)
#       negative hard limit (in µm, relative to controller
readback position)
#       tolerance for hard limits in µm (relative to hard limits
set above)
#)
LC400ConfigAxis("LC400ftdi", 0, 55, -55, 0.1)
LC400ConfigAxis("LC400ftdi", 1, 55, -55, 0.1)
LC400ConfigAxis("LC400ftdi", 2, 55, -55, 0.1)
```

## Running the application

To run the application execute the st.cmd command from the iocNPoint directory. To confirm correct communication with the nPoint controller, check that the firmware version and FTDI chipID have valid numbers. Below is an example of the output when the application is started correctly:

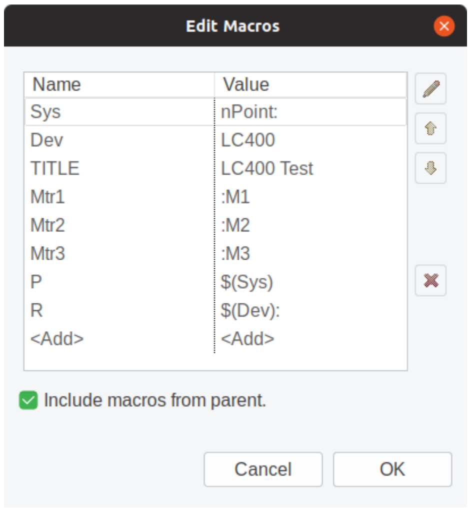


```
#!../../bin/linux-x86_64/nPoint
< envPaths
epicsEnvSet("ARCH","linux-x86_64")
epicsEnvSet("IOC","iocNPoint")
epicsEnvSet("TOP","/home/domitto/workspace/epics-motor/modules/motorNPoint/iocs/nPointIOC")
epicsEnvSet("MOTOR","/home/domitto/workspace/epics-motor")
epicsEnvSet("ASYN","/home/domitto/workspace/asyn")
epicsEnvSet("SNCSEQ","/usr/lib/epics")
epicsEnvSet("BUSY","/usr/lib/epics")
epicsEnvSet("EPICS_BASE","/usr/lib/epics")
cd "/home/domitto/workspace/epics-motor/modules/motorNPoint/iocs/nPointIOC"
## Register all support components
dbLoadDatabase "dbd/nPoint.dbd"
nPoint_registerRecordDeviceDriver pdbbase
cd "/home/domitto/workspace/epics-motor/modules/motorNPoint/iocs/nPointIOC/iocBoot/iocNPoint"
## motorUtil (allstop & alldone)
dbLoadRecords("/home/domitto/workspace/epics-motor/db/motorUtil.db", "P=nPoint:")
##
< LC400.cmd
#emulated serial port (slower transfer rates)
#drvAsynSerialPortConfigure("serial1", "/dev/ttyUSB0", 0, 0, 0)
