“Neuropix”

Phase IIIA System User API

Version: July 28, 2016

Number:

Issued by: Imec

Title Neuropix Phase IIIA System User API

Author(s) Jan Putzeys

Reviewed by Silke Musa

Revision V1.12

Date July 28, 2016

Keywords: Neural probe, neural devices, active electrodes

Revision history

Version	Date	Description	Responsible
V1	May 27, 2014	Initial release, preliminary	Jan Putzeys, Jin Lin
V1.1	June 27, 2015	Extended function list	Jan Putzeys, Tao Lee
V1.2	December 09, 2015	Update on a new API	Jan Putzeys
V1.3	January 20, 2016	2.1: neuropix_open extended 2.1: neuropix_initialize removed. 2.2: Version number explained 3: Electrode selectivity: zero-based table 5.1: Changed parameters for neuropix_setgain 5.2: Changed parameter for neuropix_setstdb. Bool values definition 5.3: Changed parameter for neuropix_setfilter 6.3:Recording functions description updated 9.1: TE: extended description 9.2: NRST updated description 9.3: PR_NRST: updated description 10: Error codes added	Jan Putzeys
V1.4	January 21, 2016	Definition of technical terms updated 3: Changed parameters for neuropix_selectelectrode	Jan Putzeys
V1.5	February 16, 2016	Added chapter: using the MSVC dll.	Jan Putzeys
V1.6	March 23, 2016	Updates on function names, function arguments, errorcodes, added functions	Jan Putzeys

V1.7	April 20, 2016	Extra parameter added to open function	Jan Putzeys
V1.8	April 25, 2016	Added function for reading/applying ADC calibration and gain correction (chapter 2.4, 2.5)	Jan Putzeys
V1.9	April 27, 2016	Added functions for reading ADC calibration and gain correction from csv file and applying to ASIC and BS (New chapter 3.0). Added functions for reading/writing probetypes (chapter 2.3) Added missing 'write_to_asic' parameter for some functions. Added enumerator readcsverrorcode (Chapter 11) Added API version (Chapter 1)	Jan Putzeys
V1.10	May 13, 2016	Added API log function and LED enable function (Chapter 2.4 and 2.6)	Jan Putzeys
V1.11	May 24, 2016	Bug removed in API version number (Chapter 1.1)	Jan Putzeys
V1.12	July 28, 2016	API version updated	Jan Putzeys

Related documents:

- 2016-01-20_Electrode_mapping.xlsx
- 2016-07-28_Neuropix_PhaseIIIA_System_and_Probe_User_Manual_V1_9.docx

Contact address data: Kapeldreef 75, 3001 Heverlee

Contents

List of abbreviations

AP	Action Potential
LFP	Local Field Potential
ASIC	Application Specific Integrated Circuit
HS	Headstage
BSC	Base Station Connect (board)
SR	Shift Register
FPGA	Field Programmable Gate Array
FPGA dev	FPGA development (board)
PCB	Printed Circuit Board
BIST	Built-In Self-Test
API	Application Programming Interface
ADC	Analog-to-Digital Converter
IC	Integrated Circuit
EEPROM	Electrically Erasable Programmable Read-Only Memory
SDRAM	Synchronous Dynamic Random Access Memory
LED	Light Emitting Diode
GPIO	General Purpose Input/Output
SDK	Software Development Kit

List of symbols

Definition of Technical Terms

- **Probe Shank:** Implanted part of the probe ASIC.
- **Probe Base:** Non-implanted part of the probe ASIC.
- **Probe Flex:** Flexible carrier PCB on which the probe ASIC is assembled

- **Electrode bank:** also called ‘array’: group of electrodes for which random selection or all channels are recorded.
- **Headstage (HS)**: Miniature board located close to the ASIC, which configures and calibrates the ASIC and serializes ASIC data.
- **Basestation connect (BSC) board:** Small board with deserializer chip, which connects to the FPGA development board.
- **FPGA development board**: Commercial KC705 FPGA development board (http://www.xilinx.com/products/boards-and-kits/ek-k7-kc705-g.html)

1 Introduction

1.1 Scope

This document provides an overview of the API commands implemented to operate the Neuropix Phase IIIA recording system and is intended to facilitate the development of a custom application to operate the system. Documentation on how to use the hardware is provided in the system user manual.

This version of the document is applicable to API version 4.2 and version 4.3.

2 System initialization

2.1 Open

The neuropix_open function creates a data and configuration link over the Ethernet port with the recording system, and initializes the hardware. Also the compatibility of hardware and software driver is verified. This function needs to have executed successfully before other operations can be made.

The function is also used when reading data from the binary .npx files after recording. Such an .npx file is created when the neural data is recorded to disc during an acquisition session. In this case, the neuropix_open function is called using the path to the .npx file as argument.

API Function:
OpenErrorCode Neuropix_basestation_api::neuropix_open (unsigned char headstage_select = 0)
This function establishes a data connection and a config link connection with the FPGA dev board. It checks the compatibility of hardware and software version, and closes the connection if they are not compatible. The function initializes the headstage and FPGA dev board and resets the probe base and shank configuration with the default values.
Parameters
headstage_select: parameter for internal use, default set to 0.
Returns
OPEN_SUCCESS if successful

API Function:

OpenErrorCode Neuropix_basestation_api::neuropix_open (const std::string & playbackfile)
This function establishes a playback data connection and a dummy config link connection.
Parameters
playbackfile : the path of the binary .npx file.
Returns
OPEN_SUCCESS if successful

2.2 Hardware/software versions

The following functions return the hardware and driver versions. This is only informative; the compatibility of hardware and software is verified with the neuropix_open function.

2.2.1 *FPGA dev board bootcode*

API Function:
ErrorCode Neuropix basestation api::neuropix_getHardwareVersion (Version_number * version)
This function returns the FPGA dev board bootcode version number.
Parameters
version : the version number to return
Returns
FAILURE if no config link connection, SUCCESS otherwise

2.2.2 *Basestation connect board*

API Function:
ConfigAccessErrorCode Neuropix_basestation_api::neuropix_getBSVersion (unsigned char & version major)
This function gets the version number of the basestation connect board.
Parameters
version_major : the BSC board version number
Returns
CONFIG_ERROR_NO_LINK if no datalink, CONFIG_SUCCESS if successful

API Function:
ConfigAccessErrorCode Neuropix_basestation_api::neuropix_getBSRevision (unsigned char & version minor)

This function gets the revision number of the basestation connect board.
Parameters
version_minor : the BaseStation Connect board revision number
Returns
CONFIG_ERROR_NO_LINK if no datalink, CONFIG_SUCCESS if successful

2.2.3 *API*

API Function:
struct Version_number Neuropix_basestation_api::neuropix_getAPIVersion ( )
This function returns the API version number.
Returns
The version number of the API

2.2.4 *Struct Version_number*

The functions for reading the version of FPGA dev board boot code and API use the variable struct Version_number to return the result. The struct has the following members:

- unsigned short major: updated in case a change is made, which affects both hardware and software. Should be equal for FPGA dev board and API to be compatible.
- unsigned short minor: updated in case smaller changes are made, which do not affect compatibility with other system components.

2.3 Probe ID

The function below returns the probe ID number, which is stored on the EEPROM located on the flex. The probe ID number contains a unique serial number and the probe option type.

