

Intel[®] Platform Controller Hub EG20T

General Purpose Input Output (GPIO) Driver for Windows*
Programmer's Guide

February 2011



INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL® PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER, AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

UNLESS OTHERWISE AGREED IN WRITING BY INTEL, THE INTEL PRODUCTS ARE NOT DESIGNED NOR INTENDED FOR ANY APPLICATION IN WHICH THE FAILURE OF THE INTEL PRODUCT COULD CREATE A SITUATION WHERE PERSONAL INJURY OR DEATH MAY OCCUR.

Intel may make changes to specifications and product descriptions at any time, without notice. Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined." Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The information here is subject to change without notice. Do not finalize a design with this information.

The products described in this document may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Contact your local Intel sales office or your distributor to obtain the latest specifications and before placing your product order.

Copies of documents which have an order number and are referenced in this document, or other Intel literature, may be obtained by calling 1-800-548-4725, or go to: http://www.intel.com/#/en_US_01.

Any software source code reprinted in this document is furnished under a software license and may only be used or copied in accordance with the terms of that license.

Intel processor numbers are not a measure of performance. Processor numbers differentiate features within each processor family, not across different processor families. Go to: <http://www.intel.com/products/processor%5Fnumber/> for details.

α Intel® Hyper-Threading Technology requires a computer system with a processor supporting Intel® HT Technology and an Intel® HT Technology-enabled chipset, BIOS and operating system. Performance will vary depending on the specific hardware and software you use. For more information including details on which processors support Intel® HT Technology, see http://www.intel.com/products/ht/hyperthreading_more.htm.

β Intel® High Definition Audio requires a system with an appropriate Intel® chipset and a motherboard with an appropriate CODEC and the necessary drivers installed. System sound quality will vary depending on actual implementation, controller, CODEC, drivers and speakers. For more information about Intel® HD audio, refer to <http://www.intel.com/>.

γ 64-bit computing on Intel® architecture requires a computer system with a processor, chipset, BIOS, operating system, device drivers and applications enabled for Intel® 64 architecture. Performance will vary depending on your hardware and software configurations. Consult with your system vendor for more information.

δ Intel® Virtualization Technology requires a computer system with an enabled Intel® processor, BIOS, virtual machine monitor (VMM) and, for some uses, certain computer system software enabled for it. Functionality, performance or other benefits will vary depending on hardware and software configurations and may require a BIOS update. Software applications may not be compatible with all operating systems. Please check with your application vendor.

ε The original equipment manufacturer must provide Intel® Trusted Platform Module (Intel® TPM) functionality, which requires an Intel® TPM-supported BIOS. Intel® TPM functionality must be initialized and may not be available in all countries.

θ For Enhanced Intel SpeedStep® Technology, see the [Processor Spec Finder](#) or contact your Intel representative for more information.

I²C* is a two-wire communications bus/protocol developed by Philips. SMBus is a subset of the I²C* bus/protocol and was developed by Intel. Implementations of the I²C* bus/protocol may require licenses from various entities, including Philips Electronics N.V. and North American Philips Corporation.

BunnyPeople, Celeron, Celeron Inside, Centrino, Centrino Inside, Core Inside, i960, Intel, the Intel logo, Intel AppUp, Intel Atom, Intel Atom Inside, Intel Core, Intel Inside, the Intel Inside logo, Intel NetBurst, Intel NetMerge, Intel NetStructure, Intel SingleDriver, Intel SpeedStep, Intel Sponsors of Tomorrow., the Intel Sponsors of Tomorrow. logo, Intel StrataFlash, Intel Viiv, Intel vPro, Intel XScale, InTru, the InTru logo, InTru soundmark, Itanium, Itanium Inside, MCS, MMX, Moblin, Pentium, Pentium Inside, skool, the skool logo, Sound Mark, The Journey Inside, vPro Inside, VTune, Xeon, and Xeon Inside are trademarks of Intel Corporation in the U.S. and other countries.

*Other names and brands may be claimed as the property of others.

Copyright © 2011, Intel Corporation and/or its suppliers and licensors. All rights reserved.