#asynSetOption("serial1",0,"baud","230400")
#asynSetOption("serial1",0,"bits","8")
#asynSetOption("serial1",0,"parity","none")
#asynSetOption("serial1",0,"stop","1")
#asynSetTraceIOMask("serial1", -1, 0x4)
#asynSetTraceMask("serial1", -1, 0x9)
#FTDI driver
drvAsynFTDIPortConfigure("LC400ftdi", "0x403", "0x6014","3000000", "2", "0", "0", "1")
Initialized libftdi
FTDI chipid: A5F0F7D1
dbLoadRecords("auxParameters.db","PORT=LC400,ADDR=0,P=nPoint:,R=LC400:")
dbLoadRecords("auxParameters.db","PORT=LC400,ADDR=1,P=nPoint:,R=LC400:")
dbLoadRecords("auxParameters.db","PORT=LC400,ADDR=2,P=nPoint:,R=LC400:")
dbLoadTemplate("LC400.substitutions")
# LC400CreateController(
#       port name,
#       serial port,
#       num axes,
#       moving poll period (ms),
#       idle poll period (ms))
LC400CreateController("LC400", "LC400ftdi", 3, 100, 1000)
Firmware Version: 4.00.00.94
#LC400CreateController("LC400", "serial1", 3, 100, 1000)
#< C300.cmd
iocInit
Starting iocInit
############################################################################
## EPICS R3.15.3-13.1 $Date: Sun 2015-11-22 17:54:12 +0100$
## EPICS Base built Aug 11 2017
############################################################################
iocRun: All initialization complete
## motorUtil (allstop & alldone)
motorUtilInit("nPoint:")
epics> █
```

Figure 1: Successful IOC startup output

# Utilizing the test OPI

Open the main.opi file located under iocs/nPointIOC/nPointApp/op/lc400 and edit the macros according to your PV prefixes, e.g.:



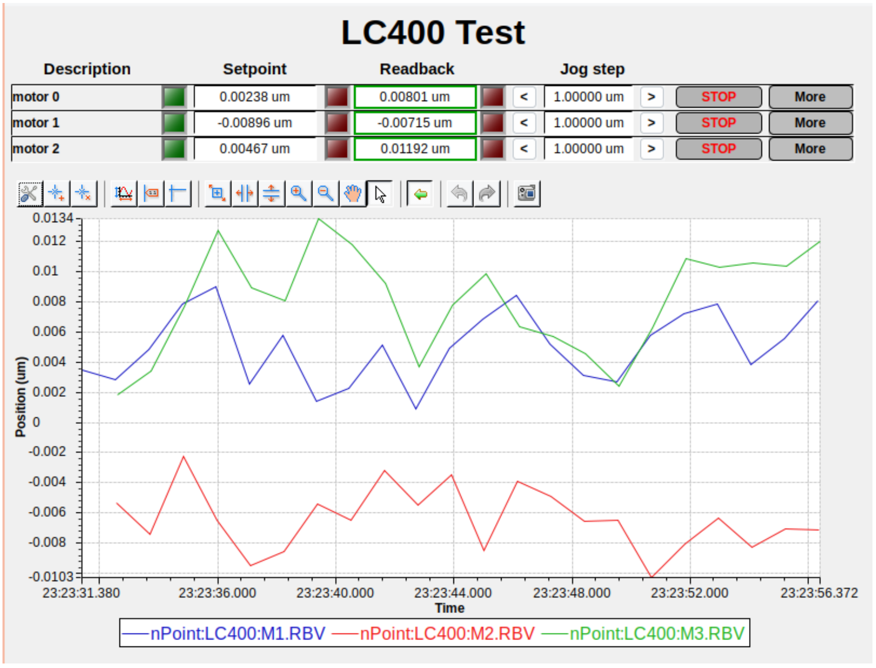
Edit Macros

| Name | Value |