API Function:
EepromErrorCode Neuropix_basestation_api::neuropix_readId (AsicID &id)
This function reads the probe ID from the EEPROM, copies it to the member and returns it.
Parameters
id : the probe ID to return
Returns
EEPROM_SUCCESS if successful

The struct AsicID has the following members:

- unsigned int serialNumber: a 30-bit serial number.
- char probeType: a 2-bit probe option number:
    - 0: Option 1
    - 1: Option 2
    - 2: Option 3
    - 3: Option 4

In case the EEPROM is not available, the user needs to manually set the probe type, in order to have correct functionality of the API functions. The following function writes the probe ID to the API member. It will also write to the EEPROM, therefore do not use this function with probes that have a functional EEPROM.

The serialNumber used with this function can random, this is not important for the operation of the API.

API Function:
EepromErrorCode Neuropix_basestation_api::neuropix_writeId (AsicID &id)
This function writes the probe ID to the API member and to the EEPROM.
Parameters
id : the probe ID to write
Returns
EEPROM_SUCCESS if successful

The following function can be used to read only the probe type from the API member:

API Function:
unsigned char Neuropix_basestation_api::neuropix_getOption ( )
This function returns the current option (probe type of the ASIC ID) of the member of the API.
Parameters
none
Returns
The option

2.4 Close

The close function closes the data and configuration link over the Ethernet port.

API Function:
void Neuropix_basestation_api::neuropix_close ( )
This function closes the data and config link connection with the device.

2.5 Log file

The system can log commands an status information to a text file (api_log.txt). This functionality is default disabled, after calling the neuropix_open function. The following function enables the logging.

API Function:
void Neuropix_basestation_api::neuropix_startLog ( )
start logging information in api_log.txt

2.6 Headstage LEDs

The 3 headstage LEDs for BIST1, BIST3, BIST5 and start trigger can be disabled. The LEDs are default on after calling the neuropix_open function.

API Function:
void Neuropix_basestation_api::neuropix_ledOff (bool led_off)
This function enables or disables the headstage LEDs.
Parameters
led_off : the desired value for led_off
Returns
DIGCTRL_SUCCESS if successful

3 Probe calibration

3.1 ADC calibration

This functions reads the ADC calibration parameters stored on the EEPROM and applies these to the probe.

API Function:
ErrorCode Neuropix_basestation_api::neuropix_applyAdcCalibrationFromEeprom ( )
Reads ADC calibration from EEPROM, puts it in the calibration member, then writes it to the base register.

Parameters
none
Returns
SUCCESS if successful

In case the EEPROM on the probe is not available, it is also possible to apply the calibration parameters to the probe after extracting them form a csv file, using the following set of functions. Refer to the system user manual for guidance on how to use these functions.

The following function reads the ADC comparator calibration parameters from the specified csv file and writes these to the API member.

API Function:
ReadCsvErrorCode Neuropix_basestation_api::neuropix_readComparatorCalibrationFromCsv (std::string filename = "comparatorCalValues.csv")
This function reads the comparator calibration values (compP and compN) from the given csv file.
Parameters
filename : the filename to read from (should be .csv), default is comparatorCalValues.csv
Returns
READCSV_SUCCESS if successful

The following function reads the ADC offset calibration parameters from the specified csv file and writes these to the API member.

API Function:
ReadCsvErrorCode Neuropix_basestation_api::neuropix_readADCOffsetCalibrationFromCsv (std::string filename = "adcOffsetCalValues.csv")
This function reads the ADC offset calibration values (cfix) from the given csv file.
Parameters
filename : the filename to read from (should be .csv), default is adcOffsetCalValues.csv
Returns
READCSV_SUCCESS if successful

The following function reads the ADC slope calibration parameters from the specified csv file and writes these to the API member.

API Function:

ReadCsvErrorCode Neuropix_basestation_api::neuropix_readADCSlopeCalibrationFromCsv (std::string filename = "adcSlopeCalValues.csv")
This function reads the ADC slope calibration values (slope, coarse and fine) from the given csv file.
Parameters
filename : the filename to read from (should be .csv), default is adcSlopeCalValues.csv
Returns
READCSV_SUCCESS if successful

The following function reads the ADC comparator calibration parameters from the API member.

API Function:
ErrorCode Neuropix_basestation_api::neuropix_getADCCompCalibration (std::vector< struct adcComp > & adcCompCalibration)
This function returns the adcCompCalibration_ member of the api.
Parameters
adcCompCalibration : current value, vector of 32
Returns
SUCCESS if successful, FAILURE if not available

The following function reads the ADC slope and offset calibration parameters from the API member.

API Function:
ErrorCode Neuropix_basestation_api::neuropix_getADCPairCommonCalibration (std::vector< struct adcPairCommon > & adcPairCommonCalibration)
This function returns the adcPairCommonCalibration_ member of the api.
Parameters
adcPairCommonCalibration : current value, vector of 16
Returns
SUCCESS if successful, FAILURE if not available

The following function writes the ADC calibration parameters (comparator, offset and slope) of one selected ADC to the base configuration register on the probe.

API Function:
BaseConfigErrorCode Neuropix_basestation_api::neuropix_ADCCalibration (unsigned char adcPairIndex, unsigned char compPxx, unsigned char compNxx, unsigned char

compPyy, unsigned char compNyy, unsigned char slope, unsigned char fine, unsigned char coarse, unsigned char cfix)
This function sets the calibration bits of a selected ADC pair, and writes the resulting base configuration to the base configuration shift register.
Parameters
adcPairIndex : the adc pair index (valid range: 0 to 15, pair index 0 is adc 1 and 3, pair index 1 is adc 2 and 4, etc compPxx : the compP value to write (valid range: 0 to 31) compNxx : the compN value to write (valid range: 0 to 31) compPyy : the compP value to write (valid range: 0 to 31) compNyy : the compN value to write (valid range: 0 to 31) slope : the slope value to write (valid range: 0 to 7) fine : the fine value to write (valid range: 0 to 3) coarse : the coarse value to write (valid range: 0 to 3) cfix : the cfix value to write (valid range: 0 to 15)
Returns
BASECONFIG_SUCCESS if successful

3.2 Gain correction

This functions reads the gain correction parameters stored on the EEPROM and writes these to the FPGA dev kit.

API Function:
ErrorCode Neuropix_basestation_api::neuropix_applyGainCalibrationFromEeprom ( )
Reads gain calibration from EEPROM, put it in the calibration member, then writes it to the BS FPGA.
Parameters
none
Returns
SUCCESS if successful

In case the EEPROM on the probe is not available, it is also possible to apply the gain correction parameters to the probe after extracting them from a csv file, using the following set of functions. Refer to the system user manual for guidance on how to use these functions.

The following function reads the gain correction parameters from the specified csv file and writes these to the API member.

API Function:
ReadCsvErrorCode Neuropix_basestation_api::neuropix_readGainCalibrationFromCsv (std::string filename = "gainCalValues.csv")
This function reads the gain calibration values (gain correction factors) from the given csv file.
Parameters
filename : the filename to read from (should be .csv), default is gainCalValues.csv
Returns
READCSV_SUCCESS if successful

The following function returns the gain correction parameters from the API member.

API Function:
ErrorCode Neuropix_basestation_api::neuropix_getGainCorrectionCalibration (std::vector< unsigned short > & gainCorrectionCalibration)
This function returns the gainCorrectionCalibration_ member of the api.
Parameters
gainCorrectionCalibration : current value, vector of unsigned short of size 384 for Option 1 and 2, 960 for Option 3, 966 for Option 4
Returns
SUCCESS if successful, FAILURE if not available

The following function writes the gain correction parameters to the basestation register.

