NanoBoard API Specification

Nano River Technologies
April 2012

Table of Contents

ABBREVIATIONS

API	Application Programming Interface
GPIO	General Purpose IO
I2C	Inter-Integrated Circuit
IIC	Inter-Integrated Circuit (same as I2C)
IO	Input / Output
NRT	Nano River Technologies
SPI	Serial Peripheral Interface
USB	Universal Serial Bus

1. Introduction

This document provides the software interface (API) available for the NanoBoard.

The board consists of four interfaces – a general purpose IO (GPIO) interface, an IIC interface, an SPI interface and analogue inputs. Chapter 3 provides a high level summary of all software tasks, grouped by interface. Chapter 4 provides details of low level USB tasks. Chapter 5 describes some further functions for high level configuration. Chapter 6 describes in details all tasks associated with the GPIO interface. Chapter 7 describes tasks associated with the I2C interface. Chapter 8 describes tasks associated with the SPI interface. Chapter 9 has functions for the analogue input interface.

In chapter 10 any pre-defined types and constants declared in the API are summarised.

2. File Summary

To follow are the main files which together make up the NanoBoard API. In order to see how these should be used in a real application, please refer to the "**NanoBoard Application Examples"** document.

nanoboard.h:	This is an include file for the NanoBoard. It contains types and constants that you can refer to in calling the tasks/functions.
nanoboard_class.h:	This is the header file for the NanoBoard Class Library.
nanoboard_class.cpp:	This is the C++ file for the NanoBoard Class Library. The file calls functions in the OpenSource libusb_0.dll.
libusb.h:	This is the header file to the OpenSource libusb_0.dll.

3. Task Overview

High level tasks are provided for each of the three physical interfaces – general purpose I/O (GPIO), IIC and SPI. This section provides a high level summary list of tasks available per interface.

3.1. USB Tasks

- **OpenDevice** — This function is simply used to open a connection to the NanoBoard.

3.2. Configuration Tasks

- **Nano_Revision** — Returns the revision register for the NanoBoard firmware.
- **Nano_PinConfiguration** — This function allows configuration of GPIO pins as digital IO, analogue inputs or SPI chip selects.

3.3. GPIO Tasks

- **Nano_GPIOSetDirection** — This function can be used to set the I/O direction for all GPIOs.
- **Nano_GPIOGetDirection** — This function returns the configured I/O direction for all GPIOs.
- **Nano_GPIOWrite** — This function sets the output level for all GPIOs.
- **Nano_GPIORead** — This function returns the logic level for all GPIOs.
- **Nano_GPIOSetSingleBitDirection** — This function sets direction for one specified GPIO.
- **Nano_GPIOGetSingleBitDirection** — This function returns direction for one specified GPIO.
- **Nano_GPIOSingleBitWrite** — This function sets the output level for one specified GPIO.
- **Nano_GPIOSingleBitRead** — This function returns the logic level for one specified GPIO.

3.4. I2C Tasks

- **Nano_I2CSetFrequency** — This function sets the rate for the I2C clock.
- **Nano_I2CScanConnectedDevices** — This function returns all I2C devices connected to the I2C bus.
- **Nano_I2CWrite** — This function performs an I2C write operation.
- **Nano_I2CRead** — This function performs an I2C read operation.

3.5. SPI Tasks

- **Nano_SPIMasterConfig** This function is used to configure the SPI line rate and the SPI transfer mode.
- **Nano_SPIReadWrite** This function performs a combined SPI read and write.
- **Nano_SPIRead** This function performs only a SPI read operation.
- **Nano_SPIWrite** This function performs only a SPI write operation.

3.6. Analogue Input Tasks

- **Nano_ADCRead** This function can be used to read an analogue input.

4. USB Tasks

4.1. OpenDevice

bool **OpenDevice ( )**

This function tries to open a connection to the NanoBoard. If the connection is successful then it returns TRUE, otherwise it will return FALSE. The task should be called at the start of applications before any other NanoBoard functions are called.

5. Configuration Tasks

5.1. Nano_Revision

```
WORD            Nano_Revision ( )
```

This function returns the revision for the NanoBoard firmware.