Contents

| | | |
|------------|--|----|
| 1.0 | Introduction | 5 |
| 2.0 | Operating System (OS) Support | 6 |
| 3.0 | Dependencies | 7 |
| 4.0 | GPIO Driver API Details | 8 |
| 4.1 | Features | 8 |
| 4.2 | Interface Details | 8 |
| 4.3 | IOCTL Usage Details | 8 |
| 4.3.1 | IOCTL_GPIO_ENABLE_INT | 8 |
| 4.3.2 | IOCTL_GPIO_DISABLE_INT | 9 |
| 4.3.3 | IOCTL_GPIO_DIRECTION | 9 |
| 4.3.4 | IOCTL_GPIO_READ | 10 |
| 4.3.5 | IOCTL_GPIO_WRITE | 11 |
| 4.3.6 | IOCTL_GPIO_NOTIFY | 11 |
| 4.4 | Structures, Enumerations and Macros | 12 |
| 4.4.1 | Structures | 13 |
| 4.4.1.1 | ioh_gpio_reqt | 13 |
| 4.4.2 | Enumerations | 13 |
| 4.4.2.1 | GPIO_SET_MODE | 13 |
| 4.4.2.2 | GPIO_SET_DIR | 13 |
| 4.4.2.3 | GPIO_SET_VALUE | 13 |
| 4.4.3 | Macros | 13 |
| 4.5 | Error Handling | 14 |
| 4.6 | Inter-IOCTL dependencies | 14 |
| 5.0 | Programming Guide | 15 |
| 5.1 | Opening the Device | 15 |
| 5.1.1 | Using GUID Interface Exposed by the Driver | 15 |
| 5.2 | Driver Configuration | 15 |
| 5.3 | Read/Write Operation | 16 |
| 5.4 | Close the Device | 17 |

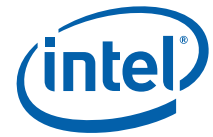
Tables

| | | |
|---|----------------------------|----|
| 1 | Supported IOCTLs | 8 |
| 2 | ioh_gpio_reqt structure | 13 |
| 3 | GPIO_SET_MODE enumeration | 13 |
| 4 | GPIO_SET_DIR enumeration | 13 |
| 5 | GPIO_SET_VALUE enumeration | 13 |
| 6 | Macros | 14 |



Revision History

| Date | Revision | Description |
|----------------|----------|--|
| February 2010 | 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 15 |
| September 2010 | 001 | Initial release |



1.0 Introduction

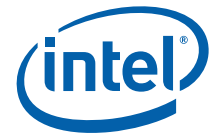
This document provides the programming details of the General Purpose Input Output (GPIO) driver for Windows*. This includes the information about the interfaces exposed by the driver and how to use those interfaces to drive the GPIO hardware.



2.0 Operating System (OS) Support

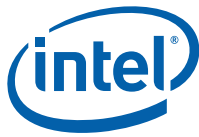
The GPIO driver is supported by the following operating systems:

| 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 GPIO Driver API Details

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

4.1 Features

The GPIO Driver supports:

- Setting of different configurations for GPIO hardware
- Writing data to GPIO hardware
- Reading data from GPIO hardware
- Setting the direction of GPIO hardware
- Enabling/disabling the interrupts from GPIO hardware
- User-mode program notification by overlapping I/O when an interrupt occurs in the GPIO hardware

4.2 Interface Details

Table 1 lists IOCTLs supported by the driver.

Table 1. Supported IOCTLs

| No | IOCTL | Remarks |
|----|------------------------|---|
| 1 | IOCTL_GPIO_ENABLE_INT | Enable the interrupt to selected pins of given GPIO port; set the interrupt mask and mode |
| 2 | IOCTL_GPIO_DISABLE_INT | Disable the interruption to selected pins of given GPIO port |
| 3 | IOCTL_GPIO_READ | Read the data of selected pins of given GPIO port |
| 4 | IOCTL_GPIO_WRITE | Write the data to selected pins of given GPIO port |
| 5 | IOCTL_GPIO_DIRECTION | Set the direction of the selected pins of given GPIO port |
| 6 | IOCTL_GPIO_NOTIFY | Read the data of selected pins of given GPIO port, when the status of the pin is changed |

4.3 IOCTL Usage Details

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

- `ioh_gpio_ioctl.h` – contains IOCTL definitions
- `ioh_gpio_common.h` – data structures and other variables used by the IOCTLs

4.3.1 IOCTL_GPIO_ENABLE_INT

This IOCTL is called to enable the interrupt for the selected pins. The prerequisite for this is that the device must be installed and opened using the Win32 API CreateFile.

```
ioh_gpio_reqt gpio_reqt;
```




```
gpio_reqt.pins = 0xff;
gpio_reqt.mode = GPIO_IN;
gpio_reqt.enable = 1;

DeviceIoControl(hHandle,
                IOCTL_GPIO_ENABLE_INT,
                &&gpio_reqt,
                sizeof(gpio_reqt),
                NULL,
                0,
                &dwSize,
                NULL);
```

