Miniboard API Specification

Nano River Technologies
February 2009

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 MiniBoard.

The board consists of three interfaces – a general purpose IO (GPIO) interface, an IIC interface and an SPI interface. 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 in details all tasks associated with the GPIO interface. Chapter 6 describes tasks associated with the IIC interface. Similarly chapter 7 describes tasks associated with the SPI interface.

In chapter 8 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 MiniBoard API. In order to see how these should be used in a real application, please refer to the "**MiniBoard Application Examples"** document.

miniboard.h:	This is an include file for the MiniBoard. It contains types and constants that you can refer to in calling the tasks/functions.
minboard_class.h:	This is the header file for the MiniBoard Class Library.
minboard_class.cpp:	This is the C++ file for the MiniBoard 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 MiniBoard.

3.2. 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.

3.3. IIC Tasks

IIC Configuration:

- **Nano_IICSetFrequency** This function sets the rate for the IIC clock.
- **Nano_IICGetFrequency** This function returns the rate for the IIC clock.

IIC High Level Commands:

- **Nano_IICRead** This function performs an IIC read operation.
- **Nano_IICWrite** This function performs an IIC write operation.
- **Nano_IICScanConnectedDevices** This function returns all IIC devices connected to the IIC bus.

IIC Wire Level Commands:

- **Nano_IICReadSCL** This function reads the level of the IIC SCL line.
- **Nano_IICReadSDA** This function reads the level of the IIC SDA line.
- **Nano_IICWriteSCL** This function sets the level of the IIC SCL line.
- **Nano_IICWriteSDA** This function sets the level of the IIC SDA line.

3.4. SPI Tasks

SPI Configuration:

- **Nano_SPISetClockConfig** This function sets the clocking mode for the SPI transfer.
- **Nano_SPIGetClockConfig** This function returns the clocking mode for the SPI transfer.
- **Nano_SPISetFrequency** This function sets the rate for the SPI clock.
- **Nano_SPIGetFrequency** This function returns the rate for the SPI clock.
- **Nano_SPISetChannelConfig** This function allows configuration of the GPIOs as additional chip selects for SPI transfers.
- **Nano_SPIGetChannelConfig** This function returns the configuration of the additional GPIO based chip selects for SPI transfers.

SPI High Level Commands:

- **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.

4. USB Tasks

4.1. OpenDevice

bool **OpenDevice ( )**

This function tries to open a connection to the MiniBoard. 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 MiniBoard functions are called.

5. GPIO Tasks

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

5.1. Nano_GPIOSetDirection

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

This function sets the direction for all 32 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 31 corresponds to GPIO_31.
Mask:	This is a mask signifying which GPIO direction pins are affected. For a bit to be updated then the corresponding mask bit must be set (=1). Bit 0 corresponds to GPIO_00. Bit 31 corresponds to GPIO_31.

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

5.2. Nano_GPIOGetDirection

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

This function returns the direction value for all 32 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.

5.3. Nano_GPIOWrite

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

This function writes to all 32 bits of GPIO.

hDevice:	This is the handle to the USB device.
Value:	This is the required value. ***The GPIO will only be written to if the direction bit is configured for output.*** 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 31 corresponds to GPIO_31.
Mask:	This is a mask signifying which GPIO pins are affected. For a bit to be updated then the corresponding mask bit must be set (=1). Bit 0 corresponds to GPIO_00. Bit 31 corresponds to GPIO_31.

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

5.4. Nano_GPIORead

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

This function returns the value of all 32 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 31 corresponds to GPIO_31.

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

5.5. 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 31 corresponds to GPIO_31.
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.

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

5.6. 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 31 corresponds to GPIO_31.
**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.

5.7. 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 31 corresponds to GPIO_31.
Value:	This is the required value. ***The GPIO will only be written to if the direction bit is configured for output.*** TRUE means that the GPIO direction bit should be driven high. FALSE means that the direction bit should be driven low.

