multiDaq Commandset

minimum required Firmware release: Version 1.0.5
Date: 2024/04/04

Table of content

C Interface

we provide a h File, the dll itself and a library for linking. The dll is compiled with the Mingw64 Compiler Suite. if You use MSVC or other compilers, the linking acrobatic is up to You.

The Interface is a pure C Interface, so no name mangling problems are expected.

Prerequisites

Win 10, 64 bit Version, or
Linux, 64 bit Version (tested on Ubuntu 22.04)

and a

64 bit C Compiler

---

python Interface

You will find complete sources of interface and examples on github
It is located in the folder /share/python

Prerequisites

Win 10, 64 bit Version, or
Linux, 64 bit Version (tested on Ubuntu 22.04)

and a

python Interpreter, Version 3 with numpy lib installed

matlab/octave Interface

You have acces to the multiDaq devices via a single mex file. The mexfile accesses our biovisionMultiDaq.dll, so this dll must be in your path. No further driver must be installed.

The driver will be started by the init command and stopped by the deinit command. The driver acts as a highly priorized task and will live, until you leave matlab or call the deinit function. After the call of the init command functions to access the device can be called.

The command set of the device is SCPI based. Detailed information is found in chapter SCPI commands

format: biovision_multidaq(command, . . .)

additional parameter depends on command.

commands:

- **'init'**:
  initializes the driver
  one optional parameter, if set to 'debug' the driver will generate additional debug messages on stdout.
- **'deinit'**:
  deinitializes the driver
  no parameter.
- **'listdevices'**
  list all connected biovision devices
  no parameter, returns cell array of device identifiers with needed information of connected devices. The results are intended for use in the open command.
- **'open'** opens a device
  first parameter is the channel number as a string from '1' .. '4', second parameter is the device identifier.
- **'close'**
  close a device
  one parameter, must be the same as the first parameter in the open command
- **'cmd'**
  transmits a SCPI command to the device
  first parameter: channel number second parameter: SCPI command third optional parameter: if is 'stream', you can fire commands in streaming mode. returns char array or uint8 array (if the answer was a SCPI binary block).
- **'get'**
  get available data in streaming mode
  first parameter: channel number returns double array with captured data.
- **'txqueue'**
  central control of txqueues of all devices one parameter, must be '0' or '1'. enables or disables the transmission of all connected devices. purpose: synchronisation of more than one device.

- **'getticks'**
  get a timestamp no parameter, returns an int64 timestamp with 10 MHz clock.

- **'gettimestamps'**
  get timestamps from a synchronized group one parameter, the channel number. get the monitored timestamps of a synchronized start of several devices. returns array of 4 int64 timestamps. first timestamp is taken when txqueue is set to '1'. second timestamp, when driver transmits the command. third timestamp, when driver receives first response. forth timestamp, when driver received first valid data.

SCPI interface

SCPI = **S**tandard **C**ommands for **P**rogrammable **I**nstruments

We follow this Specification: https://www.ivifoundation.org/docs/SCPI-99.PDF

scpi binary blocks

SCPI binary block format is defined in SCPI specs

block format: "#nN",[binary data],"" * n is one character, length of N * N is string of n characters defining the length of the block * both n and N are decimal numbers

format of a SCPI command

All SCPI commands without question mark responses only, if an error occurrs.

format: [:]token1:token2:token [param1] , [param2] ....

example: *idn?[LF] responses with the identification string

possible tokens of example command "CONFigure:SCAn:RATe " does the same:

- conf:sca:rat 1000
- configure:sca:rat 1000
- conf:scan:rate 1000

You have to use the words in uppercase letters or in the whole word. If a parameter is a string, it must be enclosed with quotation marks.

example: conf:trig:mode "pulse",15

available commands:

- **"CONFigure:IMU:PARAms <nIMU>,<rangeAcc>,<rangeGyro>"**
  configures the IMU parameter.
- **"CONFigure:IMU:RANGESAcc?"**
  returns list of possible ranges of the ACC.
- **"CONFigure:IMU:RANGESGyro?"**
  returns list of possible ranges of the Gyroscope.
- **"CONFigure:SCAn:NUMchans?"**
  returns numbers of connected devices as a list. nADC24,nADC16,nIMU (or) nAdc24,nAdc16,nIMU,nAUX (if an aux device is connected).
- **"CONFigure:SCAn:RATe?"**
  returns configured samplerate.

