Hytec Electronics Ltd

Hytec IP8401 AsynDriver User Manual

For:

Diamond Light Source

APPROVED AND AUTHORISED	
AUTHORS *Jim Chen*	**CREATION DATE** *25th June 2009*

PRODUCT DESCRIPTION

Project name *Hytec IP8401 AsynDriver*

Release Issue: 4
Date: 07/08/2009

Author:	Jim Chen
Owner:	
Project Manager:	Graham Cross
Client:	Diamond Light Source
Document Number:	DLS/ASYNDRV/001

Document History

Document Location

The source of the document can be found in the Hytec SharePoint Linux Project – EPICS/AsynDriver/Documents

Revision History

Date of next revision:

Version Number	Revision date	Previous revision date	Summary of Changes	Changes marked
Issue1	29/06/2009		First Official Release	
Issue2	30/06/2009		Revised typos and errors	
Issue3	03/07/2009		Added support for MCA and EPID records	
Issue4	07/08/2009		1. Added initialization routine 2. Re-defined abnormal conditions for ai averaging	

Approvals

This document requires the following approvals.
Signed approval forms are filed in the project files.

Name	Signature	Title	Date of Issue	Version
Graham Cross		Hytec Project Manager	29 Jun 2009	Issue 1
Graham Cross		Hytec Project Manager	30 Jun 2009	Issue 2
Graham Cross		Hytec Project Manager	03 Jul 2009	Issue 3
Graham Cross		Hytec Project Manager	07 Aug 2009	Issue 4

Distribution

This document has been distributed to:

Name	Title	Date of Issue	Version
Hytec Linux Project Team			
Diamond Light Source			

Identifier	DLS/ASYNDRV/001
Title	Hytec IP8401 AsynDriver User Manual
Purpose	This document describes the functionalities and usage of Hytec IP8401 asynDriver that is designed for EPICS applications.
Composition	
Derivation	1. Generic ADC Driver Software requirements. Doc No. TDI-CTRL-REQ-015, issue 1.1, released on 13 Sep 2008 2. ADC8401 8-Channel 16-bit Industry Pack. Issue 5 version 516, released on 15 Jun 2009 3. VCB8002 VME64x Four Site Industry Pack Carrier Board. Issue 3 version 307, released on 08 Jan 2007
Format and Presentation	This document will be presented as Microsoft Word format.
Allocated to	Hytec Linux Project Team
Quality Criteria	
Quality check method to be used	Peer inspection Review
Quality check skills required	EPICS Core Record Support Device Support AsynDriver Device Driver Development VME64x and Industry Pack technology

Contents

1. Introduction

Please note: Hytec IP8401 asynDriver needs the firmware version of V516 or later.

Hytec 8401 industry pack is an 8 channel, 16bit simultaneous analogue to digital converter (ADC) with 1M byte (up to 2Mbyte) on board memory that gives each channel 64k word data storage.

This AsynDriver is designed as per Diamond Light Source requirements for supporting EPICS applications.

It has three operation modes: continuous acquisition mode, trigger mode and gated mode that are implemented by the following Asyn interfaces.

- asynDrvUser
- asynInt32
- asynFloat64
- asynInt32Array
- asynFloat64Array
- asynUInt32Digital
- asynCommon

The mode selection is initially determined by configuration routine during the system boot time. Once it is set, it applies to all 8 channels. During the runtime, the user can change the mode online without rebooting the IOC. However, some of the settings such as ENABLE, REENABLE etc are reset except the continuous mode.

Few other settings are set during this stage as well such as interrupt vector, clock rate, external or internal clock, number of samples to collect (for continuous and trigger mode only), average (for ai records only) and offset (for trigger and gated mode only) etc. Again, once these parameters have been set, they apply to all 8 channels.

All the above settings can also be altered at runtime except interrupt vector. Some of the runtime changes will result in the ADC to issue a software stop/restart. Whenever one of these settings is changed online, a notification is generated to the subscribed users (records) whoever monitors the changes.

For all three modes, the driver provides ADC data readings for both polled records (with SCAN set to “.5 second” for example) and asyn callback records (with SCAN set to “I/O Intr”) in both ai (averaged) and waveform forms. This will be described in detail in the following sections.

Few commands such as enable/disable ADC, re-enable, software trigger for trigger mode etc have been implemented by both asynInt32 and asynUInt32Digital interfaces. They both support bi, bo, mbbi and mbbo records. AsynUInt32Digital interface implementation is for backwards compatibility.

The user can interrogate driver status such as Overflow, AverageOverflow, gate/trigger state and support etc. These are implemented also by both asynInt32 and asynUInt32Digital interfaces and the latter is for backwards compatibility.

To accommodate MCA and EPID record support, two special reasons have been defined in the reason enumeration (see below) to match the FastSweep records' need. These are DATA (=0 in the enumeration) and SCAN_PERIOD (=2 in the enumeration).

Due to the single conversion notification requirement of the FastSweep record, an interrupt is generated on each single ADC conversion when requiring MCA and EPID support. This puts constrain to the ADC speed which is limited to 20 kHz. In the normal case of ADC usage (without MCA and EPID support) the speed can still go as high as 100 kHz. For this reason, one argument is defined in the init routine named "fastADC" to distinguish these two cases. When fastADC = 1, it is used as normal ADC which can have speed up to 100 kHz. When fastADC = 0, it is used for the support of MCA and EPID records.

A summary of services provided by the driver is listed below.

