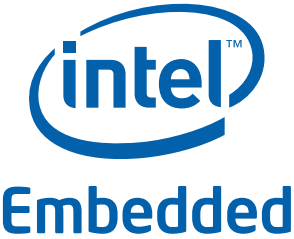

# Intel® Platform Controller Hub EG20T

## Inter Integrated Circuit ($I^2C$*) Driver for Windows* Programmer's Guide

*February 2011*

Order Number: 324258-002US

# Contents



## Tables

# Revision History

| Date | Revision | Description |
|---|---|---|
| February 2011 | 002 | Updated Section 2.0, “Operating System (OS) Support” on page 6<br>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 Inter Integrated Circuit ($I^2C$*) driver for Windows*. This includes the information about the interfaces exposed by the driver and how to use those interfaces to drive the $I^2C$* hardware.

$I^2C$* (Inter-Integrated Circuit) is a multi-master serial computer bus that is used to attach low-speed peripherals to a motherboard or embedded system. $I^2C$* uses only two bidirectional open-drain lines, Serial Data Line (SDA) and Serial Clock (SCL), pulled up with resistors.

# 2.0 Operating System (OS) Support

The I²C* 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 I²C* Driver API Details

This section provides information about the interfaces exposed by the I²C* 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 I²C* hardware to work properly.

## 4.1 Features

The I²C* Driver supports:

- Setting different configurations for I²C* hardware
- Multi master devices only
- Setting I²C* slave device address
- Mode Select – fast mode (400 kbps) or standard mode (100 kbps) only
- Buffer Mode select -- using 32-byte hard buffering or not
- EEPROM Software Reset Mode select –- using EEPROM Software Reset Mode or not
- Reading I²C* hardware and bus status items
- I²C* Bus Master byte/multi-byte read transactions
- I²C* Bus Master byte/multi-byte write transactions

## 4.2 Interface Details

Table 1 lists IOCTLs supported by the driver.

**Table 1. Supported IOCTLs**

| No | IOCTL | Remarks |
|---|---|---|
| 1 | IOCTL_I2C_CONFIG | This IOCTL is used for configuration information |
| 2 | IOCTL_I2C_ENABLE_INT | This IOCTL is used to enable the interrupts |
| 3 | IOCTL_I2C_DISABLE_INT | This IOCTL is used to disable the interrupts |
| 4 | IOCTL_I2C_READ | This IOCTL is used to read information from the devices connected to the I²C* hardware |
| 5 | IOCTL_I2C_WRITE | This IOCTLS is used to write information to the devices connected to the I²C* hardware |
| 6 | IOCTL_I2C_RESET | This IOCTL is used to reset the device |
| 7 | IOCTL_I2C_COMPOUND_RDWR | This IOCTL is used for compound mode read and write |

## 4.3 IOCTL Usage Details

This section assumes a single client model, in which there is a single application-level program configuring the I²C* interface and initiating I/O operations. The following files contain the details of the IOCTLs and data structures used:

- ioh_I2C_ioctls.h – contains IOCTL definitions
- ioh_I2C_common.h – data structures and other variables used by the IOCTLs

### 4.3.1 IOCTL_I2C_CONFIG

Before performing any operation, the interface must be initialized and configured. This IOCTL is used to initialize and configure the I²C* interface. The prerequisite is that the device must be installed and opened using the Win32 API CreateFile.

```
IOH_I2C_MSG I2CConfig;

        UCHAR msgbuf[1];

        msgbuf[0] = EEPROM_RST_PATTERN1;

        I2CConfig.SlaveAddress = EEPROM_ADDRESS;

        I2CConfig.MsgData = msgbuf;

        I2CConfig.MsgLength = 1;

        I2CConfig.Flags = IOH_EEPROM_SW_RST_MODE_ENABLE;

DeviceIoControl(hHandle,

                IOCTL_I2C_CONFIG,

                &I2CConfig,

                sizeof(I2Cconfig),

                NULL,

                0,

                &dwSize,

                NULL);
```

### 4.3.2 IOCTL_I2C_ENABLE_INT

This enables the interrupts of the I²C* interface.

```
DeviceIoControl(hHandle,

                IOCTL_I2C_ENABLE_INT,

                NULL,

                0,

                NULL,

                0,

                &dwSize,

                NULL);
```

### 4.3.3 IOCTL_I2C_DISABLE_INT

This disables the interrupts of the I²C* interface.

```
DeviceIoControl(hHandle,

                IOCTL_I2C_DISABLE_INT,
```

```
NULL,
0,
NULL,
0,
&dwSize,
NULL);
```

### 4.3.4 IOCTL_I2C_READ

This IOCTL is used to read information from the devices connected to the I²C* hardware.

Refer to Section 5.3 for usage details.

### 4.3.5 IOCTL_I2C_WRITE

This IOCTL is used to write information to the devices connected to the I²C* hardware.

Refer to Section 5.3 for usage details.

## 4.4 Structures and Macros

This section provides the details on the structures and macros used by interfaces exposed by I²C* driver. All the structures and macros used by the interfaces are defined in ioh_i2c_common.h.

### 4.4.1 Structures

#### 4.4.1.1 IOH_I2C_MSG

This structure holds the user supplied configuration information for configuring the I²C* controller.

**Table 2. IOH_I2C_MSG structure**

| Name | Description |
|---|---|
| ULONG SlaveAddress | Slave address |
| ULONG Flags | Flags to convey multiple status information such as 10 bit chip addressing, read or write from slave to master, normal or buffer mode |
| ULONG MsgLength | Message length |
| PUCHAR MsgData | Message Data |

#### 4.4.1.2 IOH_I2C_COMPOUND_DATA

This structure is used for sending compound data messages.