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

5.8. 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 31 corresponds to GPIO_31.
**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.

6. IIC Tasks

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

6.1. Nano_IICSetFrequency

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

This function sets the clock frequency for IIC transfers.

hDevice: This is the handle to the USB device.

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

Setting	Clock Frequency
NANO_IIC_FREQ_FAST	400kHz (Fast Mode)
NANO_IIC_FREQ_200KHZ	200kHz
NANO_IIC_FREQ_STD	100kHz (Standard Mode)
NANO_IIC_FREQ_80KHZ	80kHz
NANO_IIC_FREQ_60KHZ	60kHz
NANO_IIC_FREQ_40KHZ	40kHz
NANO_IIC_FREQ_20KHZ	20kHz
NANO_IIC_FREQ_10KHZ	10kHz

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

6.2. Nano_IICGetFrequency

NANO_RESULT **Nano_IICGetFrequency (**
HANDLE *hDevice,*
BYTE **pFrequency*
)

This function returns the clock frequency setting for IIC transfers.

hDevice: This is the handle to the USB device.

**pFrequency*: This is a pointer to be filled with the IIC frequency setting. The following table relates the setting constant to the actual observed clock frequency.

Setting	Clock Frequency
NANO_IIC_FREQ_FAST	400kHz (Fast Mode)
NANO_IIC_FREQ_200KHZ	200kHz
NANO_IIC_FREQ_STD	100kHz (Standard Mode)
NANO_IIC_FREQ_80KHZ	80kHz
NANO_IIC_FREQ_60KHZ	60kHz
NANO_IIC_FREQ_40KHZ	40kHz
NANO_IIC_FREQ_20KHZ	20kHz
NANO_IIC_FREQ_10KHZ	10kHz

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

6.3. Nano_IICRead

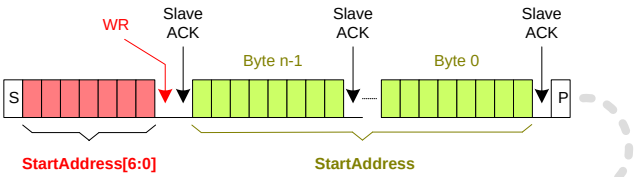
```
NANO_RESULT     Nano_IICRead (
                HANDLE      hDevice,
                BYTE        SlaveAddress,
                BYTE        StartAddressLength,
                ULONG       StartAddress,
                WORD        BufferLength,
                BYTE        *pInBuffer
               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

This function provides a complete IIC read command, reading up to 256 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.

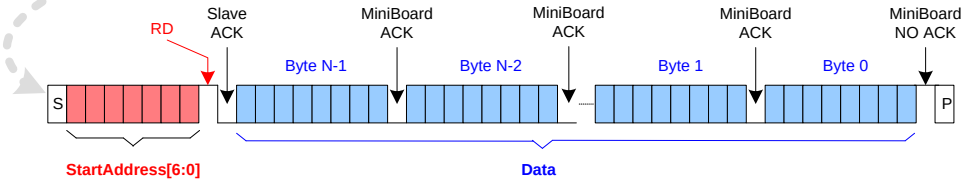
hDevice:	This is the handle to the USB device.
SlaveAddress:	This is the IIC slave address (device ID).
StartAddressLength:	This is the width (in bytes) of the start address. The valid range is 0-4 bytes. If 0 is specified then simply no start address is included.
StartAddress:	This is starting address within the device for the transfer.
BufferLength:	Sets the buffer length for the transfer. Maximum is 256 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_IIC_PROTOCOL_ERROR then there has been an error in the IIC 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.

Write (No Data)



Page Read



6.4. Nano_IICWrite

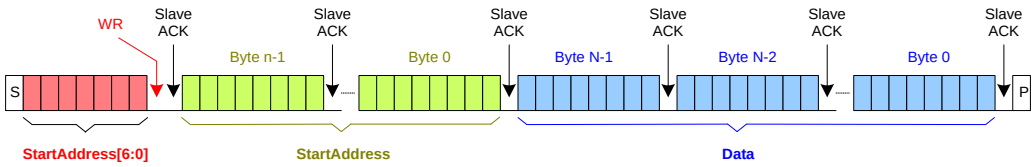
```
NANO_RESULT     Nano_IICWrite (
                HANDLE      hDevice,
                BYTE        SlaveAddress,
                BYTE        StartAddressLength,
                ULONG       StartAddress,
                WORD        BufferLength,
                BYTE        *pOutBuffer
             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