- Both polled and callback Int32 ai data – single channel raw ADC value (range: -32767 ~ +32767) averaged over a period of time. This can be obtained by using either DATA or GENDATA reason. They both return the same value.
- The callback data only notifies the user when the data collecting period expires or an event happens.
- Both polled and callback Int32 waveform data – an array of raw ADC values. Using reason DATA returns an array of 8 elements that are the 8 channel single ADC readings at the same moment. The NORD value of the record is set to 8. This is for MCA record support. When using reason GENDATA, it returns an array of NELM (number of elements) of a single channel over a period of time that the user is interested. This is the normal waveform case. The NORD is set to the true number of valid data in the array.
- Both polled and callback Float64 ai data – single channel calibrated and/or converted ADC value. When using reason DATA, which is for EPID support, the returned value is the calibrated ADC value (range: -32767 ~ +32767) but not converted to voltage. When using reason GENDATA, it returns the normal calibrated and converted ADC value (range: -10V ~ + 10V).
- Both polled and callback Float64 waveform data – an array of calibrated and converted ADC value (range: -10V ~ + 10V) over a period of time that the user is interested. It returns the same set of data defined by NELM no matter the reason is DATA or GENDATA. NORD is set to the true number of valid data in the array.
- Whenever the scanning rate is changed at runtime, a callback is generated that returns the scan period in second to records with reason of SCAN_PERIOD.

  Please note, this scan period value only gets returned when changing the scan rate at runtime. The initialisation doesn't set it.

- The user can interrogate average overflow, data overflow, gated/trigger mode state and driver support by using reason AVERAGEOVERFLOW, OVERFLOW, GATETRIGGERSTATE and SUPPORT respectively.
  - Average overflow (either 0 or 1) if set means something wrong with averaging for ai record.
  - Overflow (either 0 or 1) means either in trigger mode the period is set too big or in gated mode the gate is too long.

- Gated/trigger state has three states, 0 – waiting for trigger of gate opening, 1 – triggered or gate opened and acquiring, 2 – trigger period finishes or gate closed.
- Support provides the lowest three bits that tells the user that the driver supports: bit0 – gate mode, bit1 – trigger mode, bit2 – trigger mode negative offset. In Hytec 8401 asyn driver at the moment, only bit0 and bit 1 is set.

- Runtime parameter changes can be achieved by issuing the following commands. Each setting change will generate a notification callback to the user record which uses any one of the reasons described below. The return value for this callback is the enumeration number of the command reason.

  - Changing mode by using SET MODE reason.
  - Changing scan rate by using SETCLOCKRATE reason.
  - Changing external/internal clock by using SETEXTCLOK reason.
  - Changing average setting by using SETAVERAGE reason.
  - Changing samples setting by using SETSAMPLE reason.

- Driver controls is done by the following commands.

  - ADC enable by using ENABLE command. The ADC only runs when it is enabled. For trigger and gated mode the ADC only runs once after it is enabled unless the re-enable is also set as below.
  - ADC continuous enabled by using REENABLE command. If this is set, in trigger and gated mode, the ADC hasn't to be enabled every time.
  - Software trigger in trigger mode can be issued by using SOFTTRIGGER command.

These reason enumeration summary EPICS record is listed below.

• DATA	-- asking data for FastSweep records
• GENDATA	-- asking for normal data
• SCAN_PERIOD	-- asking for scan period in second
• SETAVERAGE	-- set average online
• SETSAMPLE	-- set samples online
• SETMODE	-- set mode online
• SETOFFSET	-- set offset online
• ENABLE	-- set/clear enable
• SOFTTRIGGER	-- software trigger
• REENABLE	-- set/clear reenable
• CLEARBUFFER	-- not implemented
• SETCLOCKRATE	-- set clock rate online
• SETEXTCLOCK	-- set internal/external clock online
• AVERAGEOVERFLOW	-- query average overflow state
• OVERFLOW	-- query overflow state
• BUFFERCOUNT	-- not implemented
• GATETRIGGERSTATE	-- query trigger/gated mode state
• SUPPORT	-- query driver support info

2. Installation

To install the IP8401 AsynDriver, download the package from Hytec website http://www.hytec-electronics.co.uk/Software/Hy8401ip-AsynDriver-17-11-2009-V1.9.tar.gz and unzip it to the EPICS support directory. Also download the support package from here http://www.hytec-electronics.co.uk/Software/Hy8401ipAsynSupport-18-11-2009.tar.gz and unzip it to the EPICS support directory as well.

The example of the IP8401 AsynDriver in the package is built for VxWorks with power PC vxWorks-ppc604_long architecture.

To be able to build sucessfully, some simbols and environment variables such as “SUPPORT”, “EPICS_BASE” in RELEASE under each configure directory need to be chanaged accordingly.

The architecture needs to be changed if it is different from the example by changing the CROSS_COMPILER_TARGET_ARCHS in CONFIG file under configure directory.

The support package includes an ipac module speicifically designed for Hytec carrier cards. So it is essential to have this module installed to be able to use Hytec IP cards.

All other modules in the support package are used for supporting mca and EPID records. If you are not intending to use these two records, those modules need not be installed. Yet it doesn’t do any harm to the system even though you install them.