API Function:
ConfigAccessErrorCode Neuropix_basestation_api::neuropix_gainCorrection (std::vector< unsigned short > & gaincorr)
This function writes the given gain correction factors to the registermap of the basestation fpga.
Parameters
gaincorr : the gain correction factors to write
Returns
CONFIG_SUCCESS if successful

4 Shank settings - Electrode connectivity

Shank programmability is only provided for Option 3 and 4 probes. For Option 3 probes, the shank contains 960 electrodes, while 384 channels are available on the base. For Option 4 probes, the shank contains 966 electrodes, and the base contains 276 channels. Each channel can connect to only a limited number of electrodes and only to

one electrode at the same time. The mapping between electrodes and channels is different between Option 3 and Option 4 probes.

A complete table of channels mapping to electrodes is provided in document '2016-01-20_Electrode_mapping.xlsx'. The following table shows an extract of the complete mapping:

Table 1: Electrode to channel mapping for Option 3.

	Bank number 0	**Bank number 1**	**Bank number 2**
Channel	**Electrode**	**Electrode**	**Electrode**
0	1	385	769
1	2	386	770
…	…	…	…
383	384	768	

The following figures show the grouping of electrodes into banks:

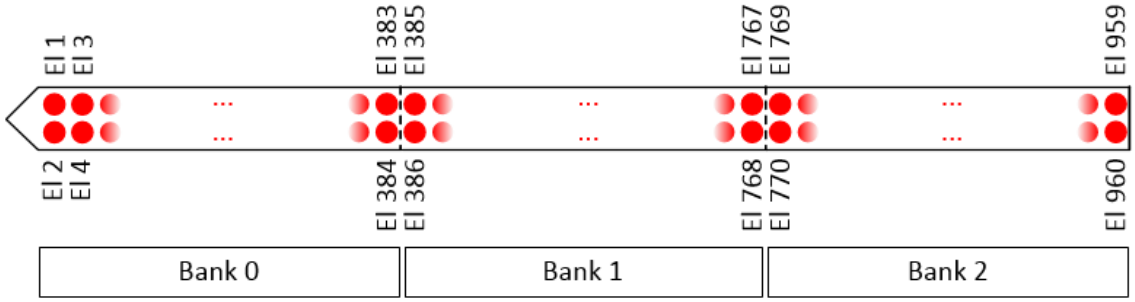

Figure 1: Electrode bank distribution over the shank for Option 3 probes.

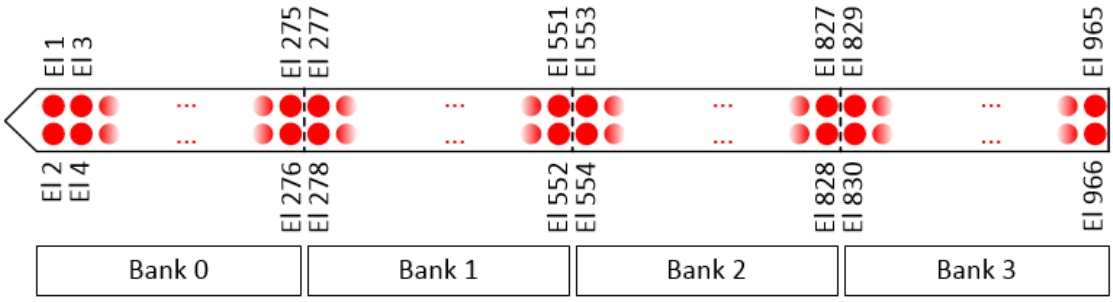

Figure 2: Electrode bank distribution over the shank for Option 4 probes.

The function below sets which electrode is connected to a channel. By using this function, the user can only connect one electrode at a time to a channel. Furthermore, the function checks the probe type, and returns an error in case the user tries to make an illegal electrode to channel connection.

API Function:
ShankConfigErrorCode Neuropix_basestation_api::neuropix_selectElectrode (int channel, int electrode_bank, bool write_to_asic = true)
This function sets the given channel to the given electrode connection, and optionally writes the resulting shank configuration chain to the headstage probe.
Parameters
channel : the channel number to set (valid range: 0 to 383 for option 3, 0 to 275 for option 4)
electrode_bank : the electrode bank number to connect to (valid range option 3: 0 to 2 or 0xFF for electrode < 960, valid range option 4: 0 to 3 or 0xFF for electrode < 966, or use the enum ElectrodeConnection)
bool write_to_asic : if false, the reference selection is written to the shank configuration member of the API but not transmitted to the probe via the UART. If true, the updated shank configuration member content is transmitted to the probe. Default set to true.
Returns
SHANK_SUCCESS if successful

From the channel number and electrode bank number it can be calculated which electrode is connected to the channel:

For Option 3: Electrode number = (channel number+1) + 384 * electrode bank number
For Option 4: Electrode number = (channel number+1) + 276 * electrode bank number

For example: Channel number = 9 and electrode bank number connection = 1, option 4 probe. Channel 9 is connected to electrode 394. The other possible electrodes (#10, #778) are disconnected by the function algorithm.

If the parameter Electrode bank is set to 0xFF, the channel is not connected to any electrode. This setting is useful for internal reference channels.

Table 2: enum ElectrodeConnection

Mode value	Enum ElectrodeConnection	Bank
0	ZERO	0
1	ONE	1
2	TWO	2
3	THREE	3
255	NONE	Channel disconnected from any electrode

5 Reference selection

5.1 Reference selection on the probe

The user has the option to select the reference input for each channel on the probe. This reference selection differs for the different probe options.

For Option 1, 2 and 3 probes, for each of the 384 channels on the base, 1 of 11 possible reference inputs can be selected. The first reference input connects to the external reference input pin on the probe. The other 10 reference input lines connect to electrodes on the shank (internal references).

For Option 4 probes, for each of the 276 channels on the base, 1 of 8 possible reference input lines can be selected. The first reference input connects to the external reference input pin on the probe. The other 7 reference input lines connect to electrodes on the shank (internal references).

Because of the shank programmability for Option 3 and Option 4 probes, the internal reference lines can connect to several reference electrodes along the shank. Therefore, which internal reference electrode is used as a reference input for the channel is the combination of the function for electrode selection and reference line selection.

5.2 API implementation

The user takes the following steps to program the reference selection:

1. Select which electrodes in the shank go to the reference lines (shank programmability), via the function *neuropix_selectElectrode*.

2. Select which of the 11 reference lines is connected to every channel (base programmability) via the function *neuropix_setReference* or *neuropix_writeAllReferences.*

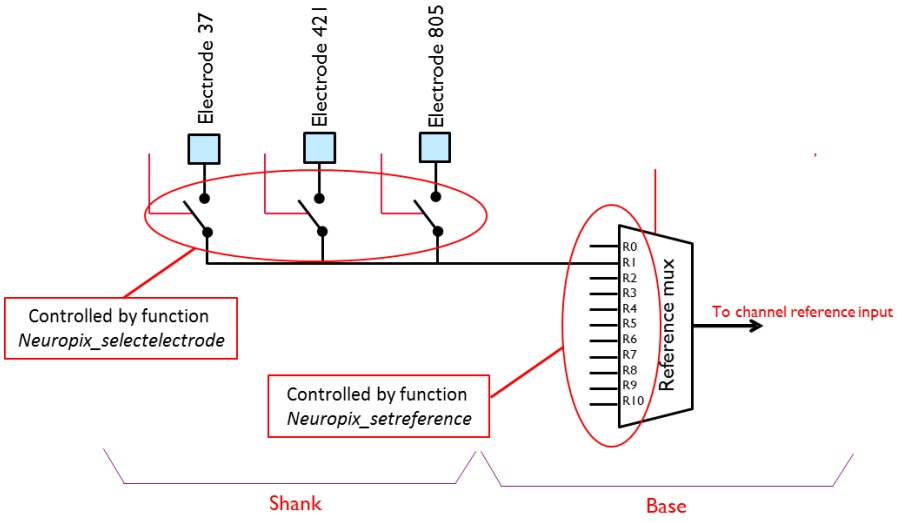

Figure 3: Internal reference selection via functions neuropix_selectElectrode and neuropix_setReference for Option 3 probes.