5.2. Nano_PinConfiguration

```
NANO_RESULT     Nano_PinConfiguration (
                        HANDLE  hDevice,
                        BYTE    IO_00,
                        BYTE    IO_01,
                        BYTE    IO_02,
                        BYTE    IO_03,
                        BYTE    IO_04,
                        BYTE    IO_05,
                        BYTE    IO_06,
                        BYTE    IO_07,
                        BYTE    IO_08,
                        BYTE    IO_09,
                        BYTE    IO_10,
                        BYTE    IO_11,
                        BYTE    IO_12,
                        BYTE    IO_13,
                        BYTE    IO_14,
                        BYTE    IO_15,
                        BYTE    IO_16,
                        BYTE    IO_17,
                        BYTE    IO_18,
                        BYTE    IO_19,
                        BYTE    IO_20,
                        BYTE    IO_21,
                        BYTE    IO_22,
                        BYTE    IO_23
                );
```

This function allows setting of the pin functionality for each GPIO pin according to the following table.

hDevice: This is the handle to the USB device.

IO_xx: This is the pin mode for each individual GPIO. GPIO_00 to GPIO_03 can be configured as digital IO, analogue input, SPI chip select (active low) or SPI chip select (active high).

GPIO_04 to GPIO_23 can be configured as digital IO, SPI chip select (active low) or SPI chip select (active high).

Parameter	Possible Values	Meaning
IO_00	GPIO_PIN ADC_PIN SPI_CS_ACTIVE_LO SPI_CS_ACTIVE_HI	Digital input or output Analogue input SPI Chip select (active low) SPI Chip select (active high)
IO_01	GPIO_PIN ADC_PIN SPI_CS_ACTIVE_LO SPI_CS_ACTIVE_HI	Digital input or output Analogue input SPI Chip select (active low) SPI Chip select (active high)
IO_02	GPIO_PIN ADC_PIN SPI_CS_ACTIVE_LO SPI_CS_ACTIVE_HI	Digital input or output Analogue input SPI Chip select (active low) SPI Chip select (active high)
IO_03	GPIO_PIN ADC_PIN SPI_CS_ACTIVE_LO SPI_CS_ACTIVE_HI	Digital input or output Analogue input SPI Chip select (active low) SPI Chip select (active high)
IO_04	GPIO_PIN SPI_CS_ACTIVE_LO SPI_CS_ACTIVE_HI	Digital input or output SPI Chip select (active low) SPI Chip select (active high)
IO_05	GPIO_PIN SPI_CS_ACTIVE_LO SPI_CS_ACTIVE_HI	Digital input or output SPI Chip select (active low) SPI Chip select (active high)
…	…	…
IO_22	GPIO_PIN SPI_CS_ACTIVE_LO SPI_CS_ACTIVE_HI	Digital input or output SPI Chip select (active low) SPI Chip select (active high)
IO_23	GPIO_PIN SPI_CS_ACTIVE_LO SPI_CS_ACTIVE_HI	Digital input or output SPI Chip select (active low) SPI Chip select (active high)

The function returns NANO_RESULT. NANO_SUCCESS is a success and NANO_XACTION_FAILURE is a failure.

6. GPIO Tasks

In this section the tasks for the GPIO interface are described in full detail.

6.1. Nano_GPIOSetDirection

```
NANO_RESULT     Nano_GPIOSetDirection (
                HANDLE    hDevice,
                ULONG     Value
                )
```

This function sets the direction for all 24 bits of GPIO.

hDevice:	This is the handle to the USB device.
Value:	This is the required direction. One bit is for each IO. Set (=1) means output, clear (=0) means input. Bit 0 corresponds to GPIO_00. Bit 23 corresponds to GPIO_23. ***Note for this command to have any effect on a particular GPIO then the corresponding pin must be configured as a GPIO using Nano_PinConfiguration(). Otherwise the command will not influence the pins behaviour.***

The function returns NANO_RESULT. NANO_SUCCESS is a success and NANO_XACTION_FAILURE is a failure.