3. Initialisation and Start Script

Hytec IP8401 AsynDriver initialisation routine has the following prototype. It is called during the start up script.

```
int Hy8401ipAsynInit(char *portName, int vmeSlotNum, int
     ipSlotNum, int vectorNum, int samples, int average, int
     offset, int scanMode, int clockRate, int extClock, int
     fastADC)
```

where: portName: char* -- the asynDriver port name

vmeSlotNum: int – VME slot number of the 8002 carrier card (2 ~ x, x depends on the crate)

ipSlotNum: int – 8401 IP card slot number (0 ~ 3)

vectorNum: int – interrupt vector

samples: int – the number of ADC readings to generate the asyn callback for I/O Intr records. This argument only applies to contiuous and trigger mode.

average: int – the number of points to be averaged for ai records.

offset: int – defines the relative to the time of trgger/gate the data of interest. This only applies to trigger and gated mode. For trigger mode, it can only be a positive value. For gated mode, it can be positive or negative. A postive offset means the start point of interest is offseted by this setting from the beginning of the gate. A negative offset means that the interest points take “offset” number of points at the end of the gate.

scanMode: int – 0 = contiuous mode, 1 = trigger mode and 2 = gated mode.

clockRate: int – scanning rate of ADC. It has the following settings.

0 – 1 Hz	1 – 2 Hz	2 – 5 Hz	3 – 10 Hz
4 – 20 Hz	5 – 50 Hz	6 – 100 Hz	7 – 200 Hz
8 – 500 Hz	9 – 1 kHz	10 – 2 kHz	11 – 5 kHz
12 – 10 kHz	13 – 20 kHz	14 – 50 kHz	15 – 100 kHz

extClock: int – either use internal clock (= 0) or external clock (= 1)

fastADC: int – defines either to use high speed ADC (= 1, normal case, up to 100 kHz without MCA and EPID support) or MCA and EPID support (= 0, up to 20 kHz). The only differences of the two are the scanning speed and in MCA/EPID support case, the driver generates asyn callback every single conversion whereas the normal case doesn't. All the other functions, mode and record supports are the same.

There is a config routine defined in the asyn driver but it is not used at the moment. Its prototype looks like this.

```
int Hy8401ipAsynConfig(const char *portName, int aiType, int
     range, int firstChan, int lastChan, int chanBuffSize )
```

None of these arguments is used in the asyn driver. The user can ignore this routine (even don't have to call it during the initialisation process).

A typical start up script for VxWorks OS target looks like this.

```
#!$(INSTALL)/bin/$(ARCH)/example

cd "$(INSTALL)"

# Load binaries on architectures that need to do so.
# VXWORKS_ONLY, LINUX_ONLY and RTEMS_ONLY are macros that resolve
# to a comment symbol on architectures that are not the current
# build architecture, so they can be used liberally to do architecture
# specific things. Alternatively, you can include an architecture
# specific file.

$(VXWORKS_ONLY)ld < bin/$(ARCH)/example.munch

#$ define the maximum EPICS channel access array size

epicsEnvSet(EPICS_CA_MAX_ARRAY_BYTES,"3000000")

## Register all support components

dbLoadDatabase("dbd/example.dbd")
example_registerRecordDeviceDriver(pdbbase)


# Run the configuration function once for each card in the IOC
############################################################
# Configure Hytec 8002 carriers
#   8002 carrier VME slot 3
#   ARGS - vmeslotnum, IPintlevel, HSintnum
#

$(VXWORKS_ONLY)IPAC3=ipacEXTAddCarrier(&EXTHy8002, "3 2 18")
```

```
###########################################################
#int Hy8401ipAsynInit(char *portName, int vmeSlotNum, int ipSlotNum, int
#     vectorNum, int samples, int average, int offset, int scanMode, int
#     clockRate, int extClock, int fastADC)
# ARGS: vmeslot=3, ipslot=1, vector=10, samples=10000, average=50, offset=0,
#       scanmode=1(trigger), clockrate=9(1kHz), extclock=0(no), fastADC=0(
#       mca & EPID support)

Hy8401ipAsynInit("myasynport1", IPAC3, 1, 10, 10000, 50, 0, 1, 9, 0, 0)

###########################################################
#int Hy8401ipAsynConfig(const char *portName, int aiType, int range, int
#     firstChan, int lastChan, int chanBuffSize )
# currently no effect for this routine in Hytec 8401ip asyn driver

Hy8401ipAsynConfig("myasynport1", 0, 1, 0, 7, 100)

###########################################################
# int initFastSweep(char *portName, char *inputName,
#                   int maxSignals, int maxPoints)
# portName   = asyn port name for this port
# inputName  = name of input port
# maxSignals = maximum number of input signals.
# maxPoints  = maximum number of points in a sweep.  The amount of memory
#              allocated will be maxPoints*maxSignals*4 bytes

$(VXWORKS_ONLY)initFastSweep("8401Sweep1","myasynport1", 8, 10000)


###########################################################
# Hytec 8402 DAC in IP site B of the IP carrier card in slot 3.
# This 8402 is used for analogue output of the EPID loop control

$(VXWORKS_ONLY)Hy8402ipConfigure (302, IPAC3, 2, 11)
initHy8402ipAsyn("DAC", 302)

###########################################################
## Load record instances

dbLoadRecords("db/example.db","P=CARD1,PORT=myasynport1")
dbLoadRecords("db/examplemca.db")
dbLoadRecords("db/exampleepid.db")

###########################################################
# set trace output level for asyn port "myasynport1"
# Level: 0x01 = Errors only
# asynSetTraceMask arguments:
#  * asyn port
#  * address of that asynport (i.e. channel number)
#  * verbosity level:
#                     0x01: error,
#                     0x11: errors, warnings and debug
#                     0x00: silent

asynSetTraceMask( "myasynport1", 0, 0x00 )

# all driver level messages

iocInit()
```

4. Operation Modes

Hytec IP8401 AsynDriver supports three operation modes: continuous mode, trigger mode and gated mode.

4.1. Continuous mode

Description

In continuous mode, the ADC starts acquisition immediately after the IOC is initialised, i.e. it is enabled automatically at the beginning. Two important configuration parameters are associated to this mode: “average” and “samples”.