This function provides a complete IIC write command, writing up to 256 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 IIC 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 256 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_IIC_PROTOCOL_ERROR then there has been an error in the IIC 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



6.5. Nano_IICScanConnectedDevices

```
NANO_RESULT     Nano_IICScanConnectedDevices (
                HANDLE      hDevice,
                DEV_LIST    pList
                )
```

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

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

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

6.6. Nano_IICReadSCL

```
NANO_RESULT     Nano_IICReadSCL (
                HANDLE      hDevice,
                BOOL        *pState
                )
```

This function reads the state of the SCL line of the IIC bus.

hDevice:	This is the handle to the USB device.
**pState*:	This is a pointer to be filled with the state of the SCL line. When TRUE, then SCL is high. When FALSE, then the SCL line is driven low.

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

6.7. Nano_IICReadSDA

```
NANO_RESULT     Nano_IICReadSDA (
                HANDLE      hDevice,
                BOOL        *pState
                )
```

This function reads the state of the SDA line of the IIC bus.

hDevice:	This is the handle to the USB device.
**pState*:	This is a pointer to be filled with the state of the SDA line. When TRUE, then SDA is high. When FALSE, then the SDA line is driven low.

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

6.8. Nano_IICWriteSCL

```
NANO_RESULT     Nano_IICWriteSCL (
                HANDLE      hDevice,
                BOOL        pState
                )
```

This function drives or releases the SCL line of the IIC bus.

hDevice:	This is the handle to the USB device.
pState:	This sets the drive of the SDL line. When TRUE, then SCL is not driven. When FALSE, then the SCL line is driven low.

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

6.9. Nano_IICWriteSDA

```
NANO_RESULT     Nano_IICWriteSDA (
                HANDLE      hDevice,
                BOOL        pState
                )
```

This function drives or releases the SDA line of the IIC bus.

hDevice:	This is the handle to the USB device.
pState:	This sets the drive of the SDA line. When TRUE, then SDA is not driven. When FALSE, then the SDA line is driven low.

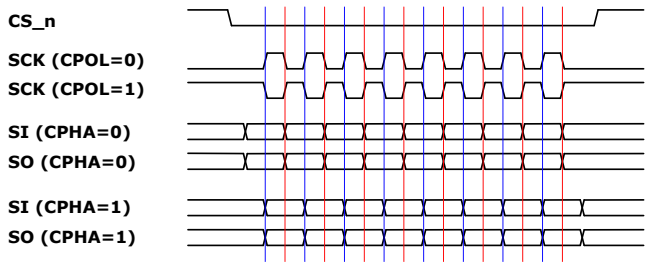
The function returns NANO_RESULT. NANO_SUCCESS is a success and NANO_XACTION_FAILURE is a failure.