6.2. Nano_GPIOGetDirection

```
NANO_RESULT     Nano_GPIOGetDirection (
                HANDLE    hDevice,
                ULONG     *pValue,
                )
```

This function returns the direction value for all 24 bits of GPIO.

hDevice:	This is the handle to the USB device.
**pValue*:	This is a pointer to be filled with the direction value. One bit is for each IO. A set value bit (=1) means an output and clear (=0) means an input.

The function returns NANO_RESULT. NANO_SUCCESS is a success and NANO_XACTION_FAILURE is a failure.

6.3. Nano_GPIOWrite

```
NANO_RESULT     Nano_GPIOWrite (
                HANDLE    hDevice,
                ULONG     Value
                )
```

This function writes to all 24 bits of GPIO.

hDevice:	This is the handle to the USB device.
Value:	This is the required value. One bit for each IO. Set (=1) means the GPIO is to be set. Clear (=0) means the GPIO is to be cleared. Bit 0 corresponds to GPIO_00. Bit 23 corresponds to GPIO_23. ***The GPIO will only be written to if the pin is configured as a GPIO using Nano_PinConfiguration() AND if the direction bit is configured for output.***

The function returns NANO_RESULT. NANO_SUCCESS is a success and NANO_XACTION_FAILURE is a failure.

6.4. Nano_GPIORead

```
NANO_RESULT     Nano_GPIORead (
                HANDLE    hDevice,
                ULONG     *pValue
                )
```

This function returns the value of all 23 bits of GPIO.

hDevice:	This is the handle to the USB device.
**pValue*:	This is a pointer to be filled with the GPIO values. The GPIO is read irrespective of whether the GPIO bit is set for input or output. Set (=1) means the GPIO is high. Clear (=0) means the GPIO is low. Bit 0 corresponds to GPIO_00. Bit 23 corresponds to GPIO_23.

The function returns NANO_RESULT. NANO_SUCCESS is a success and NANO_XACTION_FAILURE is a failure.

6.5. Nano_GPIOSetSingleBitDirection

```
NANO_RESULT     Nano_GPIOSetSingleBitDirection (
                HANDLE      hDevice,
                ULONG       GPIONumber,
                BOOL        bOutput
                )
```

This function sets the direction for one of the GPIO bits.

hDevice:	This is the handle to the USB device.
GPIONumber:	This specifies which GPIO direction bit to update, 0 means GPIO_00 and 23 corresponds to GPIO_23.
bOutput:	This is the required value for the direction bit. TRUE means that the GPIO direction bit should be set for output. FALSE means that the direction bit should be input. ***Note for this command to have any effect on a particular GPIO then the corresponding pin must be configured as a GPIO using Nano_PinConfiguration(). Otherwise the command will not influence the pins behaviour.***

The function returns NANO_RESULT. NANO_SUCCESS is a success and NANO_XACTION_FAILURE is a failure.

6.6. Nano_GPIOGetSingleBitDirection

```
NANO_RESULT     Nano_GPIOGetSingleBitDirection (
                HANDLE      hDevice,
                ULONG       GPIONumber,
                BOOL        *pbOutput
                )
```

This function returns the direction of one of the GPIO bits.

hDevice:	This is the handle to the USB device.
GPIONumber:	This specifies which GPIO direction bit to return, 0 means GPIO_00 and 23 corresponds to GPIO_23.
**pbOutput*:	This is a pointer to be filled with the direction value. TRUE means that the GPIO direction bit is set for output. FALSE means that the direction bit is set as input.

The function returns NANO_RESULT. NANO_SUCCESS is a success and NANO_XACTION_FAILURE is a failure.

6.7. Nano_GPIOSingleBitWrite

```
NANO_RESULT     Nano_GPIOSingleBitWrite (
                HANDLE      hDevice,
                ULONG       GPIONumber,
                BOOL        Value
                )
```

This function sets the value for one of the GPIO bits.

hDevice:	This is the handle to the USB device.
GPIONumber:	This specifies which GPIO to update, 0 means GPIO_00 and 23 corresponds to GPIO_23.
Value:	This is the required value. TRUE means that the GPIO direction bit should be driven high. FALSE means that the direction bit should be driven low. ***The GPIO will only be written to if the pin is configured as a GPIO using Nano_PinConfiguration() AND if the direction bit is configured for output.***

The function returns NANO_RESULT. NANO_SUCCESS is a success and NANO_XACTION_FAILURE is a failure.