For Option 3 probes, the following table shows which electrode can connect to an internal reference line. As explained earlier, this selection is made via the function *neuropix_selectElectrode*.

Table 3: Electrode mapping to internal reference lines for Option 3.

Ref input	**Electrode**	**Electrode**	**Electrode**
R0 (External)	-	-	-
R1 (Internal 1)	37	421	805
R2 (Internal 2)	76	460	844
R3 (Internal 3)	113	497	881
R4 (Internal 4)	152	536	920
R5 (Internal 5)	189	573	957
R6 (Internal 6)	228	612	
R7 (Internal 7)	265	649	
R8 (Internal 8)	304	688	
R9 (Internal 9)	341	725	
R10 (Internal 10)	380	764	

The following table shows which electrode can connect to an internal reference line for Option 4 probes.

Table 4: Electrode mapping to internal reference lines for Option 4.

Ref input	**Electrode**	**Electrode**	**Electrode**	**Electrode**
R0 (External)	-	-	-	-

R1 (Internal 1)	37	313	589	865
R2 (Internal 2)	76	352	628	904
R3 (Internal 3)	113	389	665	941
R4 (Internal 4)	152	428	704	
R5 (Internal 5)	189	465	741	
R6 (Internal 6)	228	504	780	
R7 (Internal 7)	265	541	817	

The following function is provided to select the reference line input for a selected channel. The function returns an error in case the user tries to write an illegal value for an Option 4 probe (for example select reference R10).

API Function:
BaseConfigErrorCode Neuropix_basestation_api::neuropix_setReference (int channel, int reference, bool write_to_asic = true)
This function selects the given reference input for the given channel by setting the channel reference of the given index to true and the other reference values to false, and optionally writes the resulting base configuration to the base configuration shift register.
Parameters
channel : the number of the channel (valid range: 0 to 383 for options 1-3, 0 to 275 for option 4)
index : the reference index to enable (valid range: 0 to 10 for options 1-3, 0 to 7 for option 4)
bool write_to_asic : if false, the reference selection is written to the base configuration member of the API but not transmitted to the probe via the UART. If true, the updated base configuration member content is transmitted to the probe. Default set to true.
Returns
BASECONFIG_SUCCESS if successful, ILLEGAL_CHANNEL_NUMBER if channel number > 383, ILLEGAL_WRITE_VALUE if index > 10, BASECONFIG_WRITE_ERROR if an error occurs writing the base configuration shift register

The following function is provided to set the reference line input for all channels on the probe to the same value.

API Function:
BaseConfigErrorCode Neuropix_basestation_api::neuropix_writeAllReferences (unsigned char reference)
This function sets the reference value of every channel to the given value and writes the resulting base configuration to the base configuration shift register.
Parameters

refindex : the reference index to enable (valid range: 0-10 for Options 1-3, 0-7 for Option 4)
Returns
BASECONFIG_SUCCESS if successful, ILLEGAL_WRITE_VALUE if index out of range, BASECONFIG_WRITE_ ERROR if error writing base configuration shift register

In case none of the channels on the probe is using the external reference signal as a reference input, it is required to disconnect the probes' external reference signal bondpad from the reference multiplexers inputs. This can be done with the following function.

API Function:
ShankConfigErrorCode Neuropix_basestation_api::neuropix_setExtRef (bool connect, bool write_to_asic = true)
This function sets the connection of the external reference signal bondpad to the R0 input of the reference multiplexers of the channels.
Parameters
connect : true: the external reference signal is connected to R0; false: the external reference signal is disconnected from R0. bool write_to_asic : if false, the reference selection is written to the shank configuration member of the API but not transmitted to the probe via the UART. If true, the updated shank configuration member content is transmitted to the probe. Default set to true.
Returns
SHANK_SUCCESS if successful

6 Channel configuration

6.1 Gain

One function is provided to set the AP and LFP gain of a selected channel. Two other functions allow the user to set the AP and LFP gain of all channels to the same value.

API Function:
BaseConfigErrorCode Neuropix_basestation_api::neuropix_setGain (int channel, int ap_gain, int lfp_gain, bool write_to_asic = true)

This function sets both AP and LFP gain of the given channel to the given values, and optionally writes the resulting base configuration to the base configuration shift register.
Parameters
channel : the channel number (valid range: 0 to 383)
ap_gain : the AP gain value (valid range: 0 to 7)
lfp_gain : the LFP gain value (valid range: 0 to 7)
bool write_to_asic : if false, the reference selection is written to the base configuration member of the API but not transmitted to the probe via the UART. If true, the updated base configuration member content is transmitted to the probe. Default set to true.
Returns
BASECONFIG_SUCCESS if successful, ILLEGAL_CHANNEL_NUMBER if channel number > 383, ILLEGAL_WRITE_VALUE if ap_gain > 7 or lfp_gain > 7, BASECONFIG_WRITE_ERROR if error writing the base configuration shift register

API Function:
BaseConfigErrorCode Neuropix_basestation_api::neuropix_writeAllAPGains (int apgain)
This function sets the AP gain of every channel to the given value, and writes the resulting base configuration to the base configuration shift register.
Parameters
apgain : the AP gain value to write (valid range: 0 to 7)
Returns
BASECONFIG_SUCCESS if successful, ILLEGAL_WRITE_VALUE if apgain > 7, BASECONFIG_WRITE_ERROR if error writing base configuration shift register

API Function:
BaseConfigErrorCode Neuropix_basestation_api::neuropix_writeAllLFPGains (int lfpgain)
This function sets the LFP gain of every channel to the given value, and writes the resulting base configuration to the base configuration shift register
Parameters
lfpgain : the LFP gain value to write (valid range: 0 to 7)
Returns
BASECONFIG_SUCCESS if successful, ILLEGAL_WRITE_VALUE if lfpgain > 7, BASECONFIG_WRITE_ERROR if error writing base configuration shift register

The following table gives an overview of the AP/LFP gain parameters and the corresponding indicative probe gain.

Table 5: AP/LFP gain parameter values and corresponding probe gain setting.

AP gain, LFP gain	Indicative probe gain
0	50
1	125
2	250
3	500
4	1000
5	1500
6	2000
7	2500

6.2 Standby

One function is provided to enable or disable the standby mode of a selected channel. Another function allows the user to set the standby mode of all channels to the same value. Putting a channel in standby only switches off the analog amplifier part of that channel. The channel output is still digitized by the ADC on the probe, and transmitted as part of the neural data.