7. SPI Tasks

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

7.1. Nano_SPISetClockConfig

```
NANO_RESULT     Nano_SPISetClockConfig (
                HANDLE      hDevice,
                BYTE        CPOL,
                BYTE        CPHA
                )
```



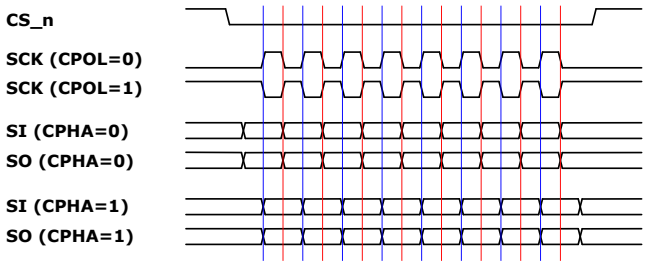
This function sets the SPI clock polarity and phase.

hDevice:	This is the handle to the USB device.
CPOL:	This sets is the desired polarity for the SPI clock. - 0 corresponds to low when idle - 1 corresponds to high when idle
CPHA:	This sets is the desired phase for the SPI clock. - 0 corresponds to data valid on leading clock edge - 1 corresponds to data valid on trailing clock edge

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

7.2. Nano_SPIGetClockConfig

```
NANO_RESULT     Nano_SPIGetClockConfig (
                HANDLE      hDevice,
                BYTE        *pCPOL,
                BYTE        *pCPHA
                )
```



This function returns the SPI clock polarity and phase.

hDevice: This is the handle to the USB device.

**pCPOL*: This is pointer to be filled with the polarity setting for the SPI clock.
- 0 corresponds to low when idle
- 1 corresponds to high when idle

**pCPHA*: This is a pointer to be filled with the phase setting for the SPI clock.
- 0 corresponds to data valid on leading clock edge
- 1 corresponds to data valid on trailing clock edge

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

7.3. Nano_SPISetFrequency

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

This function sets the clock frequency for SPI transfers.

hDevice: This is the handle to the USB device.

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

Setting	Clock Frequency
NANO_SPI_FREQ_400KHZ	400kHz
NANO_SPI_FREQ_200KHZ	200kHz
NANO_SPI_FREQ_100KHZ	100kHz
NANO_SPI_FREQ_80KHZ	80kHz
NANO_SPI_FREQ_60KHZ	60kHz
NANO_SPI_FREQ_40KHZ	40kHz
NANO_SPI_FREQ_20KHZ	20kHz
NANO_SPI_FREQ_10KHZ	10kHz

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

7.4. Nano_SPIGetFrequency

NANO_RESULT **Nano_SPIGetFrequency (**
HANDLE *hDevice,*
BYTE **pFrequency*
)

This function returns the clock frequency setting for SPI transfers.

hDevice: This is the handle to the USB device.

**pFrequency*: This is a pointer to be filled with the SPI frequency setting. The following table relates the setting constant to the actual observed clock frequency.

Setting	Clock Frequency
NANO_SPI_FREQ_400KHZ	400kHz
NANO_SPI_FREQ_200KHZ	200kHz
NANO_SPI_FREQ_100KHZ	100kHz
NANO_SPI_FREQ_80KHZ	80kHz
NANO_SPI_FREQ_60KHZ	60kHz
NANO_SPI_FREQ_40KHZ	40kHz
NANO_SPI_FREQ_20KHZ	20kHz
NANO_SPI_FREQ_10KHZ	10kHz

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

7.5. Nano_SPISetChannelConfig

```
NANO_RESULT    Nano_SPISetChannelConfig (
               HANDLE    hDevice,
               BYTE      Channel,
               BYTE      Configuration
               )
```

The GPIO pins can individually be used as additional chip selects for SPI transfers. This function allows configuration of a particular GPIO for this purpose.

hDevice: This is the handle to the USB device.

Channel: This specifies a GPIO pin to be configured as the chip select for SPI transfers. Multiple chip select pins can be used by calling this function multiple times and specifying different channel settings.

Setting	Meaning
NANO_SPI_CHIPSELECT_GPIO_0	Configure the chip select when using GPIO 0
NANO_SPI_CHIPSELECT_GPIO_1	Configure the chip select when using GPIO 1
NANO_SPI_CHIPSELECT_GPIO_2	Configure the chip select when using GPIO 2
. . .	. . .
NANO_SPI_CHIPSELECT_GPIO_29	Configure the chip select when using GPIO 29
NANO_SPI_CHIPSELECT_GPIO_30	Configure the chip select when using GPIO 30
NANO_SPI_CHIPSELECT_GPIO_31	Configure the chip select when using GPIO 31

Configuration: This specifies the desired behaviour of the chip select.

