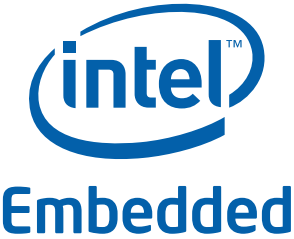

Intel® Platform Controller Hub EG20T

Serial Peripheral Interface (SPI) Driver for Windows* Programmer's Guide

February 2011

Order Number: 324260-002US

Contents

Tables

Revision History

Date	Revision	Description
February 2011	002	Updated Section 2.0, “Operating System (OS) Support” on page 6 Added Section 5.1.1, “Using GUID Interface Exposed by the Driver” on page 12
September 2010	001	Initial release

1.0 Introduction

This document provides the programming details of the Serial Peripheral Interface (SPI) driver for Windows*. This includes information about the interfaces exposed by the driver and how to use those interfaces to drive the SPI hardware.

The SPI bus is a communication bus that operates in full duplex mode. Devices communicate in master/slave mode, in which the master device initiates the data transfer. The SPI hardware supports four different modes for communication.

2.0 Operating System (OS) Support

The SPI driver is supported by the following operating systems:

No	OS	Notes
1	Microsoft Windows XP*	Service Pack 3
2	Windows Embedded Standard*	2009
3	Windows Embedded POSReady*	2009
4	Microsoft Windows 7*	
5	Windows Embedded Standard7	

3.0 Dependencies

This driver is only dependent upon appropriate OS driver installation. Also, this driver is not dependent upon any other software delivered.

4.0 SPI Driver API Details

This section provides information about the interfaces exposed by the SPI driver. The current implementation of the driver exposes the interfaces through Input/Output Controls (IOCTLs), which can be called from the application (user mode) using the Win32 API DeviceIoControl (refer to the MSDN documentation for more details on this API). The following sections provide information about the IOCTLs and how to use them to configure the SPI hardware to work properly.

4.1 Features

The SPI Driver allows setting different configurations for SPI hardware. It supports:

- Master mode only
- Either 8-bit (byte) or 16-bit (word) transfer size
- Setting serial clock rate for transfer up to 5 Mbps
- Bus master byte/multi-byte read transactions
- Bus master byte/multi-byte write transactions
- Different modes – Mode 0, Mode 1, Mode 2 and Mode 3
- Either LSB first or MSB first data transfer

4.2 Interface Details

Table 1 lists the IOCTLs supported by the driver.

Table 1. Supported IOCTLs

No	IOCTL	Remarks
1	IOCTL_SPI_CONFIG	This IOCTL is used for configuration information such as mode, LSB first or MSB first, bits per word, baud rate, etc. for the hardware.
2	IOCTL_SPI_ENABLE_INT	This IOCTL is used to enable the interrupts.
3	IOCTL_SPI_DISABLE_INT	This IOCTL is used to disable the interrupts.
4	IOCTL_SPI_READ	This IOCTL is used to read information from the devices connected to the SPI hardware.
5	IOCTL_SPI_WRITE	This IOCTL is used to write information to the devices connected to the SPI hardware.
6	IOCTL_SPI_GET_CONFIG	This IOCTL is used to get current configuration details.

4.3 IOCTL Usage Details

This section assumes a single-client model, in which there is a single application-level program configuring the SPI interface and initiating I/O operations. The following files contain the details of the IOCTLs and data structures used:

- ioh_spi_ioctls.h – contains IOCTL definitions
- ioh_spi_common.h – data structures and other variables used by the IOCTLs

4.3.1 IOCTL_SPI_CONFIG

Before doing any operation, the interface must be initialized and configured. This IOCTL is used to initialize and configure the SPI interface. The prerequisite for this is that the device must be installed and opened using the Win32 API CreateFile.

```
IOH_SPI_CONFIG spiConfig = {0};

spiConfig.SlaveNo = 1;

spiConfig.BaudRate = 10000;

spiConfig.BitsPerWord = IOH_SPI_8_BPW;

spiConfig.Mode = IOH_SPI_MODE_0;

DeviceIoControl(hHandle,IOCTL_SPI_CONFIG,&spiConfig,sizeof(spiConfig),
                NULL,0,&dwSize,NULL)
```

4.3.2 IOCTL_SPI_ENABLE_INT

This enables the interrupts of SPI interface.

```
DeviceIoControl(hHandle,IOCTL_SPI_ENABLE_INT,NULL,0,NULL,0,&dwSize,NULL);
```