“average” is used for ai records in both polled and callback cases. It is the number that the driver uses to average the acquired ADC values then returned to the records.

“samples” is used for waveform records which specifies the maximum samples possible to be returned to the records. The real number returned is specified by the driver in the NORD field.

Returned data can be either integer or floating. Integer data is the raw ADC values and floating data is converted to voltage values (-10V ~ +10V in the case of 8401).

Continuous mode can be stop/start by the ENABLE command at any time.

Configuration example (in the start up script)

```
############################################################
#int Hy8401ipAsynInit(char *portName, int vmeSlotNum, int ipSlotNum, int
#     vectorNum, int samples, int average, int offset, int scanMode, int
#     clockRate, int extClock, int fastADC)
# ARGS: vmeslot=3, ipslot=1, vector=10, samples=10000, average=50, offset=0,
#       scanmode=0(continuous), clockrate=9(1kHz), extclock=0(no), fastADC=0(
#       mca & EPID support)

Hy8401ipAsynInit("myasynport1", IPAC3, 1, 10, 10000, 50, 0, 0, 9, 0, 0)
```

Constrains

- When “samples” exceeds 64k, 64k is used and overflow flag is set.
- Both “average” and “samples” cannot exceed 64K. Normally “average” should not exceed “samples” setting.

For ai records:

- When “average” exceeds 64k, 64k is used and averageOverflow flag is set.
- When “average” exceeds “samples”, “samples” number of data acquired is used for averaging. The averageOverflow flag is set.

For waveform records:

- If nelements is greater than 64k, 64k is used.
- NORD is always set to whatever nelements. This means that at the very beginning when ADC starts, some of the data in the waveform array is whatever the memory left initially. But after certain time (especially when ADC finishes a full cycle), all data is true ADC readings.

Record examples

Polled ai:

Integer

```
record(ai, "$(P):CHANNEL0:IN") {
        field(SCAN, "1 second")
        field(DTYP, "asynInt32")
        field(INP, "@asyn($(PORT) 0) DATA")
}
```

Float

```
record(ai, "$(P):CHANNEL_FLOAT0:IN") {
        field(SCAN, "1 second")
        field(DTYP, "asynFloat64")
        field(INP, "@asyn($(PORT) 0) DATA")
}
```

Polled waveform:

Integer

```
record(waveform,"$(P):WAVEFORMFLOAT1:IN") {
        field(SCAN, "1 second")
        field(DTYP,"asynInt32ArrayIn")
        field(INP,"@asyn($(PORT) 1) DATA")
        field(NELM,"35")
        field(FTVL," LONG")
}
```

Float

```
record(waveform,"$(P):WAVEFORMFLOAT1:IN") {
        field(SCAN, "1 second")
        field(DTYP,"asynFloat64ArrayIn")
        field(INP,"@asyn($(PORT) 1) DATA")
        field(NELM,"35")
        field(FTVL,"DOUBLE")
}
```

I/O Intr ai:

Integer

```
record(ai, "$(P):CHANNEL1:IN") {
        field(SCAN, "I/O Intr")
```

```
		field(DTYP, "asynInt32")
		field(INP, "@asyn($(PORT) 1) DATA")
	}
```

Float

```
	record(ai, "$(P):CHANNELFLOAT2:IN") {
		field(SCAN, "I/O Intr")
		field(DTYP, "asynFloat64")
		field(INP, "@asyn($(PORT) 2) DATA")
	}
```

<u>I/O Intr waveform</u>:

Integer

```
	record(waveform,"$(P):INTRWAVEFORM4:IN") {
		field(SCAN, "I/O Intr")
		field(DTYP,"asynInt32ArrayIn")
		field(INP,"@asyn($(PORT) 4) DATA")
		field(NELM,"25")
		field(FTVL,"LONG")
	}
```

Float

```
	record(waveform,"$(P):INTRWAVEFORMFLOAT5:IN") {
		field(SCAN, "I/O Intr")
		field(DTYP,"asynFloat64ArrayIn")
		field(INP,"@asyn($(PORT) 5) DATA")
		field(NELM,"40")
		field(FTVL,"DOUBLE")
	}
```

Operation sequence

- After initialisation or when changing from other modes to continuous mode, the ADC starts acquisition immediately.
- After “samples” number of readings, the driver triggers the asyn notification for callback records (with SCAN field set to I/O Intr).
- When “enable” is cleared, the ADC stops. It will resume when “enable” is set again.

4.2. Trigger mode

Description

In trigger mode, the ADC doesn’t acquire data until it is enabled and either a software trigger or a hardware trigger has been issued. And once triggered, the ADC only collects “samples” + “offset” number of readings then stops.

There are three important configuration parameters associated to this mode: “average”, “offset” and “samples”. Also there are two flags: overflow and averageOverflow that reflect the validation of the data collected.

“average” is used for ai records in both polled and callback cases. It is the number setting that the driver uses to average the acquired ADC values then returned to the records. At any time, anything goes abnormal (please see below constrains description), the averageOverflow flag is set. Flagged averageOverflow purely means something goes wrong.

“samples” is used for waveform records which specifies the maximum samples possible to be returned to the records. The real number returned is specified by the driver in the NORD field.

“offset” is used to offset the start point for data returned as refer to the trigger point.

Returned data can be either integer or floating. Integer data is the raw ADC value and floating data is converted voltage value (-10V ~ +10V in the case of 8401).