Setting	Meaning
NANO_SPI_CHIPSELECT_OFF	Disable this pin as a chip select
NANO_SPI_CHIPSELECT_ACTIVE_LOW	Configure the chip select as active low
NANO_SPI_CHIPSELECT_ACTIVE_HIGH	Configure the chip select as active high

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

7.6. Nano_SPIGetChannelConfig

```
NANO_RESULT    Nano_SPIGetChannelConfig (
               HANDLE    hDevice,
               BYTE      Channel,
               BYTE      *pConfiguration
               )
```

The GPIO pins can individually be used as additional chip selects for SPI transfers. This function returns the configuration of a particular GPIO based chip select.

hDevice: This is the handle to the USB device.

Channel: This specifies which GPIO pin is to be checked as the chip select for SPI transfers. Multiple chip select pins can be checked by calling this function multiple times and specifying different channel settings.

Setting	Meaning
NANO_SPI_CHIPSELECT_GPIO_0	Configure the chip select when using GPIO 0
NANO_SPI_CHIPSELECT_GPIO_1	Configure the chip select when using GPIO 1
NANO_SPI_CHIPSELECT_GPIO_2	Configure the chip select when using GPIO 2
. . .	. . .
NANO_SPI_CHIPSELECT_GPIO_29	Configure the chip select when using GPIO 29
NANO_SPI_CHIPSELECT_GPIO_30	Configure the chip select when using GPIO 30
NANO_SPI_CHIPSELECT_GPIO_31	Configure the chip select when using GPIO 31

**pConfiguration*: This is a pointer to be filled with the current behaviour of the chip select.

Setting	Meaning
NANO_SPI_CHIPSELECT_OFF	Disable this pin as a chip select
NANO_SPI_CHIPSELECT_ACTIVE_LOW	Configure the chip select as active low
NANO_SPI_CHIPSELECT_ACTIVE_HIGH	Configure the chip select as active high

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

7.7. Nano_SPIReadWrite

```
NANO_RESULT     Nano_SPIReadWrite (
                HANDLE      hDevice,
                BYTE        Channel,
                BYTE        *pInBuffer,
                BYTE        *pOutBuffer,
                WORD        Length,
                )
```

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 transferred and fill the output buffer with the data to be sent. 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 using the ***Nano_SPISetChannelConfig()*** function.

Setting	Meaning
NANO_SPI_CHIPSELECT_CS	Use the dedicated chip select CS pin
NANO_SPI_CHIPSELECT_GPIO_0	Use GPIO_0 as the chip select
NANO_SPI_CHIPSELECT_GPIO_1	Use GPIO_1 as the chip select
NANO_SPI_CHIPSELECT_GPIO_2	Use GPIO_2 as the chip select
. . .	. . .
NANO_SPI_CHIPSELECT_GPIO_29	Use GPIO_29 as the chip select
NANO_SPI_CHIPSELECT_GPIO_30	Use GPIO_30 as the chip select
NANO_SPI_CHIPSELECT_GPIO_31	Use GPIO_31 as the chip select

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

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

Length: This is the length of the transfer in bytes. Maximum is 2^{16}-1 meaning 2^{16}-1 bytes are read and 2^{16}-1 bytes are written in one transfer.

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

7.8. Nano_SPIRead

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

This function provides reading from 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 transferred. 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. Be sure to configure that pin using the ***Nano_SPISetChannelConfig()*** function.