4.3.3 IOCTL_SPI_DISABLE_INT

This disables the interrupts of SPI interface.

```
DeviceIoControl(hHandle,IOCTL_SPI_DISABLE_INT,NULL,0,NULL,0,&dwSize,NULL);
```

4.3.4 IOCTL_SPI_READ

The read operation requires two buffers to send to the driver. For every read operation there must be a dummy write operation. For example, if you are planning to read 16 bytes of data from a device connected to the SPI interface, you first must write 16 bytes of dummy data to the device.

```
IOH_SPI_RW ioData = {0};

ioData.SpiConfig = spiConfig;

ioData.buffer = dummy_write_buff;//array

ioData.nSize = 64;

DeviceIoControl(hHandle,IOCTL_SPI_READ,&ioData,sizeof(ioData),ReadBuff,
                sizeof(ReadBuff),&dwSize,NULL);
```

4.3.5 IOCTL_SPI_WRITE

This IOCTL is used to perform the write operation.

```
IOH_SPI_RW ioData = {0};

ioData.SpiConfig = spiConfig;

ioData.buffer = write_buff;//array

ioData.nSize = 64;

DeviceIoControl(hHandle,IOCTL_SPI_WRITE,&ioData,sizeof(ioData),NULL,
                NULL,&dwSize,NULL);
```

4.3.6 IOCTL_SPI_GET_CONFIG

This IOCTL is used to get the configuration of the SPI interface.

```
IOH_SPI_CONFIG spiConfig = {0};

DeviceIoControl(hHandle,IOCTL_SPI_GET_CONFIG,NULL,0,

                &spiConfig,sizeof(spiConfig),&dwSize,NULL);
```

4.4 Structures and Macros

This section provides the details on the structures and macros used by interfaces exposed by the SPI driver. All the structures and macros used by the interfaces are defined in ioh_spi_common.h.

4.4.1 Structures

4.4.1.1 IOH_SPI_CONFIG

This structure holds the user supplied configuration information for configuring the SPI controller.

Table 2. IOH_SPI_CONFIG Structure

Name	Description
ULONG SlaveNo	Slave device number
ULONG BaudRate	Baud rate of the device
UCHAR BitsPerWord	Transfer size 8 bit or 16 bit
UCHAR Mode	Mode0,1,2,3 and MSB/LSB first

4.4.1.2 IOH_SPI_RW

This structure is used for the read and write operation. It holds the configuration details, data size and data.

Table 3. IOH_SPI_RW Structure

Name	Description
IOH_SPI_CONFIG SpiConfig	Configuration information. Refer to "IOH_SPI_CONFIG" on page 10.
ULONG nSize	Data size.
PVOID Buffer	Data buffer.

4.4.2 Macros

Table 4. Enumerations (Sheet 1 of 2)

Name	Description
IOH_SPI_CPHA	SPI Clock Phase. This is used to specify SPI MODE information.
IOH_SPI_CPOL	SPI Clock Polarity. This is used to specify SPI MODE information.
IOH_SPI_MODE_0	This specifies MODE as 0. CPHA = 0 and CPOL =0.
IOH_SPI_MODE_1	This specifies the SPI mode as 1.

Table 4. Enumerations (Sheet 2 of 2)

Name	Description
IOH_SPI_MODE_2	This specifies the SPI mode as 2.
IOH_SPI_MODE_3	This specifies the SPI mode as 3.
IOH_SPI_LSB_FIRST	This configures the SPI hardware to perform transfer in LSB First data transfer.
IOH_SPI_8_BPW	This configures the SPI hardware to perform 8-bits per word data transfer.
IOH_SPI_16_BPW	This configures the SPI hardware to perform 16-bits per word data transfer.
IOH_SPI_ENABLE	This is used by the IOCTL_SPI_ENABLE_INT ioctl to enable interrupts in the SPI hardware.
IOH_SPI_DISABLE	This is used by the IOCTL_SPI_DISABLE_INT ioctl to disable interrupts in the SPI hardware.

4.5 Error Handling

Since the IOCTL command is implemented using the Windows* API, the return value of the call is dependent on and defined by the OS. On Windows*, the return value is a non-zero value. If the error is detected within or outside the driver, an appropriate system defined value is returned by the driver.

4.6 Inter-IOCTL Dependencies

There are no inter-IOCTL dependencies. Once the driver has been loaded successfully, the IOCTLs stated above can be used in any order.