Configuration example (in the start up script)

```
############################################################
#int Hy8401ipAsynInit(char *portName, int vmeSlotNum, int ipSlotNum, int
#     vectorNum, int samples, int average, int offset, int scanMode, int
#     clockRate, int extClock, int fastADC)
# ARGS: vmeslot=3, ipslot=1, vector=10, samples=10000, average=50, offset=0,
#       scanmode=1(trigger), clockrate=15(100kHz), extclock=0(no), fastADC=1(
#       normal ADC)

Hy8401ipAsynInit("myasynport1", IPAC3, 1, 10, 10000, 50, 0, 1, 15, 0, 1)
```

Constrains

- “offset” + “samples” cannot exceed 64K.
- “offset” cannot exceed “samples”.
- Whenever the setting of either the “samples” or the “offset” is set, a check is carried out. If “samples” + “offset” exceeds 64k, (64k – “offset”) is used for “samples” and overflow flag is set.
- “average” cannot exceed “samples” + “offset”. If it does, averageOverflow flag is set and “average” is set to “samples” + “offset”.
- Hytec IP8401 asynDriver doesn’t support negative “offset” at the moment.

At runtime for polled ai records:

Assuming at a certain point after the trigger fires, the ADC has collected N number of readings,

- Whenever N is less than “offset”, averageOverflow flag is set. Also if N is less than “average”, all data collected as far is used to do the averaging. If N is greater than “average”, the most recent “average” number of readings is used.
- When N is greater than “offset” and (N - “offset”) is greater than or equals to “average”, the most recent “average” number of readings is used for averaging and averageOverflow

flag is cleared. If not, the ADC values starting from "offset" to N is used for averaging and the averageOverflow flag is set.

At runtime for polled waveform records:

Assuming at a certain point after the trigger fires, the ADC has collected N number of readings,

- If N is less than or equals to "offset", all data collected as far is returned. NORD is set to N.
- If N is greater than "offset", N – "offset" number of readings is returned and NORD is set to N – "offset".

For callback ai records:

- When "average" exceeds "samples", "samples" number of data acquired is used for averaging. The averageOverflow flag is set.

Record examples

Same as continuous mode. Please refer to previous section.

Operation sequence

- After configuration the ADC is set up and waiting for "enable" command and triggers.
- When "enable" command is set, the ADC is armed but not acquiring. It waits for a trigger to kick off.
- Once a trigger is fired either by software command or hardware, the ADC starts acquisition.
- After "samples" + "offset" number of values collected, the ADC stops. The driver fires the asyn notification for subscribed callback records (with SCAN field set to I/O Intr).
- If "re-enable" is set, the ADC is still armed. Further trigger will start the acquisition again. If "re-enable" is not set, the ADC is disarmed. For further trigger needs to set "enable" command again.
- To disable the ADC from any triggering, set "enable" to 0.

Notes

- Trigger will not start the ADC until it is enabled.
- Before the first trigger, the data returned for any records has no meaning (random). After a trigger period finishes, all records maintain the same values until next trigger.

4.3. Gated mode

Description

In gated mode, the ADC doesn't acquire data until it is enabled and the hardware inhibit is de-asserted. And once start the ADC only stops when it is inhibited or disabled.

There are two important configuration parameters associated to this mode: "average" and "offset".

"average" is used for ai records in both polled and callback cases. It is the number of readings that the driver uses for averaging calculation.

"offset" is used to determine the start point for data returned as refer to the point where the inhibit is de-asserted. If "offset" is positive, the collecting point starts from the "offset" and till the end of the gated period. If "offset" is negative, the collection starts from the gated end point and take "offset" number of data backwards.

Returned data can be either integer or floating. Integer data is the raw ADC value and floating data is converted to voltage value (-10V ~ +10V in the case of 8401).

Configuration example (in the start up script)

```
############################################################
#int Hy8401ipAsynInit(char *portName, int vmeSlotNum, int ipSlotNum, int
#     vectorNum, int samples, int average, int offset, int scanMode, int
#     clockRate, int extClock, int fastADC)
# ARGS: vmeslot=3, ipslot=1, vector=10, samples=10000, average=50, offset=0,
#       scanmode=2(gated), clockrate=13(20kHz), extclock=0(no), fastADC=0(
#       MCA and EPID)

Hy8401ipAsynInit("myasynport1", IPAC3, 1, 10, 10000, 50, 0, 2, 13, 0, 0)
```

Constrains

- For gated mode, the ADC starts acquisition from the very beginning but it doesn't transfer data to the memory hence doesn't serve any EPICS records. At start, the user needs to make sure that the hardware inhibit signal is asserted. It would otherwise populate the records once it is enabled.
- "offset" must be less than 64K.
- If "average" shouldn't exceed 64k.

At runtime for both polled and callback ai records:

Assuming at a certain point after the gate opens the ADC has collected "N" number of readings.

- In the case of a positive "offset"
    - If N is less than "offset", averageOverflow flag is set. Also if N is less than "average", all data collected as far is used to do the averaging. If N is greater than "average", the most recent "average" number of readings is used.
    - If N is greater than "offset" and (N - "offset") is greater than or equals to "average", the most recent "average" number of readings is used for averaging

and averageOverflow flag is cleared. If not, the ADC values starting from “offset” to N is used for averaging and the averageOverflow flag is set.

- In the case of a negative “offset”
    - If N is greater than absolute “offset” value, if “average” is also greater than absolute “offset” value, most recent “offset” readings will be used for averaging and the averageOverflow flag is set. If “average” is less than absolute “offset” value, most recent “average” number of samples will be used and the averageOverflow flag is cleared.
    - If N is less than absolute “offset” value, averageOverflow flag is set. If N is less than “average”, all data collected as far is used to do the averaging. If N is greater than “average”, the most recent “average” number of readings is used.