|---|---|
| Sys | nPoint: |
| Dev | LC400 |
| TITLE | LC400 Test |
| Mtr1 | :M1 |
| Mtr2 | :M2 |
| Mtr3 | :M3 |
| P | $(Sys) |
| R | $(Dev): |
| <Add> | <Add> |

✓ Include macros from parent.

Figure 2: OPI macro settings

Run the main.opi file and the XY plot should update with the current positions from the controller:



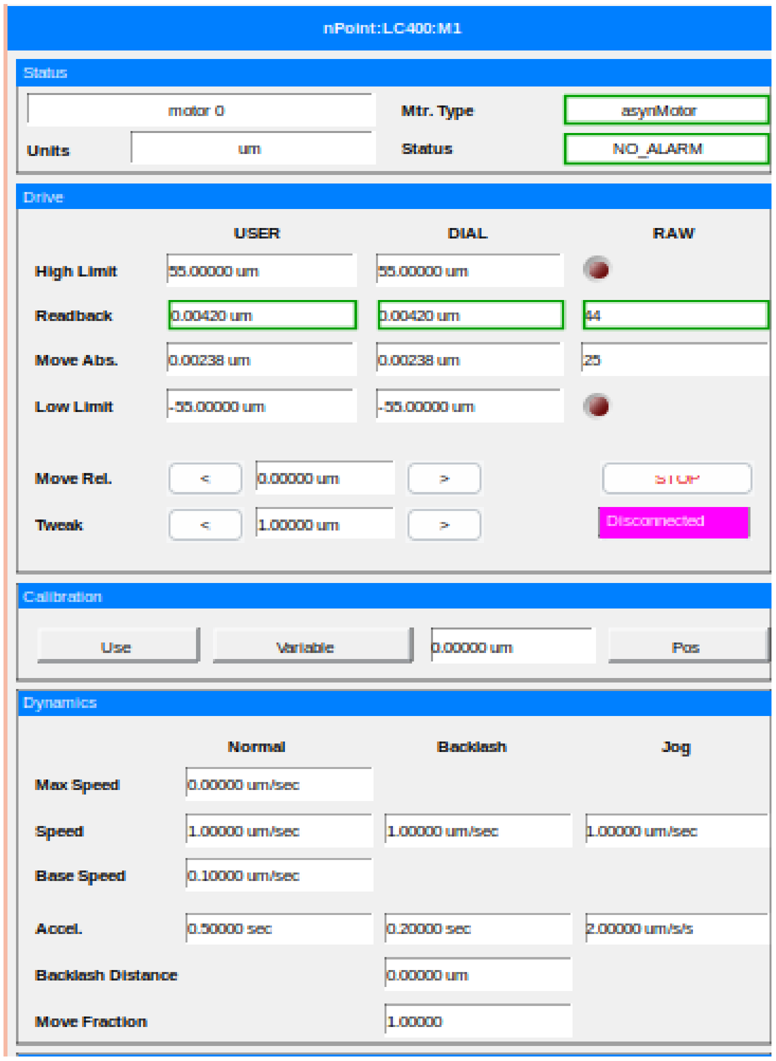
LC400 Test

| Description | | Setpoint | | Readback | | Jog step | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| motor 0 | 🟩 | 0.00238 um | 🟥 | 0.00801 um | 🟥 | < | 1.00000 um | > | STOP | More |
| motor 1 | 🟩 | -0.00896 um | 🟥 | -0.00715 um | 🟥 | < | 1.00000 um | > | STOP | More |
| motor 2 | 🟩 | 0.00467 um | 🟥 | 0.01192 um | 🟥 | < | 1.00000 um | > | STOP | More |

Figure 3: Main OPI screen

From this screen you can move each axis to a specific setpoint, see the limits and moving status and command the axis to Stop prior to move completion.

From **More→Motor Config** you can set specific motor record configurations for the axis such as acceleration and velocity:



| | | | |
|---|---|---|---|
| motor 0 | | Mtr. Type | asynMotor |
| Units | um | Status | NO_ALARM |

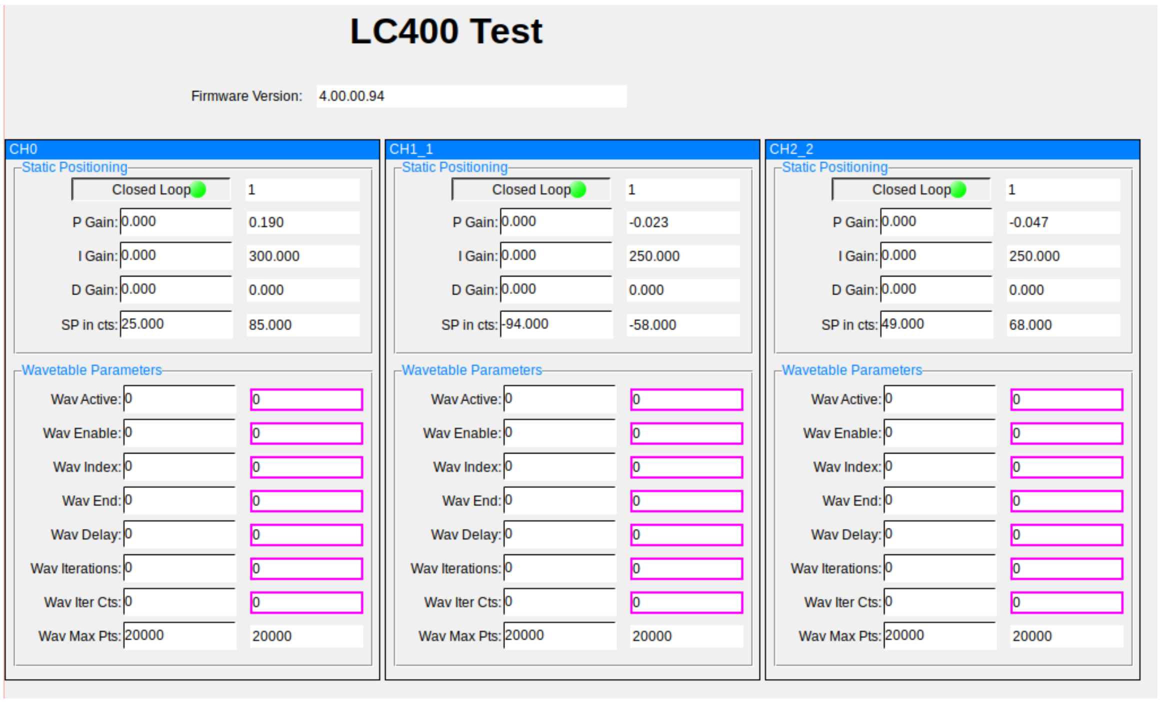
| | USER | DIAL | RAW |
|---|---|---|---|
| High Limit | 55.00000 um | 55.00000 um | ● |
| Readback | 0.00420 um | 0.00420 um | 44 |
| Move Abs. | 0.00238 um | 0.00238 um | 25 |
| Low Limit | -55.00000 um | -55.00000 um | ● |
| Move Rel. | 0.00000 um | | |
| Tweak | 1.00000 um | | |

| | | | |
|---|---|---|---|
| Use | Variable | 0.00000 um | Pos |

| | Normal | Backlash | Jog |
|---|---|---|---|