5.0 Programming Guide

This section describes the basic procedure for using the SPI driver from a user mode application. All operations are through the IOCTLs exposed by the SPI driver. Refer to Section 4.3 for details on the IOCTLs. The steps involved in accessing the GPIO driver from the user mode application are described below:

- Open the device.
- Initialize and configure the driver with desired settings through the interfaces exposed.
- Perform read/write operations.
- Close the device.

5.1 Opening the Device

SPI driver is opened using the Win32 CreateFile API. To get the device name, refer to Section 5.1.1.

5.1.1 Using GUID Interface Exposed by the Driver

A device interface class is a way of exporting device and driver functionality to other system components, including other drivers, as well as user-mode applications. A driver can register a device interface class, and then enable an instance of the class for each device object to which user-mode I/O requests might be sent. The topcliff IOH SPI driver registers the following interface.

No	Interface Name
1	GUID_DEVINTERFACE_IOHSPI

This is defined ioh_spi_common.h.

Device interfaces are available to both kernel-mode components and user-mode applications. User-mode code can use SetupDiXxx functions to find out about registered, enabled device interfaces.

Please refer the following site to get the details about SetupDiXxx functions.

http://msdn.microsoft.com/en-us/library/dd406734.aspx

5.2 Driver Configuration

The following IOCTLS are used to initialize and configure the settings for the SPI driver:

- IOCTL_SPI_CONFIG
- IOCTL_SPI_ENABLE_INT
- IOCTL_SPI_DISABLE_INT
- IOCTL_GET_SPI_CONFIG

DeviceIoControl Win32 API is used for sending information to the SPI driver.

```
IOH_SPI_CONFIG spiConfig = {0};

spiConfig.SlaveNo = 1;

spiConfig.BaudRate = 10000;
```

```
spiConfig.BitsPerWord = IOH_SPI_8_BPW;

spiConfig.Mode = IOH_SPI_MODE_0;

DeviceIoControl(hHandle,IOCTL_SPI_CONFIG,&spiConfig,sizeof(spiConfig),

                NULL,0,&dwSize,NULL);

...

DeviceIoControl(hHandle,IOCTL_SPI_GET_CONFIG,NULL,0,

                &spiConfig,sizeof(spiConfig),&dwSize,NULL);
```

- IOCTL_SPI_ENABLE_INT

This IOCTL enables all the interrupts.

```
bRet = DeviceIoControl( hDevice,

        IOCTL_SPI_ENABLE_INT,

        NULL,

        0,

        NULL,

        0,

        &dwRet,

        NULL);
```

- IOCTL_SPI_DISABLE_INT

This IOCTL disables all the interrupts.

```
bRet = DeviceIoControl( hDevice,

        IOCTL_SPI_DISABLE_INT,

        NULL,

        0,

        NULL,

        0,

        &dwRet,

        NULL);
```

5.3 Read / Write Operation

IOCTL_SPI_READ and IOCTL_SPI_WRITE are used for read and write operations respectively.

To perform Read/Write Operation, the device needs to be enabled.

```
IOH_SPI_CONFIG spiConfig = {0};

spiConfig.SlaveNo = 1;

spiConfig.BaudRate = 10000;
```

```
spiConfig.BitsPerWord = IOH_SPI_8_BPW;

spiConfig.Mode = IOH_SPI_MODE_0;

DeviceIoControl(hHandle,IOCTL_SPI_CONFIG,&spiConfig,sizeof(spiConfig),

                NULL,0,&dwSize,NULL);

DeviceIoControl(hHandle,IOCTL_SPI_ENABLE_INT,NULL,0,NULL,0,&dwSize,NULL);

IOH_SPI_RW ioData = {0};

ioData.SpiConfig = spiConfig;

ioData.buffer = write_buff;//array

ioData.nSize = 64;

DeviceIoControl(hHandle,IOCTL_SPI_WRITE,&ioData,sizeof(ioData),NULL,

                NULL,&dwSize,NULL);

ioData.SpiConfig = spiConfig;

ioData.buffer = dummy_write_buff;//array

ioData.nSize = 64;

DeviceIoControl(hHandle,IOCTL_SPI_READ,&ioData,sizeof(ioData),ReadBuff,

                sizeof(ReadBuff),&dwSize,NULL);

DeviceIoControl(hHandle,IOCTL_SPI_DISABLE_INT,NULL,0,NULL,0,&dwSize,NULL);
```

5.4 Close the Device

Once all the operations related to the SPI driver are completed, the device handle must be freed by the application by calling the Win32 API CloseHandle.

```
CloseHandle(hHandle);
```