6.8. Nano_GPIOSingleBitRead

```
NANO_RESULT     Nano_GPIOSingleBitRead (
                HANDLE      hDevice,
                ULONG       GPIONumber,
                BOOL        *pValue
                )
```

This function returns the value of one of the GPIO bits.

hDevice:	This is the handle to the USB device.
GPIONumber:	This specifies which GPIO value to return, 0 means GPIO_00 and 23 corresponds to GPIO_23.
**pValue*:	This is a pointer to be filled with the value. TRUE means that the GPIO is high. FALSE means that the GPIO is low.

The function returns NANO_RESULT. NANO_SUCCESS is a success and NANO_XACTION_FAILURE is a failure.

7. I2C Tasks

In this section the tasks for the I2C interface are described in full detail.

7.1. Nano_I2CSetFrequency

```
NANO_RESULT     Nano_I2CSetFrequency (
                HANDLE      hDevice,
                BYTE        Frequency
                )
```

This function sets the clock frequency for I2C transfers.

hDevice: This is the handle to the USB device.

Frequency: This is the desired clock frequency for I2C transfers. The following table relates the setting constant to the actual observed clock frequency.

Setting	Clock Frequency
NANO_I2C_FREQ_1MHZ	1Mbit/s
NANO_I2C_FREQ_FAST	400kBit/s (I2C Fast Mode)
NANO_I2C_FREQ_200KHZ	200kBit/s
NANO_I2C_FREQ_STD	100kBit/s (I2C Standard Mode)
NANO_I2C_FREQ_50KHZ	50kBit/s

The function returns NANO_RESULT. NANO_SUCCESS is a success and NANO_XACTION_FAILURE is a failure.

7.2. Nano_I2CScanConnectedDevices

```
NANO_RESULT     Nano_I2CScanConnectedDevices (
                HANDLE      hDevice,
                DEV_LIST    *pList
                )
```

This function scans all devices on the I2C bus and returns a list of their I2C addresses.

hDevice:	This is the handle to the USB device.
**pList*:	This is a status list of all I2C devices responding on the I2C bus.

The function returns NANO_RESULT. NANO_SUCCESS is a success and NANO_XACTION_FAILURE is a failure.

7.3. Nano_I2CWrite

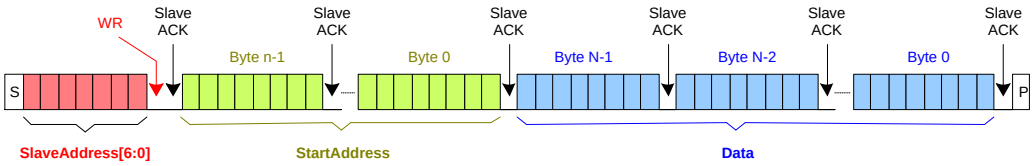
```
NANO_RESULT     Nano_I2CWrite (
                HANDLE      hDevice,
                BYTE        SlaveAddress,
                BYTE        StartAddressLength,
                ULONG       StartAddress,
                WORD        BufferLength,
                BYTE        *pOutBuffer
                )
```

This function provides a complete I2C write command, writing up to 64 bytes of data at a time. The user should configure the slave address, start address, buffer length and fill the data buffer.

hDevice:	This is the handle to the USB device.
SlaveAddress:	This is the I2C slave address (device ID).
StartAddressLength:	This is the width (in bytes) of the start address. The valid range is 0-4 bytes. If 0 is specified then simply no start address is included.
StartAddress:	This is starting address within the device for the transfer.
BufferLength:	Sets the buffer length for the transfer. Maximum is 64 bytes.
**pOutBuffer*:	This is pointer to a buffer which should contain the data to be written.

The function returns NANO_RESULT. The function will return NANO_SUCCESS for a successful write. If the function returns NANO_I2C_PROTOCOL_ERROR then there has been an error in the I2C protocol, for example maybe the slave device has not acknowledged properly. If the function returns a NANO_XACTION_FAILURE, then there is a USB communication failure.