| Max Speed | 0.00000 um/sec | | |
| Speed | 1.00000 um/sec | 1.00000 um/sec | 1.00000 um/sec |
| Base Speed | 0.10000 um/sec | | |
| Accel. | 0.50000 sec | 0.20000 sec | 2.00000 um/s/s |
| Backlash Distance | | 0.00000 um | |
| Move Fraction | | 1.00000 | |

Figure 4: Advanced motor record configuration

From **More → Extra Parameters** you can set specific LC400 parameters such as PID values, static positioning movement, and wavetable parameters including the wavetable size. To increase the polling efficiency these parameters will only be updated when the axes are idle.



LC400 Test

Firmware Version: 4.00.00.94

|  | CH0 |  | CH1_1 |  | CH2_2 |  |
|---|---|---|---|---|---|---|
| **Static Positioning** |  |  |  |  |  |  |
| Closed Loop ● |  | 1 |  | 1 |  | 1 |
| P Gain: | 0.000 | 0.190 | 0.000 | -0.023 | 0.000 | -0.047 |
| I Gain: | 0.000 | 300.000 | 0.000 | 250.000 | 0.000 | 250.000 |
| D Gain: | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| SP in cts: | 25.000 | 85.000 | -94.000 | -58.000 | 49.000 | 68.000 |
| **Wavetable Parameters** |  |  |  |  |  |  |
| Wav Active: | 0 | 0 | 0 | 0 | 0 | 0 |
| Wav Enable: | 0 | 0 | 0 | 0 | 0 | 0 |
| Wav Index: | 0 | 0 | 0 | 0 | 0 | 0 |
| Wav End: | 0 | 0 | 0 | 0 | 0 | 0 |
| Wav Delay: | 0 | 0 | 0 | 0 | 0 | 0 |
| Wav Iterations: | 0 | 0 | 0 | 0 | 0 | 0 |
| Wav Iter Cts: | 0 | 0 | 0 | 0 | 0 | 0 |
| Wav Max Pts: | 20000 | 20000 | 20000 | 20000 | 20000 | 20000 |

Figure 5: Extra parameters OPI

# EPICS database description

The following records are available from the standard application and shown in the following table along with their description.