- **"CONFigure:SCAn:RATe <rate>"**
  configures Samplerate, valid range is 100 ... 1000.

- **"CONFigure:SCAn:OVErsampling <value>"**
  configures the Oversampling Rate of ADC16.
  valid values: 1 or 2.

- **"CONFigure:SCAn:OVErsampling?"**
  returns the configured Oversampling Rate of ADC16.

- **"CONFigure:SCAn:LISt <chan1>,<chan2>,..."**
  configures the List of channels. default is 0,1,2,3,4,5,6,7
  and will be restored, if you close the device
  purpose: set the order of channels
  in contrast to the old biodaq devices, this command will not set the number of channels

- **"CONFigure:SCAn:LISt?"**
  returns the complete order of channels as a list.
  You will see 8 entries.

- **"CONFigure:AUX:BAUD <value>"**
  configures the Baudrate of AUX Device (default: 1 Mbps).

- **"CONFigure:AUX:BAUD?"**
  returns configured Baudrate of the aux device.

- **"CONFigure:AUX:DISable"**
  disable AUX device, no parameter
  purpose: reconfiguration of AUX device.

- **"CONFigure:AUX:ENAble"**
  enable AUX Device. no parameter

- **"CONFigure:AUX:NAME?"**
  returns the name of the AUX device, if present. if empty, no AUX device was detected.

- **"CONFigure:AUX:CONFIGanswer?"**
  returns the configured configanswer as hexadecimal string.

- **"CONFigure:AUX:PARam <hexadecimalstring>"**
  configures the configuration bytes of the AUX device.

- **"CONFigure:AUX:PARam?"**
  returns configuration bytes of the AUX device as hexadeimal String.

- **"CONFigure:TRIGger:MODE <mode_string> [,<additional parameters>]"**
  configures Trigger
  valid mode_strings: "pulse","level","schmitt"
  no additional parameter is allowed for mode "level".
  one optional parameter for mode = "pulse": positive number of the pulselen, default: 10
  three additional parameter are required for mode = "schmitt": <chan>,<thresholdUp>,

<thresholdDwn>.
mode "schmitt" monitors logical channel <chan> and sets the Triggeroutput automaticly.
it sets Triggeroutput to 1 if value(chan)> > <ThreshouldUp>
and resets to 0 if <value(chan)> < <thessholdDwn>
if <ThreshouldUp> < <ThreshouldDwn>:
it sets Triggeroutput to 1 if value(chan)> < <ThreshouldUp>
and resets to 0 if <value(chan)> > <thessholdDwn>

- **"CONFigure:TRIGger:MODE?"**
  returns Trigger mode parameter.

- **"TRIGger:SET <value>"**
  sets the triggeroutput. value: 0 or 1
  in mode "schmitt": this command will be ignored.
  in mode "pulse": output is set to <value> and <!value> automaticly after n clocks of samplerate.
  in mode "level": output is set to 0 if <value>==0, else set to 1.

- **"MEMory:RAW? <membank>,<length>"**
  the allowed value of mandatory parameter is 0 .. 20
  the optional parameter set the desired length (max 4096)
  if not used, you get the whole flash block with 4096 bytes
  returns content of selected memory bank as uint8 array.

- **"MEMory:RAW <membank>"**
  the allowed value of mandatory parameter is 0 .. 20 prepares to write a binary block to selected memory bank. maximum length of the following binaryBlock is 4096 bytes this command erases the whole block of flash memory and writes the following SCPI binblock to the flash memory

- **"CONFigure:DEVice <nAdc32>,<nAdc16>,<nIMU>,<nAux>"**
  nAux is optional and should only be used, if an aux device is present.
  configures the device with the current configuration values
  this command has a longer duration, you have to wait.
  You should wait at minimum 30 ms at samplerate = 1000 or 300 ms at samplerate = 100.
  Alternativly you can test the status with 'conf:dev:stat?'

- **"CONFigure:DEVice?"**
  returns numbers of configured devices

- **"CONFigure:DEVice:STATus?"**
  returns status of configuration.
  new configuration of devices set status to 0
  status is set to 1, when configuration is complete

- **"INITiate"**
  start the streaming mode. response are continuously sent binary blocks.
  You can stop this only with the abort command.

- **"ABORt"**
  aborts the streaming mode