Page Write



S **Start Condition**

P **Stop Condition**

7.4. Nano_I2CRead

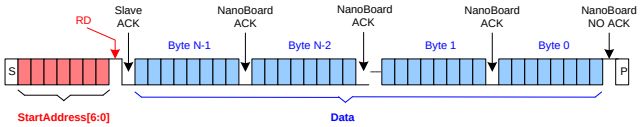
```
NANO_RESULT     Nano_I2CRead (
                HANDLE      hDevice,
                BYTE        SlaveAddress,
                BYTE        StartAddressLength,
                ULONG       StartAddress,
                WORD        BufferLength,
                BYTE        *pInBuffer
                )
```

This function provides a complete I2C read command, reading up to 64 bytes of data at a time. The user should configure the slave address, start address, start address width and buffer length. At the end of the transaction the read data is placed in the data buffer.

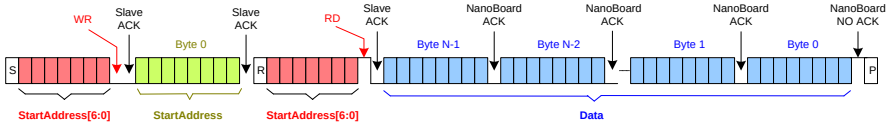
hDevice:	This is the handle to the USB device.
SlaveAddress:	This is the I2C slave address (device ID).
StartAddressLength:	This is the width (in bytes) of the start address. The valid range is 0-4 bytes. If 0 is specified then simply no start address is included.
StartAddress:	This is starting address within the device for the transfer.
BufferLength:	Sets the buffer length for the transfer. Maximum is 64 bytes.
**pInBuffer*:	This is pointer to a buffer which is filled with read data.

The function returns NANO_RESULT. The function will return NANO_SUCCESS for a successful read. If the function returns NANO_I2C_PROTOCOL_ERROR then there has been an error in the I2C protocol, for example maybe the slave device has not acknowledged properly. If the function returns a NANO_XACTION_FAILURE, then there is a USB communication failure.

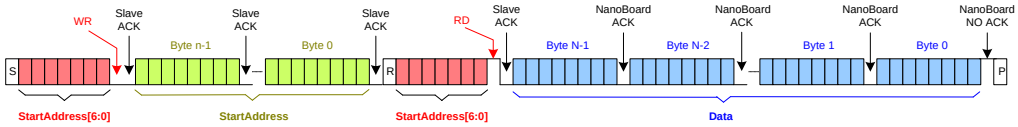
Page Read
(StartAddressLength=0)



Page Read
(StartAddressLength=1)



Page Read
(StartAddressLength=n)



S **Start Condition**

R **Repeated Start Condition**

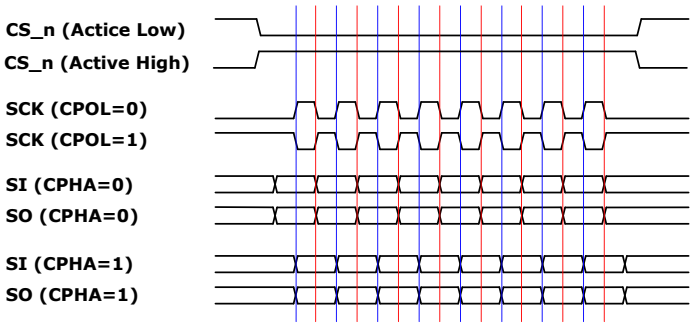
P **Stop Condition**

8. SPI Tasks

In this section the tasks which are used for SPI Master are described in full detail.

8.1. Nano_MasterConfig

```
NANO_RESULT     Nano_SPIMasterConfig (
HANDLE          hDevice,
BYTE            Frequency,
BYTE            Mode
);
```



This function configures the SPI line rate and the SPI operation mode.

hDevice: This is the handle to the USB device.

Frequency: This sets the line rate according to the following table.