Setting	Meaning
NANO_SPI_CHIPSELECT_CS	Use the dedicated chip select CS pin
NANO_SPI_CHIPSELECT_GPIO_0	Use GPIO_0 as the chip select
NANO_SPI_CHIPSELECT_GPIO_1	Use GPIO_1 as the chip select
NANO_SPI_CHIPSELECT_GPIO_2	Use GPIO_2 as the chip select
. . .	. . .
NANO_SPI_CHIPSELECT_GPIO_29	Use GPIO_29 as the chip select
NANO_SPI_CHIPSELECT_GPIO_30	Use GPIO_30 as the chip select
NANO_SPI_CHIPSELECT_GPIO_31	Use GPIO_31 as the chip select

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

Length: This is the length of the transfer in bytes. Maximum is 2^{16}-1 bytes.

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

7.9. Nano_SPIWrite

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

This function provides a write 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 write. Be sure to configure that pin using the ***Nano_SPISetChannelConfig()*** function.

Setting	Meaning
NANO_SPI_CHIPSELECT_CS	Use the dedicated chip select CS pin
NANO_SPI_CHIPSELECT_GPIO_0	Use GPIO_0 as the chip select
NANO_SPI_CHIPSELECT_GPIO_1	Use GPIO_1 as the chip select
NANO_SPI_CHIPSELECT_GPIO_2	Use GPIO_2 as the chip select
. . .	. . .
NANO_SPI_CHIPSELECT_GPIO_29	Use GPIO_29 as the chip select
NANO_SPI_CHIPSELECT_GPIO_30	Use GPIO_30 as the chip select
NANO_SPI_CHIPSELECT_GPIO_31	Use GPIO_31 as the chip select

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

Length: This is the length of the transfer in bytes. Maximum is 2^{16}-1 bytes.

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

8. Pre-defined Types and Constants

8.1. NANO_RESULT

A type is defined to describe different exit results from Miniboard 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_IIC_PROTOCOL_ERROR = 4         // There was an error discovered in the IIC transfer
                                            // (for example an ACK error)
} NANO_RESULT;
```

8.2. NANO_IIC_FREQ

A type is defined to declare different IIC clock frequencies.

```
typedef enum
{
        NANO_IIC_FREQ_FAST = 0,             // 400kHz (Fast Mode)
        NANO_IIC_FREQ_200KHZ = 1,           // 200kHz
        NANO_IIC_FREQ_STD = 2,              // 100kHz (Standard Mode)
        NANO_IIC_FREQ_80KHZ = 3,            // 80kHz
        NANO_IIC_FREQ_60KHZ = 4,            // 60kHz
        NANO_IIC_FREQ_40KHZ = 5,            // 40kHz
        NANO_IIC_FREQ_20KHZ = 6,            // 20kHz
        NANO_IIC_FREQ_10KHZ = 7             // 10kHz
} NANO_IIC_FREQ;
```

8.3. NANO_SPI_FREQ

A type is defined to declare different SPI clock frequencies.

```
typedef enum
{
        NANO_SPI_FREQ_400KHZ = 0,           // 400kHz
        NANO_SPI_FREQ_200KHZ = 1,           // 200kHz
        NANO_SPI_FREQ_100KHZ = 2,           // 100kHz
        NANO_SPI_FREQ_80KHZ = 3,            // 80kHz
        NANO_SPI_FREQ_60KHZ = 4,            // 60kHz
        NANO_SPI_FREQ_40KHZ = 5,            // 40kHz
        NANO_SPI_FREQ_20KHZ = 6,            // 20kHz
        NANO_SPI_FREQ_10KHZ = 7             // 10kHz
} NANO_SPI_FREQ;
```

8.4. NANO_SPI_CHIPSELECT

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