**Table 3. IOH_I2C_COMPOUND_DATA structure**

| Name | Description |
|---|---|
| PIOH_I2C_MSG msgs | This is a pointer to structure IOH_I2C_MSG. Refer to section "IOH_I2C_MSG" on page 10. |
| ULONG dwNumMsgs | Number of messages to be included in the compound data message. |

### 4.4.2 Macros

Table 4 contains the macro definitions exposed by the I²C* driver.

**Table 4. Macros**

| Name | Description |
|---|---|
| I2C_M_TEN | This defines the chip address as 10 bit address. |
| I2C_M_RD | This defines the I²C* read mode. |
| I2C_M_WR | This defines the I²C* write mode. |
| IOH_NORMAL_MODE_ENABLE | This defines the I²C* Normal mode. |
| IOH_BUFFER_MODE_ENABLE | This defines the I²C* Buffer mode. |
| IOH_EEPROM_SW_RST_MODE_ENABLE | This defines the I²C* reset mode. |

## 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 will be 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 explains the basic procedure to use the I²C* driver from a user mode application. All operations are performed through the IOCTLs that are exposed by the I²C* driver. Refer to Section 4.3 for details on the IOCTLs. The steps involved in accessing the I²C* 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

The I²C* 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 I2C driver registers the following interface.

| No | Interface Name |
|---|---|
| 1 | GUID_DEVINTERFACE_IOHI2C |

This is defined ioh_i2c_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 I²C* driver:

- IOCTL_I2C_CONFIG

*DeviceIoControl* Win32 API is used for sending information to the I²C* driver.

```
IOH_I2C_MSG I2CConfig;

UCHAR msgbuf[1];


msgbuf[0] = EEPROM_RST_PATTERN1;

I2CConfig.SlaveAddress = EEPROM_ADDRESS;
```

```
I2CConfig.MsgData = msgbuf;

I2CConfig.MsgLength = 1;

I2CConfig.Flags = IOH_EEPROM_SW_RST_MODE_ENABLE;


bRet = DeviceIoControl(hHandle,
        IOCTL_I2C_CONFIG,
        &I2CConfig,
        sizeof(I2CConfig),
        NULL,
        0,
        &dwSize,
        NULL);
```

- IOCTL_I2C_ENABLE_INT

This IOCTL enables all the interrupts.

```
bRet = DeviceIoControl( hDevice,
        IOCTL_I2C_ENABLE_INT,
        NULL,
        0,
        NULL,
        0,
        &dwRet,
        NULL);
```

- IOCTL_I2C_DISABLE_INT

This IOCTL disables all the interrupts.

```
bRet = DeviceIoControl( hDevice,
        IOCTL_I2C_DISABLE_INT,
        NULL,
        0,
        NULL,
        0,
        &dwRet,
        NULL);
```

- IOCTL_I2C_RESET

This IOCTL resets the device to a known initial state.

```
bRet = DeviceIoControl( hDevice,
        IOCTL_I2C_RESET,
        NULL,
        0,
        NULL,
        0,
        &dwRet,
        NULL);
```

## 5.3 Read / Write Operation

IOCTL_I2C_READ and IOCTL_I2C_WRITE are used for read and write operations respectively.

- IOCTL_I2C_READ

This IOCTL is used to read the data from the I2C* device.

```
IOH_I2C_MSG I2CConfig;
UCHAR msgbuf[1];
UCHAR ucData[30];


DeviceIoControl(hHandle,
                IOCTL_I2C_ENABLE_INT,
                NULL,
                0,
                NULL,
                0,
                &dwSize,
                NULL);


msgbuf[0] = NORMAL_MODE;
I2CConfig.SlaveAddress = EEPROM_ADDRESS;
I2CConfig.MsgData = msgbuf;
I2CConfig.MsgLength = 1;
I2CConfig.Flags = 0;


DeviceIoControl(hHandle,
                IOCTL_I2C_CONFIG,
```

```
                &I2CConfig,
                sizeof(I2CConfig),
                NULL,
                0,
                &dwSize,
                NULL);

DeviceIoControl(hHandle,
                IOCTL_I2C_READ,
                &ucData,
                sizeof(ucData),
                &ucData,
                sizeof(ucData),
                &dwSize,
                NULL);

DeviceIoControl(hHandle,
                IOCTL_I2C_DISABLE_INT,
                NULL,
                0,
                NULL,
                0,
                &dwSize,
                NULL);
```

- IOCTL_I2C_WRITE

This IOCTL is used to write the data to the device.

```
IOH_I2C_MSG I2CConfig;
UCHAR msgbuf[1];
UCHAR ucData[30];

DeviceIoControl(hHandle,
                IOCTL_I2C_ENABLE_INT,
                NULL,
                0,
```

```
                NULL,
                0,
                &dwSize,
                NULL);

msgbuf[0] = NORMAL_MODE;
I2CConfig.SlaveAddress = EEPROM_ADDRESS;
I2CConfig.MsgData = msgbuf;
I2CConfig.MsgLength = 1;
I2CConfig.Flags = 0;

DeviceIoControl(hHandle,
                IOCTL_I2C_CONFIG,
                &I2CConfig,
                sizeof(I2CConfig),
                NULL,
                0,
                &dwSize,
                NULL);
DeviceIoControl(hHandle,
                IOCTL_I2C_READ,
                &ucData,
                sizeof(ucData),
                &ucData,
                sizeof(ucData),
                &dwSize,
                NULL);

DeviceIoControl(hHandle,
                IOCTL_I2C_DISABLE_INT,
                NULL,
                0,
                NULL,
                0,
```

```
                    &dwSize,

                    NULL);
```

## 5.4 Close the Device

Once all operations related to the I²C* driver are finished the device handle must free the application by calling the Win32 API CloseHandle.

```
CloseHandle(hHandle);
```