API Function:
BaseConfigErrorCode Neuropix_basestation_api::neuropix_setStdb (int channel, bool standby, bool write_to_asic)
This function sets the standby mode of a selected channel to the given value, and writes the resulting base configuration to the base configuration shift register.
Parameters
channel : the number of the channel (valid range: 0 to 383)
standby : the standby value to write
bool write_to_asic : if false, the reference selection is written to the base configuration member of the API but not transmitted to the probe via the UART. If true, the updated base configuration member content is transmitted to the probe. Default set to true.
Returns
BASECONFIG_SUCCESS if successful, ILLEGAL_CHANNEL_NUMBER if channel > 383, BASECONFIG_WRITE_ERROR if error writing the base configuration shift register

API Function:

BaseConfigErrorCode Neuropix_basestation_api::neuropix_writeAllStandby (bool standby)
This function sets the standby value of every channel to the given value, and writes the resulting base configuration to the base configuration shift register.
Parameters
standby : the standby value to write
Returns
BASECONFIG_SUCCESS if successful, BASECONFIG_WRITE_ERROR if error writing base configuration shift register

The parameter standby can take the following values:

Table 6: Standby values.

bool standby	Channel status
False	Channel on
True	Channel Off

6.3 Filter

The filter high-pass cut-off frequency of the AP band (common for all AP channels) is set with the following function:

API Function:
BaseConfigErrorCode Neuropix_basestation_api::neuropix_setFilter (int filter)
This function sets the high-pass filter frequency to the given value and writes the resulting base configuration to the base configuration shift register.
Parameters
filter : the bw_hp value to write (valid range: 0 = 300Hz, 1 = 500Hz, 3 = 1kHz)
Returns
BASECONFIG_SUCCESS if successful, ILLEGAL_WRITE_VALUE if bw_hp > 3 or bw_hp == 2

The following table gives an overview of the filter parameters and the corresponding high-pass cut-off frequency:

Table 7: Cut-off frequencies.

int filter	Cut-off frequency
0	300Hz
1	500Hz
2	Not used
3	1kHz

7 Data acquisition/synchronization

7.1 Trigger mode

The readout system has two trigger configurations: input or output. In output mode, the user sends an API command, which generates a trigger pulse and simultaneously starts the neural data acquisition. In input mode, the system starts the neural data acquisition when an external pulse is received. The signal connection used for the trigger is a bidirectional line on the basestation connect board labelled 'EXT_START'.

The following function selects between input or output mode:

API Function:
ConfigAccessErrorCode Neuropix_basestation_api::neuropix_triggerMode (bool drive_sync_ext_start)
This function configures the trigger mode.
Parameters
drive_sync_ext_start: if true, EXT_START will be driven by the FPGA
Returns
CONFIG_ERROR_NO_LINK if no configuration link, CONFIG_SUCCESS if successful

7.2 Start trigger

In case the system is configured in start trigger output mode, the following function generates a start trigger on the basestation connect board SMA connector labelled 'EXT_START', and thereby starts the acquisition of neural data by the probe, which is buffered on the FPGA dev board memory. After the start trigger, the user needs to start transferring the neural data to disk using the recording function.

API Function:
ConfigAccessErrorCode Neuropix_basestation_api::neuropix_setNeuralStart ( )
This function generates the start trigger
Returns
CONFIG_ERROR_NO_LINK if no configuration link, CONFIG_SUCCESS if successful

7.3 Recording

The API provides a function to read out the data from the FPGA dev board over the TCP/IP interface. Optionally, the recorded data can be streamed to a binary file.

7.3.1 *Reading data over the TCP/IP interface*

The function neuropix_readElectrodeData reads data over the TCP/IP interface from the FPGA dev board memory and returns a data packet in the format of the class 'ElectrodePacket'. The function is also used to read data packets from .npx files.

API Function:
virtual ReadErrorCode Neuropix_basestation_api::neuropix_readElectrodeData (ElectrodePacket &result)
This function reads an electrode packet of data.
Parameters
result: the resulting packet of electrode data
Returns
error of ReadErrorCode

7.3.2 *Streaming data to disc*

It is possible to stream all data, which is being acquired from the FPGA dev board memory to a binary file on the PC hard drive. The function neuropix_startRecording defines the file to which all data read via the function neuropix_readElectrodeData is streamed to. The streaming to disk is terminated via the function neuropix_stopRecording.

The data is saved to a binary file with the extension .npx.

API Function:
ErrorCode Neuropix_basestation_api::neuropix_startRecording (const std::string filename = "datalog.npx"**)**
Start recording data. All data read (via neuropix_readElectrodeData) will be recorded to file selected.
Returns
SUCCESS if successful, FAILURE is no data connection

API Function:
ErrorCode Neuropix_basestation_api::neuropix_stopRecording ( )
Stop recording data.
Returns
SUCCESS if successful, FAILURE is no data connection

7.3.3 *Reading data from .npx file*

The data stored in an .npx file can be read with the function neuropix_readElectrodeData, which returns the data in the format of ElectrodePacket.

In order to do so, the user first needs to open a session to the .npx file, using the function neuropix_open.

7.3.4 *ElectrodePacket data format*

The function neuropix_readElectrodeData returns an object of class ElectrodePacket. This contains the following members:

- void printPacket ( ): This function prints an electrode packet.
- float apData[12][384]: 12 samples for each of 384 AP channels.
- unsigned int ctrs[12][13]: The 13 counters of all 12 iterations. (390 KHz probe ADC clock).
- float lfpData[384]: 384 LFP data samples, 1 per channel.
- bool startTrigger[12]: The start trigger status on BSC board.
- unsigned short synchronization[12]: Synch. header (16-bit) board.

The ElectrodePacket contains one LFP sample of each of the 384 channels, and 12 AP samples of each of the 384 channels. This is due to the difference in sampling rate frequency of LFP (2.5 KHz) and AP (30 KHz). By the time the system has sampled each LFP channel once, it has sampled each AP channel 12 times.

The parameter synchronization is the 16-bit synch. word, which is recorded via the BSC board header connector synchronously with the neural data. The signal is sampled at 30 KHz, the same as the AP channels.

Also the Ext. Start signal on the BSC board is recorded like the synchronization (parameter startTrigger).

The counter value (ctrs) is the value of the internal probe ADC counter (390 KHz). It can be monitored for debugging reasons (i.e. data loss) but this is already done on the FPGA dev board.

7.3.5 *Binary file data format*

When the neural data is recorded to a binary .npx file, it is stored in packets in the format explained below. This extra information is not required if the function neuropix_readElectrodeData is used to read the binary file.

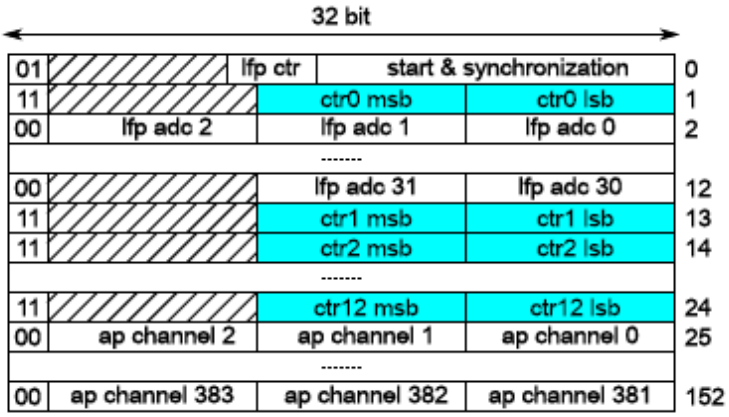

Figure 4: Data format of 1 packet in the binary file.