For both callback and polled waveform records:

Assuming at a point or at the end of the gate the ADC has collected “N” number of readings.

- In the case of a positive “offset”
    - If N is greater than “offset”, (N – “offset”) samples are returned starting from “offset”.
    - Otherwise all data collected is returned.
- In the case of a negative “offset”
    - If N is greater than absolute “offset” value, “offset” number of samples are returned starting from N + “offset” (note here “offset” is negative).
    - Otherwise all data collected is returned.

Record examples

Same as continuous mode. Please refer to previous section.

Operation sequence

- When using the gated mode, the user needs to be aware that at the very beginning the inhibit input on the front panel of 8002 should be logic low (0). This results in the inhibit bit asserted and prevents the ADC values from writing to the memory or populating to the records.
- The configuration routine sets up the ADC to continuous acquiring (with EX bit set to 0) but not writes to the memory until the ADC is enabled.
- When “enable” command is set, the software inhibit is de-asserted but the ADC is not writing to the memory until the hardware inhibit is de-asserted.
- Once the hardware inhibit is de-asserted (the front panel inhibit signal goes high to logic 1), the acquired data is written to the memory and records get populated until the hardware inhibit is asserted again or the ADC is disabled.
- When and only when the hardware inhibit signal is asserted again, the writing to the memory stops. The driver fires the asyn notification for records subscribed (with SCAN field set to I/O Intr).
- If “re-enable” is not set, further hardware inhibit de-assert will not start the acquisition writing to the memory unless a new “enable” command is issued. If “re-enable” is set, any hardware inhibit de-assertion will start the acquisition writing to the memory again.

- To disable the ADC from writing data to the memory, set “enable” to 0.

Notes

- Hardware inhibit de-assertion will not start the ADC writing to the memory unless it is enabled.
- Before the first hardware gate open, the data reading for polled records has no meaning (random). After the first gate open, all readings stay the same until next hardware gate opens.

5. Online Setting Changes

Some of the settings can be altered at runtime. This is to suit the needs of online re-configuration without issuing a system reboot. Obviously some of the runtime changes will software reset the ADC which means the current acquisition will be stopped and restarted.

Whenever a runtime change occurs, an asyn callback is generated to notify whoever monitors these changes. This notification is done by the asynInt32 interface. Please see late explanation.

In addition, when changing the ADC scan clock rate, another callback is generated by asynFloat64 interface to return the scan interval in second for MCA and EPID records.

5.1. Set Clock Rate

Description

This is the simplest online setting change. It simply changes the ADC clock rate without stopping and re-starting the ADC. The only thing will be affected is the speed of acquisition that becomes either faster or slower which will subsequently affect the timing of completion for continuous and trigger mode or more/less data for gated mode.

Implementation

The function is implemented by asynInt32 interface.

Record example

```
record(longout, "$(P):CLOCKRATE:OUT") {
        field(DTYP, "asynInt32")
        field(OUT, "@asyn($(PORT) 0) SETCLOCKRATE")
}
```

5.2. Set Internal/External Clock – will stop/start ADC

Description

This will change the clock source either from internal or external. Issuing this command online will stop/start the ADC. After the re-start, the ADC will be either enabled or disabled according the enable state before the change.

Implementation

The function is implemented by both asynInt32 and asynUInt32Digital interfaces.

Record example

asynInt32 record example

```
	record(longout, "$(P):EXTCLOCK:OUT") {
		field(DTYP, "asynInt32")
		field(OUT, "@asyn($(PORT) 0) SETEXTCLOCK")
	}
```

asynUInt32Digital record example

```
	record(longout, "$(P):UINTEXTCLOCK:OUT") {
		field(DTYP, "asynUInt32Digital")
		field(OUT, "@asynMask($(PORT) 0, 0x1, 1.0) SETEXTCLOCK ")
	}
```

5.3. Set Mode – will stop/start ADC

Description

This command changes the operation mode at runtime. This will cause a major change of the ADC operation hence it will stop the ADC and reset all other settings such as interrupt mechanism etc.

After the change, the ADC goes back to its initial state as per the mode plus it has to be enabled and/or re-enabled to be able to use it.

Implementation

The function is implemented by asynInt32 interface.

Record example

```
	record(longout, "$(P):MODE:OUT") {
		field(DTYP, "asynInt32")
		field(OUT, "@asyn($(PORT) 0) SETMODE")
	}
```

Note, the mode value can be:

0 – continuous mode
1 – trigger mode
2 – gated mode.

5.4. Set Samples – will stop/start ADC

Description

Setting samples at runtime affects continuous and trigger mode only. It has nothing to do with gated mode. Changing this will stop/start the ADC. It would also trigger an overflow check against the maximum bank size (64k for 1M byte memory).

After the re-start, the ADC will be either enabled or disabled according the enable state before the change.

Implementation

The function is implemented by asynInt32 interface.

Record example

```
record(longout, "$(P):SAMPLES:OUT") {
        field(DTYP, "asynInt32")
        field(OUT, "@asyn($(PORT) 0) SETSAMPLE")
}
```

5.5. Set Offset – will stop/start ADC

Description

Offset changes at runtime only apply to trigger and gated mode. It has nothing to do with continuous mode. Changing this will stop/start the ADC. It would also trigger an overflow check against the maximum bank size.

After the re-start, the ADC will be either enabled or disabled according the enable state before the change.

Implementation

The function is implemented by asynInt32 interface.