Setting	Clock Frequency
NANO_I2C_FREQ_12MHZ	12Mbit/s
NANO_I2C_FREQ_3MHZ	3Mbit/s
NANO_I2C_FREQ_750KHZ	750kBit/s

Mode: This sets the SPI mode.

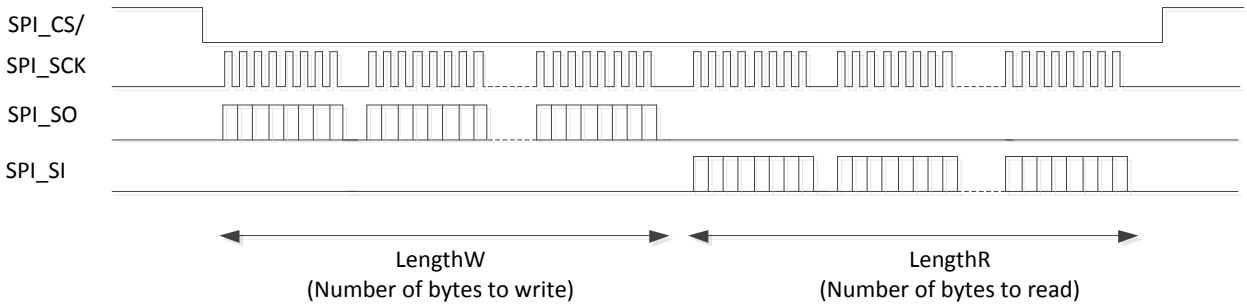
Setting	Clock Frequency
0	SPI bus Mode CPOL=0,CPHA=0
1	SPI bus Mode CPOL=0,CPHA=1
2	SPI bus Mode CPOL=1,CPHA=0
3	SPI bus Mode CPOL=1,CPHA=1

The function returns NANO_RESULT. NANO_SUCCESS is a success and NANO_XACTION_FAILURE is a failure.

8.2. Nano_SPIReadWrite

```
NANO_RESULT     Nano_SPIMasterReadWrite (
                HANDLE   hDevice,
                BYTE     Channel,
                WORD     LengthW,
                BYTE     *pOutBuffer,
                WORD     LengthR,
                BYTE     *pInBuffer
                );
```

This function provides simultaneous write and read, to and from the SPI slave respectively. The function is valid for a particular chip select channel. In order to use this function, one must specify the particular chip select channel, the length of data to be written, the data to be written and the length of data to be read. Upon competition the input buffer will contain the read data.



hDevice: This is the handle to the USB device.

Channel: This specifies the pin which is to be used as a chip select for the read/write. Be sure to configure that pin as an SPI master using the *Nano_PinConfiguration()* function.

Channel	SPI chip select
NANO_SPI_CHANNEL_0	Chip select used is GPIO_00.
NANO_SPI_CHANNEL_1	Chip select used is GPIO_01.
NANO_SPI_CHANNEL_2	Chip select used is GPIO_02.
NANO_SPI_CHANNEL_3	Chip select used is GPIO_03.
. . .	. . .
NANO_SPI_CHANNEL_23	Chip select used is GPIO_23.

LengthW:	This is the length of the data to be written in bytes. Maximum is 4096 bytes.
**pOutBuffer*:	This is a pointer to a buffer to be sent during the write to the SPI slave.
LengthR:	This is the length of the data to be read in bytes. Maximum is 4096 bytes.
**pInBuffer*:	This is a pointer to a buffer to be filled with data from reading the SPI slave.

The function returns NANO_RESULT. NANO_SUCCESS is a success and NANO_XACTION_FAILURE is a failure.

8.3. Nano_SPIWrite

```
NANO_RESULT     Nano_SPIMasterWrite (
                HANDLE    hDevice,
                BYTE      Channel,
                WORD      Length,
                BYTE      *pOutBuffer
                );
```

This function provides a write-only to the SPI slave. The function is valid for a particular chip select channel. In order to use this function, one must specify the particular chip select channel, the length of data to be transferred and fill the output buffer with the data to be sent.

hDevice: This is the handle to the USB device.

Channel: This specifies the pin which is to be used as a chip select for the read/write. Be sure to configure that pin as an SPI master using the *Nano_PinConfiguration()* function.