1 Packet of electrode data contains 153 32-bit words. The following data is included in 1 packet:

1. A 2-bit marker prefix to indicate the kind of data:
    - 00: AP/LFP data
    - 01: Electrode packet header
    - 11: counters
    - 10: reserved.
2. 1 sample for each of the 384 AP channels. Each sample is 10 bits.
3. 1 sample for 32 LFP channels. Thus after 12 packets 1 sample for each of the 384 LFP channels has been received. Each sample is 10 bits.
4. A 4-bit LFP counter. This counter allows to demux the LFP data of 13 packets into 384 LFP channels.
5. 1 16-bit synchronization word (ref. basestation connect board)
6. 1-bit start trigger status.
7. 13 subsequent values of a 20-bit counter. This is the probe ADC counter, running at 390 KHz, which is 13x the AP sampling frequency (30 KHz).
   In the table above, the 20 bits are split in 10 bit MSB and 10 bit LSB. They can, however, be appended to 20 bit.
   The counter value in each new packet continues where the counter in the previous packet ended. For example packet #0 has counter values from 0 to 12. Packet #1 has counter values from 13 to 25.
   The counter value can be monitored to see if packets have been lost during the transmission from basestation to PC.

7.4 SDRAM filling

The FPGA development board contains a SDRAM which buffers the incoming data before it is read out by the PC via the Ethernet port. An API function is available to monitor the percentage of filling of this buffer.

API Function:
float Neuropix_basestation_api::neuropix_fifoFilling( )
This function returns the filling % of the Basestation buffer
Returns
filling level, in %

7.5 Reset Basestation datapath

In case the SDRAM buffer is full, the datapath on the FPGA development board needs to be reset. This can be done using a API function, but must be done together with a probe reset:

Step 1: set probe to reset mode: neuropix_nrst(false)
Step 2: reset SDRAM: neuropix_resetDatapath()
Step 3: set probe out of reset mode: neuropix_nrst(true)

A better way is to use the start trigger function, into which the datapath reset procedure is integrated.

API Function:
ErrorCode Neuropix_basestation_api::neuropix_resetDatapath( )
This function resets the datapath of the basestation fpga: • first the data_generator, decoupling fifos, deserializer_intf, scale, reorder, dma "functional cores" are put in reset (their corresponding configuration registers are not reset) • next the DRAM fifo is flushed • then the reset from the first step is released
Returns
SUCCESS if successful

8 Impedance

The following functions start an impedance measurement sequence on the probe, and return the measured impedance values (only for Option 1 probes).

API Function:
CalibErrorCode Neuropix_basestation_api::neuropix_impedanceMeasurement (std::string fileName = "results_impedanceMeasurement.csv")
This function calculates the impedance for each electrode.
Parameters
fileName : the file name to save the results to, should be .csv, default file name is results_impedanceMeasurement.csv
Returns
CALIB_SUCCESS if successful

API Function:
ErrorCode Neuropix_basestation_api::neuropix_getImpedanceMeasurement (std::vector< int > &impedanceMeasurement)
This function returns the impedanceMeasurement_ member of the api.
Parameters
impedanceMeasurement: current value, vector of int of size 384 for Options 1 and 2, 960 for Option 3, 966 for Option 4
Returns
SUCCESS if successful, FAILURE if not available

9 BIST

The following functions are provided to verify the correct functionality of different components of the read-out system. Further information can be found in the system user manual.

9.1 BIST #4

API Function:
BistTest4ErrorCode Neuropix_basestation_api::neuropix_test4 ( )
run BIST test 4 : test leds, test LCD, test DRAM (selftest), test TCP data link (loopback)
Returns
BISTTEST4_SUCCESS if successful, BISTTEST4_NO_DEVICE if no device opened, BISTTEST4_LED_ERROR if register read/write failure, BISTTEST4_DRAM_ERROR if DRAM selftest failure, BISTTEST4_LOOPBACK_ ERROR if data link loopback failure

9.2 BIST #5

API Function:
BistTestErrorCode Neuropix_basestation_api::neuropix_test5 (unsigned char & hs brd ver, unsigned char & hs fpga ver, unsigned int time sec)
run BIST test 5 : read HeadStage FPGA version + SW_HBEAT blink test @ 2Hz
Parameters
hs_brd_ver : HeadStage Board Version result hs_fpga_ver : HeadStage FPGA Version result time_sec : length of SW_HBEAT blink test, in seconds
Returns
BISTTEST_SUCCESS if successful, BISTTEST_NO_DEVICE if no device opened, BISTTEST_UART_ERROR if UART communication error

9.3 BIST #6

API Function:
BistTest6ErrorCode Neuropix_basestation_api::neuropix_startTest6 ( )
start BIST test 6 : Ser/Des PRBS pattern
Returns
BISTTEST6_SUCCESS if started successful, BISTTEST6_NO_DEVICE if no device opened, BISTTEST6_UART_ERROR if UART communication error, BISTTEST6_SER_ERROR if serializer status at address 0x04 does not equal 0x87, BISTTEST6_DESER_ERROR if deserializer status at address 0x04 does not equal 0x87, BISTTEST6_PRBS_ERR if PRBS_ERR is not zero

API Function:
BistTest6ErrorCode Neuropix_basestation_api::neuropix_stopTest6 (unsigned char &prbs_err)
stop BIST test 6 : Ser/Des PRBS pattern
Parameters
prbs_err : error counter result
Returns
BISTTEST6_SUCCESS if successful, BISTTEST6_NO_DEVICE if no device opened, BISTTEST6_UART_ERROR if UART communication error, BISTTEST6_SER_ERROR if serializer status at address 0x04 does not equal 0x87, BISTTEST6_DESER_ERROR if deserializer status at address 0x04 does not equal 0x87, BISTTEST6_PRBS_ERR if PRBS_ERR not zero

9.4 BIST #7

API Function:
BistTestErrorCode Neuropix_basestation_api::neuropix_startTest7 ( )
start BIST test 7 : Headstage Neural data test pattern
Returns
BISTTEST_SUCCESS if started successful, BISTTEST_NO_DEVICE if no device opened, BISTTEST_UART_ERROR if UART communication error

API Function:
BistTestErrorCode Neuropix_basestation_api::neuropix_stopTest7 ( )
start BIST test 7 : Headstage Neural data test pattern
Returns
BISTTEST_SUCCESS if stopped successful, BISTTEST_NO_DEVICE if no device opened, BISTTEST_UART_ERROR if UART communication error

API Function:
unsigned int Neuropix_basestation_api::neuropix_test7GetErrorCounter ( )
BIST test 7 results
Returns
number of errors in data test pattern

API Function:
unsigned char Neuropix_basestation_api::neuropix_test7GetErrorMask ( )
BIST test 7 results
Returns
mask of errors in data test pattern: bit 0 to 3 is respectively SPI line 1 to 4

API Function:
unsigned char Neuropix_basestation_api::neuropix_test7GetTotalChecked ( )
BIST test 7 results
Returns
total number of patterns checked

9.5 BIST #8

API Function:
BistTest8ErrorCode Neuropix_basestation_api::neuropix_startTest8 (bool te, unsigned char spi line)
start BIST test 8a (TE = true) or test 8b (TE = false)
Parameters
te : TE value spi_line : 0 to 3, the SPI line to test.
Returns
BISTTEST8_SUCCESS if started successful, BISTTEST8_NO_DEVICE if no device opened, BISTTEST8_UART_ERROR if UART communication error, BISTTEST8_RANGE_ERR if SPI line out of range, BISTTEST8_DIGCTRL_ERR if digital control access failed