Record example

```
record(longout, "$(P):OFFSET:OUT") {
        field(DTYP, "asynInt32")
        field(OUT, "@asyn($(PORT) 0) SETOFFSET")
}
```

5.6. Set Average

Description

Changing average affects ai records only but for all modes. This will not stop/start the ADC. It will not trigger an average overflow check since this check is done every time when the ai data is calculated.

Implementation

The function is implemented by asynInt32 interface.

Record example

```
record(longout, "$(P):AVERAGE:OUT") {
        field(DTYP, "asynInt32")
        field(OUT, "@asyn($(PORT) 0) SETAVERAGE")
}
```

5.7. Asyn Callback for Setting Changes

Description

As mentioned before, whenever there is a setting change online, the asynInt32 interface will generate a callback to notify the user the change. These settings include clock rate, internal/external clock select, mode, samples, offset and average etc.

This callback returns the reason enumeration value that is associated to the setting. The definition of the enumeration is:

```
typedef enum{
  /* Write commands */
        hy8401SweepData = 0,
        hy8401Data,
        hy8401ScanPeriod,
        hy8401SetAverage,          -- 3
        hy8401SetSample,           -- 4
        hy8401SetMode,             -- 5
        hy8401SetOffset,           -- 6
        hy8401EnableState,
        hy8401SoftTrigger,
        hy8401ReEnableState,
        hy8401ClearBuffer,
        hy8401SetClockRate,        -- 11
        hy8401SetExtClock,         -- 12

/* Read commands */
        hy8401AverageOverflow,
        hy8401Overflow,
        hy8401BufferCount,
        hy8401GateTriggerState,
        hy8401Support,
        hy8401_MAX_COMMANDS,                /* must always be last! */
} hy8401_cmd_enum;
```

Note, only highlighted items will be returned.

Implementation

The function is implemented by asynInt32 interface.

Record example

```
record(longin, "$(P):CHANGEMONITOR:IN") {
        field(SCAN, "I/O Intr")
        field(DTYP, "asynInt32")
        field(INP, "@asyn($(PORT) 0) SETSAMPLE")
}
```

Note, the monitoring reason ("SETSAMPLE" as in the example above) can be any of the reasons mentioned in the description. Regardless which one is specified here, the driver returns the real command enumeration value.

When changing ADC clock rate, it will generate another callback for asynFloat64 records when its reason is set to SCANPERIOD. This returns the scanning interval in second. This callback is for MCA and EPID records.

6. Online Command

Few commands have been implemented which allow the user to control the ADC operation stop/start or software trigger for example. These commands are implemented by both asynInt32 and asynUInt32Digital interfaces. Both functions exactly the same. Actually the asynUInt32Digital interface calls asynInt32 to do the real job. It is designed for backwards compatibility.

6.1. Enable/Disable ADC

Description

This command as it says enables or disables the ADC in all cases. It is the same command for all modes. The ADC will be only in operation (either continuous or can be triggered or can be gated) when it is enabled. For trigger and gated mode, this command works only once unless the re-enable state is set.

Implementation

The function is implemented by both asynInt32 and asynUInt32Digital interfaces.

Record example

By asynInt32 interface

```
record(longout, "$(P):ENABLE:OUT") {
        field(DTYP, "asynInt32")
        field(OUT, "@asyn($(PORT) 0) ENABLE")
}
```

By asynUInt32Digital interface

```
record(bo, "$(P):UINTENABLE:OUT") {
        field(DTYP, "asynUInt32Digital")
        field(OUT, "@asynMask($(PORT) 0, 0x1, 1.0) ENABLE")
}
```

6.2. Re-enable for Trigger or Gated Mode

Description

This command sets the re-enables state of the ADC operation. It only applies to trigger and gated mode. Once it is set, further trigger (for trigger mode) or inhibit de-asserted (for gated mode) will initialise another data acquisition. If it is not set, the further trigger or de-assertion will only work when there is “enable” command being issued before that.

Implementation

The function is implemented by both asynInt32 and asynUInt32Digital interfaces.

Record example

By asynInt32 interface

```
record(longout, "$(P):REENABLE:OUT") {
        field(DTYP, "asynInt32")
        field(OUT, "@asyn($(PORT) 0) REENABLE")
}
```

By asynUInt32Digital interface

```
record(longout, "$(P):REENABLE:OUT") {
        field(DTYP, "asynUInt32Digital")
        field(OUT, "@asynMask($(PORT) 0, 0x1, 1.0) REENABLE")
}
```

6.3. Software Trigger ADC

Description

Software trigger only applies to trigger mode. It is the software trigger to start the ADC as compare to the hardware trigger though it does exactly the same thing afterwards.

Note, software (same as hardware trigger) will not trigger the ADC unless the ADC is enabled.

Implementation

The function is implemented by both asynInt32 and asynUInt32Digital interfaces.

Record example

By asynInt32 interface

```
record(longout, "$(P):SOFTTRIGGER:OUT") {
        field(DTYP, "asynInt32")
        field(OUT, "@asyn($(PORT) 0) SOFTTRIGGER")
}
```

By asynUInt32Digital interface

```
record(longout, "$(P):UINTSOFTTRIG:OUT") {
        field(DTYP, "asynUInt32Digital")
        field(OUT, "@asynMask($(PORT) 0, 0x1, 1.0) SOFTTRIGGER")
}
```

6.4. Clear Buffer

This command is not implemented in Hytec 8401 asynDriver.

7. Online Status Querying

ADC operation status can be queried during runtime. This includes the overflow, average overflow, gated/trigger mode state and driver support etc.

7.1. Query Overflow State

Description

Overflow flag is checked and set whenever one of the following actions is taken.