| **Record Name** | **Description** |
|---|---|
| $(P)$(M) | Standard motor record interface |
| $(P)$(R)CH$(ADDR):FW_RBV | String representing the motor controller firmware |
| $(P)$(R)CH$(ADDR):SP | 20 bit digital positioning command |
| $(P)$(R)CH$(ADDR):SP_RBV | 20 bit sensor reading for the current DSP cycle |
| $(P)$(R)CH$(ADDR):PG | Proportional gain of the control loop |
| $(P)$(R)CH$(ADDR):PG_RBV | Proportional gain readback of the control loop |
| $(P)$(R)CH$(ADDR):IG | Integral gain of the control loop |
| $(P)$(R)CH$(ADDR):IG_RBV | Integral gain readback of the control loop |
| $(P)$(R)CH$(ADDR):DG | Derivative gain of the control loop |
| $(P)$(R)CH$(ADDR):DG_RBV | Derivative gain readback of the control loop |
| $(P)$(R)CH$(ADDR):CL | A value 1 enables the servo loop, a value 0 disables the servo (sets the channel to open loop) |
| $(P)$(R)CH$(ADDR):CL_RBV | Servo control loop readback |
| $(P)$(R)CH$(ADDR):WAV_MAX_PTS | Maximum number of points for the wavetable |
| $(P)$(R)CH$(ADDR):WAV_MAX_PTS_RBV | Max number of points for the wavetable readback |
| The following PVS are useful for debugging purposes only | |
| $(P)$(R)CH$(ADDR):WAV_EN | Control the wavetable scanning |
| $(P)$(R)CH$(ADDR):WAV_EN_RBV | Wavetable scanning readback |
| $(P)$(R)CH$(ADDR):WAV_IDX | The index of the wavetable that will be output during the movement |
| $(P)$(R)CH$(ADDR):WAV_IDX_RBV | Wavetable index readback |
| $(P)$(R)CH$(ADDR):WAV_DLY | The number of clock cycles to wait before the next wavetable point is output |
| $(P)$(R)CH$(ADDR):WAV_DLY_RBV | Wavetable delay readback |
| $(P)$(R)CH$(ADDR):WAV_EIDX | Specify the last point that will be output during the wavetable movement |
| $(P)$(R)CH$(ADDR):WAV_EIDX_RBV | Wavetable last point readback |
| $(P)$(R)CH$(ADDR):WAV_ACT | Set to 1 as a software trigger to start the wavetable output |
| $(P)$(R)CH$(ADDR):WAV_ACT_RBV | Wavetable active readback |

| **Record Name** | **Description** |
|---|---|
| $(P)$(R)CH$(ADDR):WAV_ITER | Number of wavetable cycles that will be output |
| $(P)$(R)CH$(ADDR):WAV_ITER_RBV | Wavetable iteration readback |
| $(P)$(R)CH$(ADDR):WAV_ITER_CTS | Wavetable cycle to start the movement |
| $(P)$(R)CH$(ADDR):WAV_ITER_CTS_RBV | Wavetable cycle count readback |

Table 2: EPICS database description

Driver details
This section describes details about the main methods of the motor record driver and its interaction with the nPoint API.

## Initialization

During the IOC startup the driver will open either the serial or FTDI connection, check the firmware version and initialize both hard limits to 0 (disabled).

## Axis poll method

If the axis being polled is moving, the driver will only query the current position, check if the movement is done and, if defined in the *st.cmd*, check for the hard limit status.

When the axis is idle, all the PID parameters as well as the motor range will be updated.

## Axis move method

The move method will generate a trapezoidal movement from the motor record parameters up to *$(P)$(R)CH$(ADDR):WAV_MAX_PTS* value and configure the controller utilizing the wavetable addresses for the axis. The IOC will command the movement to start and wait until the controller gets to the second index of the array to write the final position to index 0 as the wavetable movement loops back to the beginning of the array after sending the last position.

The method will not implement any movement if the optional hard limit is set and the movement is in the same direction as an active limit.

## Axis move velocity

The IOC will command a regular move at a controlled speed to the maximum configured range. It can be either the configured hard limit or, if that’s not defined, the axis range from the controller.

# Troubleshooting

Below is a brief overview of the basics of troubleshooting the most common application problems. Following these steps can help identify or solve these problems.

## Building the application inside the motor record module

If you are trying to build the nPoint LC400 IOC inside the motor record module make sure you are editing the motor record configure directory instead of the LC400, the settings will then be loaded automatically for all the modules built. Also make sure that you set the flag BUILD_IOCS to YES in the motor record configure/CONFIG_SITE file.

## Firmware version not printing, serial communication problems

If using the FTDI direct access, make sure that your user has the correct access permissions (R/W) to the USB port created when connecting the controller to the computer. Check if the product and vendor ID matches the one in the udev rule file. Sometimes power cycling the motor controller also helps reestablishing the serial communication if the controller is stuck in a non-responsive state.

## Axis is moving erratically

Verify that the axes are in closed loop mode with appropriate values for the PID control loop.

Disable the RTRY field by in the motor record by setting its value to 0 as it conflicts with the nPoint closed loop control settings.

Verify that your nPoint motor controller has the required firmware version necessary for the wavetable movements (see Hardware Requirements).

## Movement takes too long to start

If you are using the legacy mode or have a big number of points set for the wavetable movement the USB transfer speed might be the limiting factor, it will take ~2 seconds to write 100k points utilizing the legacy mode and ~1 second utilizing the FTDI direct interface mode. You can change *$(P)$(R)CH$(ADDR):WAV_MAX_PTS* to a smaller number of points to compensate for that.