API Function:
BistTest8ErrorCode Neuropix_basestation_api::neuropix_stopTest8 ( )
stop BIST test 8a/b
Returns
BISTTEST8_SUCCESS if stopped successful, BISTTEST8_NO_DEVICE if no device opened, BISTTEST8_UART_ERROR if UART communication error, BISTTEST8_DIGCTRL_ERR if digital control access failed

API Function:
BistTestErrorCode Neuropix_basestation_api::neuropix_test8GetErrorCounter (bool te, bool &spi_adc_err, bool &spi_ctr_err, bool &spi_sync_err)
read result of BIST test 8a (TE = true) or test 8b (TE = false)
Parameters
te: TE value spi_adc_err : to store spi_adc_err bit spi_ctr_err : to store spi_ctr_err bit spi_sync_err : to store spi_sync_err bit
Returns
BISTTEST_SUCCESS if successful, BISTTEST_NO_DEVICE if no device opened, BISTTEST_UART_ERROR if UART communication error

9.6 BIST #9

API Function:
BistTest9ErrorCode Neuropix_basestation_api::neuropix_startTest9 (bool te)
start BIST test 9a (TE = true) or test 9b (TE = false)
Parameters
te : TE value.
Returns
BISTTEST9_SUCCESS if started successful, BISTTEST9_NO_DEVICE if no device opened, BISTTEST9_DIGCTRL_ERR if digital control access failed

API Function:
BistTestErrorCode Neuropix_basestation_api::neuropix_stopTest9 ( )
stop BIST test 9a/b
Returns
BISTTEST_SUCCESS if stopped successful, BISTTEST_NO_DEVICE if no device opened, BISTTEST_UART_ERROR if UART communication error

API Function:
void Neuropix_basestation_api::neuropix_test9GetResults (unsigned int &sync_errors, unsigned int & sync_total, unsigned int &ctr_errors, unsigned int & ctr_total, unsigned int & te_spi0_errors, unsigned int & te_spi1_errors, unsigned int & te_spi2_errors, unsigned int & te_spi3_errors, unsigned int & te_spi_total)
read result of BIST test 9a/b
Parameters
sync_errors : to store the number of sync errors sync_total : to store total number of sync checks (1 check = all 4 syncs are correct in 1 spi packet) ctr_errors : to store the number of counter errors ctr_total : to store the total number of counter 10 lsbs and 10 msbs checks (1 check = all 4 10 lsbs or all 4 10 msbs are correct) te_spi0_errors : to store the number of data errors on spi line 0 te_spi1_errors : to store the number of data errors on spi line 1 te_spi2_errors : to store the number of data errors on spi line 2 te_spi3_errors : to store the number of data errors on spi line 3 te_spi_total : to store the total number of data checked per spi line

10 Miscellaneous

10.1 MODE

Function to control the operation mode of the probe. 2 Modes are available for user operation: recording and impedance.

API Function:
DigitalControlErrorCode Neuropix_basestation_api::neuropix_mode (unsigned char mode)
This function writes the probe MODE field of the Headstage General Control Register #2 of the headstage FPGA.
Parameters
mode : the mode value to write (range 2 to 3) or use the enum AsicMode
Returns
DIGCTRL_SUCCESS if successful

Table 8: enum AsicMode values

Mode value	Enum AsicMode	ASIC Mode
0	ASIC_CONFIGURATION	Internal use
1	ASIC_CALIBRATION	Internal use
2	ASIC_IMPEDANCE	Impedance
3	ASIC_RECORDING	Recording

10.2 TE

Function to enable/disable a test mode on the probe. When TE = true, the neural data is being replaced by a fixed value, which is different between odd and even electrode numbers. The value for odd and even electrode numbers is 0.8V and 0.4V, respectively.

API Function:
DigitalControlErrorCode Neuropix_basestation_api::neuropix_te (unsigned char te)
This function sets the value for the probe TE signal on the HS FPGA.
Parameters
te : the TE value to write (valid range: 0 to 1)
Returns
DIGCTRL_SUCCESS if successful

10.3 NRST

The NRST signal resets amplifiers as well as digital multiplexers and ADC counters on the probe. When NRST = false, no neural data is being transmitted by the probe.

Control of the NRST signal will be integrated in the start trigger procedure, but this low-level access function is provided for debugging purposes.

API Function:
DigitalControlErrorCode Neuropix_basestation_api::neuropix_nrst (bool nrst)
This function sets the value for the probe NRST signal on the HS FPGA.
Parameters
nrst : the desired value for NRST
Returns
DIGCTRL_SUCCESS if successful

10.4 PR_NRST

The PR_NRST signal (active low) resets the analog channels on the probe. It can be used for reducing the settling time of the amplifier, after an NRST pulse, by setting it to false for a number of seconds. The signal must be set to true for regular operation of the probe.

API Function:
DigitalControlErrorCode Neuropix_basestation_api::neuropix_prnrst (bool pr_nrst)
This function sets the value for the probe PR_NRST signal on the HS FPGA.
Parameters
pr_nrst : the desired value for PR_NRST
Returns
DIGCTRL_SUCCESS if successful

11 Errorcode enumerators

OpenErrorCode		
0	OPEN_SUCCESS	device opened successfully
1	DATA_LINK_FAILED	error while opening data link
2	CONFIG_LINK_FAILED	error while opening config link
3	DEVICE_ALREADY_OPEN	device was already open
4	WRONG_HW_SW_VERSION	incompatible hw and sw version
5	CONFIG_DESER_FAILED	error while configuring deserializer

6	CONFIG_SER_FAILED	error while configuring serializer
7	CONFIG_HS_ND_ENABLE_FAILED	error while configuring headstage ND_ENABLE
8	CONFIG_HS_TE_FAILED	error while configuring headstage TE
9	CONFIG_HS_PRNRST_FAILED	error while configuring headstage PR_NRST
10	CONFIG_HS_NRST_FAILED	error while configuring headstage NRST
11	CONFIG_HS_MODE_FAILED	error while configuring headstage MODE
12	CHECK_ASIC_ID_FAILED	error while reading asic ID from EEPROM
13	CONFIG_BASE_REG_FAILED	error while writing base shift register
14	CONFIG_SHANK_REG_FAILED	error while writing shank shift register
15	CONFIG_TEST_REG_FAILED	error while writing test shift register
16	CONFIG_TRIGGER_MODE_FAILED	error while configuring basestation start_trigger_output_enable
17	CONFIG_NEURAL_START_FAILED	error while configuring basestation neural_start
18	CONFIG_SYNC_EXT_START_FAILED	error while configuring basestation start_trigger_output
19	CONFIG_CALIB_FAILED	error while configuring members for calibration
20	CONFIG_EEPROM_FAILED	error while configuring the eeprom
21	CONFIG_HS_REG_FAILED	error while configuring the headstage registers
22	CONFIG_DATAMODE_FAILED	error while setting the datamode

ErrorCode		
0	SUCCESS	successful
1	FAILURE	an error was detected

ConfigAccessErrorCode		
0	READCSV_SUCCESS	Reading the csv file was successful
1	READCSV_FILE_ERR	Error opening the filestream of the csv
2	READCSV_NUMBER_OF_ELEMENTS	Invalid number of elements read
3	READCSV_OUT_OF_RANGE	Read element is out of range