- “samples” + “offset” exceeds memory bank size for trigger mode when setting “samples” either at runtime or during the init.
- “samples” + “offset” exceeds memory bank size for trigger mode when setting “offset” either at runtime or during the init.
- “samples” exceeds memory bank size for continuous mode either during runtime or init.

Note, gated mode doesn’t have this check since there is no way of knowing either the gated period is too long (exceeds 64k) at the moment. It could be improved though by the upper conversion memory has been filled interrupt. This needs more logic and can be discussed later.

Implementation

The function is implemented by both asynInt32 and asynUInt32Digital interfaces.

Record example

By asynInt32 interface

```
record(longin, "$(P):OVERFLOW:IN") {
        field(SCAN, "1 second")
        field(DTYP, "asynInt32")
        field(INP, "@asyn($(PORT) 0) OVERFLOW")
}
```

By asynUInt32Digital interface

```
record(longin, "$(P):UINTOVERFLOW:IN") {
        field(SCAN, "1 second")
        field(DTYP, "asynUInt32Digital")
        field(INP, "@asynMask($(PORT) 0, 0x1, 1.0) OVERFLOW")
}
```

7.2. Query Average Overflow State

Description

Average overflow flag only applies for ai records. It is checked and set whenever one of the following actions is taken.

- During set up time, for continuous mode if the average setting is greater than 64k.
- During set up time, for trigger mode if the average setting is greater than “samples” + “offset”
- During set up time, for gated mode if the average setting is greater than 64k
- During runtime (for both polled and callback records), for gated mode if the “offset” is positive and the average is greater than the number of samples collected minus “offset”
- During runtime (for both polled and callback records), for gated mode if the “offset” is negative and the average is greater than the number of samples collected.

Implementation

The function is implemented by both asynInt32 and asynUInt32Digital interfaces.

Record example

By asynInt32 interface

```
record(longin, "$(P):AVERAGEOVERFLOW:IN") {
        field(SCAN, "1 second")
        field(DTYP, "asynInt32")
        field(INP, "@asyn($(PORT) 0) AVERAGEOVERFLOW")
}
```

By asynUInt32Digital interface

```
record(longin, "$(P):UINTAVERAGEOVERFLOW:IN") {
        field(SCAN, "1 second")
        field(DTYP, "asynUInt32Digital")
        field(INP, "@asynMask($(PORT) 0, 0x1, 1.0) AVERAGEOVERFLOW")
}
```

7.3. Query Gated/Trigger State

Description

For gated and trigger mode, the ADC could be in one of the three operation states: waiting for trigger or inhibit asserted, triggered or inhibit de-asserted and collecting data and, collecting finished and data is ready. The user can use a polled record to read this state out from the driver.

Implementation

The function is implemented by both asynInt32 and asynUInt32Digital interfaces.

Record example

By asynInt32 interface

```
record(ai, "$(P):TGSTATE:IN") {
        field(SCAN, "1 second")
        field(DTYP, "asynInt32")
        field(INP, "@asyn($(PORT) 0) GATETRIGGERSTATE")
}
```

By asynUInt32Digital interface

```
record(mbbi, "$(P):UINTTGSTATE:IN") {
        field(SCAN, "1 second")
        field(DTYP, "asynUInt32Digital")
        field(INP, "@asynMask($(PORT) 0, 0x3, 1.0) GATETRIGGERSTATE")
}
```

7.4. Query Driver Support

Description

Driver support returns a bit mask showing what functions are supported by AsynDriver. These bits are:

- Bit 0 – gate mode support
- Bit 1 – trigger mode support
- Bit 2 – trigger mode negative offsets support

In Hytec IP8401 case, only the first two are supported at the moment hence a value of 0x03 will be returned whenever there is a request.

Implementation

The function is implemented by both asynInt32 and asynUInt32Digital interfaces.

Record example

By asynInt32 interface

```
record(longin, "$(P):SUPPORT:IN") {
        field(SCAN, "1 second")
        field(DTYP, "asynInt32")
        field(INP, "@asyn($(PORT) 0) SUPPORT")
```

```
	}
```

By asynUInt32Digital interface

```
	record(longin, "$(P):UINTSUPPORT:IN") {
		field(SCAN, "1 second")
		field(DTYP, "asynUInt32Digital")
		field(INP, "@asynMask($(PORT) 0, 0x7, 1.0) SUPPORT")
	}
```

8. MCA and EPID Record Support

The Hytec IP8401 asynDriver also supports MultiChannel Analyser (MCA) and EPID, the fast feedback control records.

This asynDriver provides 8 channel simultaneously ADC values after each conversion via asynInt32Array interface with the callback notification to the FastSweep driver which is used by MCA records when the asynUser reason is SWEEPDATA (=0 in the enumeration).

The data returned is 8 channel raw values with calibration but not averaged.

The driver also provides a callback which returns the scan period in second whenever the scanRate is changed. This is implemented by the asynFloat64 interface with asynUser reason of SCANPERIOD.

For EPID support, after each conversion, any of the 8 channels can return its calibrated raw data in floating data type to the EPID record. Again, this value is not averaged. This is implemented by asynFloat64 interface in callback notification.

9. Report Interface

The report interface returns information about the driver and the hardware it supports.

If the “details” argument is less than 1, it return basic information such as port name, VME slot number, IP slot number and register base address etc.

When the “details” argument is greater than 1, it returns the above basic information plus statistics of both successful and failed callbacks and clients’ information whoever uses asynInt32, asynInt32Array, asynFloat64 or asynFloat64Array interfaces etc.