```
typedef enum
{
        NANO_SPI_CHIPSELECT_CS = 0,         // Use the dedicated SPI_CSn
        NANO_SPI_CHIPSELECT_GPIO_0 = 1,     // Use GPIO_00
        NANO_SPI_CHIPSELECT_GPIO_1 = 2,     // Use GPIO_01
        NANO_SPI_CHIPSELECT_GPIO_2 = 3,     // Use GPIO_02
        NANO_SPI_CHIPSELECT_GPIO_3 = 4,     // Use GPIO_03
        NANO_SPI_CHIPSELECT_GPIO_4 = 5,     // Use GPIO_04
        NANO_SPI_CHIPSELECT_GPIO_5 = 6,     // Use GPIO_05
        NANO_SPI_CHIPSELECT_GPIO_6 = 7,     // Use GPIO_06
        NANO_SPI_CHIPSELECT_GPIO_7 = 8,     // Use GPIO_07
        NANO_SPI_CHIPSELECT_GPIO_8 = 9,     // Use GPIO_08
        NANO_SPI_CHIPSELECT_GPIO_9 = 10,    // Use GPIO_09
        NANO_SPI_CHIPSELECT_GPIO_10 = 11,   // Use GPIO_10
        NANO_SPI_CHIPSELECT_GPIO_11 = 12,   // Use GPIO_11
        NANO_SPI_CHIPSELECT_GPIO_12 = 13,   // Use GPIO_12
        NANO_SPI_CHIPSELECT_GPIO_13 = 14,   // Use GPIO_13
        NANO_SPI_CHIPSELECT_GPIO_14 = 15,   // Use GPIO_14
        NANO_SPI_CHIPSELECT_GPIO_15 = 16,   // Use GPIO_15
        NANO_SPI_CHIPSELECT_GPIO_16 = 17,   // Use GPIO_16
        NANO_SPI_CHIPSELECT_GPIO_17 = 18,   // Use GPIO_17
        NANO_SPI_CHIPSELECT_GPIO_18 = 19,   // Use GPIO_18
        NANO_SPI_CHIPSELECT_GPIO_19 = 20,   // Use GPIO_19
        NANO_SPI_CHIPSELECT_GPIO_20 = 21,   // Use GPIO_20
        NANO_SPI_CHIPSELECT_GPIO_21 = 22,   // Use GPIO_21
        NANO_SPI_CHIPSELECT_GPIO_22 = 23,   // Use GPIO_22
```

```
        NANO_SPI_CHIPSELECT_GPIO_23 = 24,       // Use GPIO_23
        NANO_SPI_CHIPSELECT_GPIO_24 = 25,       // Use GPIO_24
        NANO_SPI_CHIPSELECT_GPIO_25 = 26,       // Use GPIO_25
        NANO_SPI_CHIPSELECT_GPIO_26 = 27,       // Use GPIO_26
        NANO_SPI_CHIPSELECT_GPIO_27 = 28,       // Use GPIO_27
        NANO_SPI_CHIPSELECT_GPIO_28 = 29,       // Use GPIO_28
        NANO_SPI_CHIPSELECT_GPIO_29 = 30,       // Use GPIO_29
        NANO_SPI_CHIPSELECT_GPIO_30 = 31,       // Use GPIO_30
        NANO_SPI_CHIPSELECT_GPIO_31 = 32        // Use GPIO_31
} NANO_SPI_CHIPSELECT;
```

8.5. NANO_SPI_CHIPSELECT_ENABLE

A type is defined to select between active high or active low SPI chip select or to disable this completely.

```
typedef enum
{
        NANO_SPI_CHIPSELECT_OFF = 0,                // Disabled
        NANO_SPI_CHIPSELECT_ACTIVE_LOW = 1,         // Active low
        NANO_SPI_CHIPSELECT_ACTIVE_HIGH = 2,        // Active high
} NANO_SPI_CHIPSELECT_ENABLE;
```

8.6. DEV_LIST

A type is declared to contain a list of all IIC addresses for devices connected to the IIC bus. If the element of the array is TRUE then the index for this element is the IIC address for a connected device – i.e. if List[0x50] is TRUE, then an IIC 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;
```