4.3.2 IOCTL_GPIO_DISABLE_INT

This disables the interrupts of GPIO pins.

```
ioh_gpio_reqt gpio_reqt;

gpio_reqt.pins = 0xff;
gpio_reqt.mode = GPIO_IN;
gpio_reqt.enable = 1;

DeviceIoControl(hHandle,
                IOCTL_GPIO_DISABLE_INT,
                &&gpio_reqt,
                sizeof(gpio_reqt),
                NULL,
                0,
                &dwSize,
                NULL);
```

4.3.3 IOCTL_GPIO_DIRECTION

This sets the direction of the selected pins of given GPIO port.

```
ioh_gpio_reqt gpio_reqt;
```



```
gpio_reqt.pins = 0xff;
gpio_reqt.mode = GPIO_IN;
gpio_reqt.enable = 1;
DeviceIoControl(hHandle,
                IOCTL_GPIO_DIRECTION,
                &&gpio_reqt,
                sizeof(gpio_reqt),
                NULL,
                0,
                &dwSize,
                NULL);
```

4.3.4 IOCTL_GPIO_READ

Read the data of selected pins of given GPIO port.

```
ioh_gpio_reqt gpio_reqt,out_buffer;
```

```
gpio_reqt.pins = 0xff;
gpio_reqt.mode = GPIO_IN;
gpio_reqt.enable = 1;

DeviceIoControl(hHandle,
                IOCTL_GPIO_DIRECTION,
                &&gpio_reqt,
                sizeof(gpio_reqt),
                NULL,
                0,
                &dwSize,
                NULL);
```

```
DeviceIoControl(hHandle,
                IOCTL_GPIO_READ,
                &&gpio_reqt,
                sizeof(gpio_reqt),
                &out_buffer,
```



```

        sizeof(out_buffer),
        &dwSize,
        NULL);

```

4.3.5 IOCTL_GPIO_WRITE

The write operation writes to the selected pins of the GPIO.

```

ioh_gpio_reqt gpio_reqt,out_buffer;

```

```

gpio_reqt.pins = 0xff;
gpio_reqt.mode = GPIO_OUT;
gpio_reqt.enable = 1;

```

```

DeviceIoControl(hHandle,
                IOCTL_GPIO_DIRECTION,
                &gpio_reqt,
                sizeof(gpio_reqt),
                NULL,
                0,
                &dwSize,
                NULL);

```

```

DeviceIoControl(hHandle,
                IOCTL_GPIO_WRITE,
                &gpio_reqt,
                sizeof(gpio_reqt),
                &out_buffer,
                sizeof(out_buffer),
                &dwSize, NULL);

```

4.3.6 IOCTL_GPIO_NOTIFY

Read the data of selected pins of given GPIO port, when the status of the pin is changed.

```

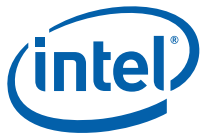
ioh_gpio_reqt gpio_reqt,out_buffer;

```

```

gpio_reqt.pins = 0xff;

```



```
gpio_reqt.mode = GPIO_IN;
gpio_reqt.enable = 1;

DeviceIoControl(hHandle,
                IOCTL_GPIO_ENABLE_INT,
                &&gpio_reqt,
                sizeof(gpio_reqt),
                NULL,
                0,
                &dwSize,
                NULL);

DeviceIoControl(hHandle,
                IOCTL_GPIO_DIRECTION,
                &&gpio_reqt,
                sizeof(gpio_reqt),
                NULL,
                0,
                &dwSize,
                NULL);

DeviceIoControl(hHandle,
                IOCTL_GPIO_NOTIFY,
                &&gpio_reqt,
                sizeof(gpio_reqt),
                &out_buffer,
                sizeof(out_buffer),
                &dwSize,
                NULL);
```

4.4 Structures, Enumerations and Macros

This section provides the details on the structures, enumerations and macros used by interfaces exposed by the GPIO driver. All the structures, enumerations and macros used by the interfaces are defined in `ioh_gpio_common.h`.



4.4.1 Structures

4.4.1.1 ioh_gpio_reqt

This structure is used for preserving information related to the GPIO request.