EepromErrorCode		
0	EEPROM_SUCCESS	accessing eeprom successful
1	EEPROM_EN_ERROR	error enabling / disabling the eeprom
2	EEPROM_ACKCLEAR_ERROR	error clearing / reading eeprom error reg
3	EEPROM_DEV_ADDR_ERROR	error writing the eeprom device address
4	EEPROM_MSBADDR_ERROR	error writing the msb of the eeprom address
5	EEPROM_LSBADDR_ERROR	error writing the lsb of the eeprom address
6	EEPROM_WRITE_ERROR	error writing the write byte to eeprom
7	EEPROM_READ_ERROR	error reading the read byte from eeprom
8	EEPROM_CMD_ERROR	error writing the command for eeprom access
9	EEPROM_NO_ACK_ERROR	no ack received
10	EEPROM_ACK_ERROR	an eeprom acknowledge error was detected
11	EEPROM_WRITEVAL_ERROR	write value out of valid range
12	EEPROM_START_SEQ_ERROR	error writing the start seq bit to eeprom

ConfigAccessErrorCode		
0	CONFIG_SUCCESS	config access successful
1	CONFIG_ERROR_NO_LINK	no config link existing
2	CONFIG_WRONG_MODE_ERROR	accessing electrode param while adc mode
3	CONFIG_WRITE_VAL_ERROR	value to write (or its size) out of range

ShankConfigErrorCode		
0	SHANK_SUCCESS	access was successful
1	SHANK_ILLEGAL_OPTION	illegal option for use of shank config
2	SHANK_ILLEGAL_CHANNEL	channel number out of range
3	SHANK_ILLEGAL_CONNECTION	connection number out of range
4	SHANK_ILLEGAL_CHAIN_CONFIG	multiple electrodes connected to channel
5	SHANK_WRITE_ERROR	error writing shank configuration shift register

BaseConfigErrorCode		
0	BASECONFIG_SUCCESS	get or set successfully
1	ILLEGAL_ADC_NUMBER	adc number out of range (valid: 0 to 31)
2	ILLEGAL_CHANNEL_NUMBER	channel number out of range (valid: 0 to 383)
3	ILLEGAL_WRITE_VALUE	value to write out of range
4	BASECONFIG_WRITE_ERROR	error writing base configuration shift register
5	ILLEGAL_CHAN_REF_READ	In case a ChannelRefxxx contains more than 1 ones

ReadErrorCode		
0	READ_SUCCESS	data successfully read
1	NO_DATA_LINK	data link not connected
2	WRONG_DATA_MODE	incorrect data mode configured
3	DATA_BUFFER_EMPTY	data not available in buffer
4	DATA_ERROR	lfp_ctr doesn't match, or header, ..., The most probable cause is DRAM FIFO overflow

CalibErrorCode		
0	CALIB_SUCCESS	calibration procedure successful
1	CALIB_SETMODE_ERROR	error while setting the mode
2	CALIB_SETBASECONFIG_ERROR	error while setting the base configuration
3	CALIB_WRITESHIFT_ERROR r	error while writing shift register
4	CALIB_SETDATAMODE_ERROR	error while setting the data mode
5	CALIB_DACCFG_ERROR	error while setting the dac
6	CALIB_READDATA_ERROR	error while reading data
7	CALIB_RESETDATAPATH_ERROR	error while resetting datapath
8	CALIB_PREVIOUSCALIB_ERROR	results of previous calibrations not available
9	CALIB_READASICID_ERROR	error while reading the ASIC ID
10	CALIB_SETSHANKCONFIG_ERROR	error while setting shank configuration

11	CALIB_CONFIG_BS_ERROR	error while writing configuration values to the basestation

DigitalcontrolErrorCode		
0	DIGCTRL_SUCCESS	digital control access successful
1	DIGCTRL_READVAL_ERROR	uart error while dig ctrl read
2	DIGCTRL_WRITEVAL_ERROR	uart error while dig ctrl write
3	DIGCTRL_WRITE_VAL_OUT_OF_RANGE	value to write is out of range

BistTestErrorCode		
0	BISTTEST_SUCCESS	BIST Test (started / stopped) successful
1	BISTTEST_NO_DEVICE	BIST Test no device opened
2	BISTTEST_UART_ERROR	BIST Test UART communication error

BistTest4ErrorCode		
0	BISTTEST4_SUCCESS	BIST Test 4 successful
1	BISTTEST4_NO_DEVICE	BIST Test 4 no device opened
2	BISTTEST4_LED_ERROR	BIST Test 4 register read/write failure
3	BISTTEST4_DRAM_ERROR	BIST Test 4 DRAM selftest failure
4	BISTTEST4_LOOPBACK_ERROR	BIST Test 4 data link loopback failure

BistTest6ErrorCode		
0	BISTTEST6_SUCCESS	BIST Test 6 (started) successful
1	BISTTEST6_NO_DEVICE	BIST Test 6 no device opened
2	BISTTEST6_UART_ERROR	BIST Test 6 UART communication error
3	BISTTEST6_SER_ERROR	BIST Test 6 serializer status at address 0x04 does not equal 0x87
4	BISTTEST6_DESER_ERROR	BIST Test 6 deserializer status at address 0x04 does not equal 0x87
5	BISTTEST6_PRBS_ERR	BIST Test 6 PRBS_ERR not zero

BistTest8ErrorCode		
0	BISTTEST8_SUCCESS	BIST Test 8 started / stopped successful
1	BISTTEST8_NO_DEVICE	BIST Test 8 no device opened
2	BISTTEST8_UART_ERROR	BIST Test 8 UART communication error
3	BISTTEST8_RANGE_ERR	BIST Test 8 SPI line out of range
4	BISTTEST8_DIGCTRL_ERR	BIST Test 8 digital control access failed

BistTest9ErrorCode

0	BISTTEST9_SUCCESS	BIST Test 9 started / stopped successful
1	BISTTEST9_NO_DEVICE	BIST Test 9 no device opened
2	BISTTEST9_DIGCTRL_ERR	BIST Test 9 digital control access failed

12 MSVC dll

In order to use the MSVC dll in a C++ project created in Microsoft Visual Studio, follow the next steps:

1. Under “Solution Explorer”, right-click “Resource Files”, “Existing Item”. Browse to the folder where the Neuropix API header files are located, and select the following header files:
    - Neuropix_basestation_api.h
    - ConnectionLinkPacket.h
    - AdcPacket.h
    - ElectrodePacket.h
    - BaseConfiguration.h
    - ShankConfiguration.h
    - TestConfiguration.h
    - hs_fpga_register.h
    - dll_import_export.h

2. Refer to the DLL and LIB file under the project’s settings: Under “Solution Explorer”, select and right-click the project. Select “Properties”, “Configuration Properties”, “VC++ Directories”. In the field “Library Directories”, add the directory where the .LIB and .DLL file are located.
3. Under “Solution Explorer”, select and right-click the project. Select “Properties”, “Configuration Properties”, “Linker”, “Input”. In the field “Additional Dependencies”, add the path to the .LIB file.
4. Optional Post-Build Event: Copy the .dll file from the resource library to the output folder: Under “Solution Explorer”, select and right-click the project. Select “Properties”, “Configuration Properties”, “Build Events”, “Post-build Events”. In the field “Command Line”, use the XCOPY command to copy the .dll file to the output folder. For example: XCOPY “C:\reposNeuropix\API\MSVC\V2.7*.dll” “$TargetDir)" /Y