Channel	SPI chip select
NANO_SPI_CHANNEL_0	Chip select used is GPIO_00.
NANO_SPI_CHANNEL_1	Chip select used is GPIO_01.
NANO_SPI_CHANNEL_2	Chip select used is GPIO_02.
NANO_SPI_CHANNEL_3	Chip select used is GPIO_03.
. . .	. . .
NANO_SPI_CHANNEL_23	Chip select used is GPIO_23.

Length: This is the length of the data to be written in bytes. Maximum is 4096 bytes.

**pOutBuffer*: This is a pointer to a buffer to be sent during the write to the SPI slave.

The function returns NANO_RESULT. NANO_SUCCESS is a success and NANO_XACTION_FAILURE is a failure.

8.4. Nano_SPIRead

```
NANO_RESULT     Nano_SPIMasterRead (
                HANDLE    hDevice,
                BYTE      Channel,
                WORD      Length,
                BYTE      *pInBuffer
                );
```

This function provides a read-only to the SPI slave. The function is valid for a particular chip select channel. In order to use this function, one must specify the particular chip select channel and the length of data to be read. Upon competition the input buffer will contain the read data.

hDevice: This is the handle to the USB device.

Channel: This specifies the pin which is to be used as a chip select for the read/write. Be sure to configure that pin as an SPI master using the *Nano_PinConfiguration()* function.

Channel	SPI chip select
NANO_SPI_CHANNEL_0	Chip select used is GPIO_00.
NANO_SPI_CHANNEL_1	Chip select used is GPIO_01.
NANO_SPI_CHANNEL_2	Chip select used is GPIO_02.
NANO_SPI_CHANNEL_3	Chip select used is GPIO_03.
. . .	. . .
NANO_SPI_CHANNEL_23	Chip select used is GPIO_23.

Length: This is the length of the data to be read in bytes. Maximum is 4096 bytes.

**pInBuffer*: This is a pointer to a buffer to be filled with data from reading the SPI slave.

The function returns NANO_RESULT. NANO_SUCCESS is a success and NANO_XACTION_FAILURE is a failure.

9. Analogue Input Tasks

In this section the tasks for the Analogue Input interface are described in full detail.

9.1. Nano_ADCRead

```
NANO_RESULT     Nano_ADCRead (
                HANDLE      hDevice,
                BYTE        Channel,
                BYTE        *pValue
                )
```

This function allows reading of one of the 4 analogue inputs.

hDevice:	This is the handle to the USB device.
Channel:	This is the particular analogue channel to read. Selection can be 0, 1, 2 or 3.
**pValue*:	This is a pointer to be filled with the analogue to digital converted input data for the selected channel. Conversion is made with 8-bit resolution. 0x00 is minimum and 0xFF is maximum.

The function returns NANO_RESULT. NANO_SUCCESS is a success and NANO_XACTION_FAILURE is a failure.

10. Pre-defined Types and Constants

10.1. NANO_RESULT

A type is defined to describe different exit results from NanoBoard tasks.

```
typedef enum
{
    NANO_SUCCESS = 0,                   // Call was successful
    NANO_XACTION_FAILURE = 1,           // There was an error in the USB transaction
    NANO_HW_NOT_FOUND = 2,              // No hardware was found
    NANO_BAD_PARAMETER = 3,             // Bad or out of range parameters
    NANO_I2C_PROTOCOL_ERROR = 4         // There was an error discovered in the IIC transfer
                                        // (for example an ACK error)
} NANO_RESULT;
```

10.2. NANO_I2C_FREQ

A type is defined to declare different IIC clock frequencies.

```
typedef enum
{
    NANO_I2C_FREQ_1MHZ = 0,             // 1MHz
    NANO_I2C_FREQ_FAST = 1,             // 400kHz (Fast Mode)
    NANO_I2C_FREQ_200KHZ = 2,           // 200kHz
    NANO_I2C_FREQ_STD = 3,              // 100kHz (Standard Mode)
    NANO_I2C_FREQ_50KHZ = 4             // 50kHz
} NANO_I2C_FREQ;
```