Table 2. ioh_gpio_reqt structure

| Name | Description |
|--------------|--|
| ULONG port | Set the port number |
| ULONG pins | Data in the case of read and write and in other cases it indicates applicable pins |
| UINT64 mode | Set the interrupt mode and direction mode |
| ULONG enable | Set 1 for enabling the interrupt and 0 for disabling the interrupt |

4.4.2 Enumerations

4.4.2.1 GPIO_SET_MODE

This enum is used for preserving information related to the imode.

Table 3. GPIO_SET_MODE enumeration

| Name | Description |
|-------------|-------------------|
| DISABLE_SEL | Disable selection |
| ENABLE_SEL | Enable selection |
| NEITHER | Select neither |

4.4.2.2 GPIO_SET_DIR

This enum is used for preserving information related to the direction.

Table 4. GPIO_SET_DIR enumeration

| Name | Description |
|------------|-------------------------|
| INPUT_SEL | Set direction as input |
| OUTPUT_SEL | Set direction as output |
| NEITHER | Set neither direction |

4.4.2.3 GPIO_SET_VALUE

This enum is used for setting a GPIO pin to active low or active high.

Table 5. GPIO_SET_VALUE enumeration

| Name | Description |
|-----------|----------------------|
| GPIO_LOW | Set GPIO active low |
| GPIO_HIGH | Set GPIO active high |

4.4.3 Macros

These are the Macro definitions exposed by the GPIO driver.



Table 6. Macros

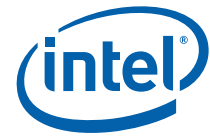
| Name | Description |
|-------------------|---------------------------------|
| IOH_GPIO_ALL_PINS | This is used to select all pins |

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 GPIO driver from a user mode application. All operations are through the IOCTLs exposed by the GPIO 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

The GPIO 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 Intel® PCH EG20T GPIO driver registers the following interface.

| No | Interface Name |
|----|---------------------------|
| 1 | GUID_DEVINTERFACE_IOHGPIO |

This is defined `ioh_gpio_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/ff549791.aspx>

5.2 Driver Configuration

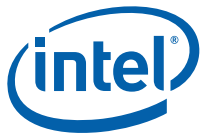
The following IOCTLs are used to initialize and configure the settings for the GPIO driver:

- IOCTL_GPIO_DIRECTION
- IOCTL_GPIO_ENABLE_INT
- IOCTL_GPIO_DISABLE_INT

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

```
ioh_gpio_reqt gpio_reqt;
```

```
gpio_reqt.pins = 0xff;
```



```
gpio_reqt.mode = GPIO_IN;
gpio_reqt.enable = 1;

DeviceIoControl(hHandle,
                IOCTL_GPIO_ENABLE_INT,
                &&gpio_reqt,
                sizeof(gpio_reqt),
                NULL,
                0,
                &dwSize,
                NULL);
```

5.3 Read/Write Operation

IOCTL_GPIO_READ and IOCTL_GPIO_WRITE are used for read and write operations respectively.

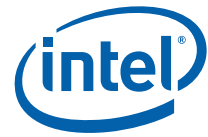
- Read Operation:

```
ioh_gpio_reqt gpio_reqt,out_buffer;
```

```
gpio_reqt.pins = 0xff;
gpio_reqt.mode = GPIO_IN;
gpio_reqt.enable = 1;
```

```
DeviceIoControl(hHandle,
                IOCTL_GPIO_DIRECTION,
                &&gpio_reqt,
                sizeof(gpio_reqt),
                NULL,
                0,
                &dwSize,
                NULL);
```

```
DeviceIoControl(hHandle,
                IOCTL_GPIO_READ,
                &&gpio_reqt,
                sizeof(gpio_reqt),
```

```

        &out_buffer,
        sizeof(out_buffer),
        &dwSize,
        NULL);

```

- Write Operation:

```
ioh_gpio_reqt gpio_reqt, out_buffer;
```

```

gpio_reqt.pins = 0xff;
gpio_reqt.mode = GPIO_OUT;
gpio_reqt.enable = 1;
DeviceIoControl(hHandle,
                IOCTL_GPIO_DIRECTION,
                &gpio_reqt,
                sizeof(gpio_reqt),
                NULL,
                0,
                &dwSize,
                NULL);

```

```

DeviceIoControl(hHandle,
                IOCTL_GPIO_WRITE,
                &gpio_reqt,
                sizeof(gpio_reqt),
                &out_buffer,
                sizeof(out_buffer),
                &dwSize,
                NULL);

```

5.4 Close the Device

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

```
CloseHandle(hHandle);
```