10.3. NANO_SPI_FREQ

A type is defined to declare different SPI clock frequencies.

```
typedef enum
{
    NANO_SPI_FREQ_12MHZ = 0,            // 12MHz
    NANO_SPI_FREQ_3MHZ = 1,             // 3MHz
    NANO_SPI_FREQ_750KHZ = 2            // 750kHz
} NANO_SPI_FREQ;
```

10.4. NANO_SPI_MODE

A type is defined to declare different SPI modes.

```
typedef enum
{
        NANO_SPI_MODE_00  = 0,                  //  Mode 0,0
        NANO_SPI_MODE_01  = 1,                  //  Mode 0,1
        NANO_SPI_MODE_10  = 2,                  //  Mode 1,0
        NANO_SPI_MODE_11  = 3                   //  Mode 1,0
} NANO_SPI_MODE;
```

10.5. NANO_SPI_CHANNEL

A type is defined to declare which pin is used as the SPI chip select.

```
typedef enum
{
        NANO_SPI_CHANNEL_0   = 0,       // Corresponds to GPIO_00
        NANO_SPI_CHANNEL_1   = 1,       // Corresponds to GPIO_01
        NANO_SPI_CHANNEL_2   = 2,       // Corresponds to GPIO_02
        NANO_SPI_CHANNEL_3   = 3,       // Corresponds to GPIO_03
        NANO_SPI_CHANNEL_4   = 4,       // Corresponds to GPIO_04
        NANO_SPI_CHANNEL_5   = 5,       // Corresponds to GPIO_05
        NANO_SPI_CHANNEL_6   = 6,       // Corresponds to GPIO_06
        NANO_SPI_CHANNEL_7   = 7,       // Corresponds to GPIO_07
        NANO_SPI_CHANNEL_8   = 8,       // Corresponds to GPIO_08
        NANO_SPI_CHANNEL_9   = 9,       // Corresponds to GPIO_09
        NANO_SPI_CHANNEL_10  = 10,      // Corresponds to GPIO_10
        NANO_SPI_CHANNEL_11  = 11,      // Corresponds to GPIO_11
        NANO_SPI_CHANNEL_12  = 12,      // Corresponds to GPIO_12
        NANO_SPI_CHANNEL_13  = 13,      // Corresponds to GPIO_13
        NANO_SPI_CHANNEL_14  = 14,      // Corresponds to GPIO_14
        NANO_SPI_CHANNEL_15  = 15,      // Corresponds to GPIO_15
        NANO_SPI_CHANNEL_16  = 16,      // Corresponds to GPIO_16
        NANO_SPI_CHANNEL_17  = 17,      // Corresponds to GPIO_17
        NANO_SPI_CHANNEL_18  = 18,      // Corresponds to GPIO_18
        NANO_SPI_CHANNEL_19  = 19,      // Corresponds to GPIO_19
        NANO_SPI_CHANNEL_20  = 20,      // Corresponds to GPIO_20
        NANO_SPI_CHANNEL_21  = 21,      // Corresponds to GPIO_21
        NANO_SPI_CHANNEL_22  = 22,      // Corresponds to GPIO_22
} NANO_SPI_CHANNEL;
```

10.6. NANO_PIN_MODE

A type is defined to select between different pin modes for the GPIOs.

```
typedef enum
{
        GPIO_PIN          = 0,      // Pin used as a GPIO
        ADC_PIN           = 1,      // Pin used as an ADC Input
        SPI_CS_ACTIVE_LO  = 2,      // Pin used as an SPI Chip select (active high)
        SPI_CS_ACTIVE_HI  = 3       // Pin used as an SPI Chip select (active low)
} NANO_PIN_MODE;
```

10.7. DEV_LIST

A type is declared to contain a list of all I2C addresses for devices connected to the I2C bus. If the element of the array is TRUE then the index for this element is the I2C address for a connected device – i.e. if List[0x50] is TRUE, then an I2C device with device ID of 0x50 is connected. If FALSE then a chip is not connected with that device ID.

```
typedef struct
{
        BOOL List[128];
} DEV_LIST;
```