Intel® Chipset General Purpose I/O (GPIO) Software Configuration

Application Note

March 2010

Document Number: 322174-002

Contents

Revision History

Document Number	Revision Number	Description	Revision Date
322174	002	Title change. Was: *General Purpose I/O (GPIO) Use in Software Application Note*	March 2010
	001	Initial release.	June 2009

§

1 Introduction

This document pertains to later Intel® Architecture (IA) South Bridge devices, and provides information on General Purpose Input/Output (GPIO) usage in software. The document also covers the registers associated with the GPIO hardware pins and presents code snippets for use in driver development.

For further related information, please see the GPIO sections of the *Intel® I/O Controller Hub 9 (ICH9) Family Datasheet*, the *Intel® EP80579 Integrated Processor Product Line Datasheet*, and the *Intel® 5 Series Chipset and Intel® 3400 Series Chipset Datasheet*.

1.1 Terminology

Term	Description
GPE	General Purpose Event
GPIO	General Purpose Input/Output
I/O	Input/Output
IA	Intel Architecture
ICH	I/O Control Hub
LPC	Low Pin Count
PCH	Platform Control Hub

1.2 Reference Documents

Document	Document No./Link
Intel® I/O Controller Hub 9 (ICH9) Family Datasheet	316972; http://www.intel.com/Assets/PDF/datasheet/316972.pdf
Intel® EP80579 Integrated Processor Product Line Datasheet	320066; http://download.intel.com/design/intarch/ep80579/320066.pdf

Document	Document No./Link
Intel® 5 Series Chipset and Intel® 3400 Series Chipset Datasheet	322169; http://www.intel.com/Assets/PDF/datasheet/322169.pdf

2 GPIO Usage

2.1 GPIO Registers and Pins

In the GPIO registers, each bit corresponds to a pin. The pins are defined in a pin function or summary table. If there are more than 32 GPIO pins, they are spread out over multiple registers. For example, if there are 64 GPIO pins, then there are two sets of GPIO registers. GPIO pins 0 through 31 are controlled by the first set and GPIO pins 32 through 63 are controlled by the second set. To control and use, for example, GPIO pin 26, software manipulates bit 26 in each of the following:

- a use select register
- an input/output select register
- a level register
- an output blink enable register
- an input signal invert register

2.1.1 Input Configuration of GPIO Pin-*n*

1. Set bit-*n* to select GPIO rather than an alternative in the GPIO_USE_SEL register.
2. Set bit-*n* to select input in the GP_IO_SEL register.
3. Read the state of the input signal by reading bit-*n* in the GP_LVL register. (Writes to bit in this register have no effect since it is configured as an input.)
4. Disable or enable input inversion by clearing or setting bit-*n* in the GPI_INV register. (The input is inverted for the General Purpose Event logic only. GP_LVL is unaffected.)

2.1.2 Output Configuration of GPIO Pin-*n*

1. Set bit-*n* to select GPIO rather than an alternative in the GPIO_USE_SEL register.
2. Clear bit-*n* to select output in the GP_IO_SEL register.
3. Write to bit-*n* in the GP_LVL register to drive the output signal. (A one drives the pin high and a zero drives the pin low.)
4. Disable or enable output blink by clearing or setting bit-*n* in the GPO_BLINK register.

2.2 FreeBSD* Code Examples

2.2.1 Find Device and Get GPIO Base Address

Find the LPC interface.

```
/* vendor and device IDs of LPC device */
#define LPC_VENDOR_ID 0x8086
#define LPC_DEVICE_ID 0x5031

/* offset into low pin count (LPC) config space of the GPIO base
address */
#define GPIO_BAR_OFFSET 0x48
#define DWORD              4

/* get the device struct for the LPC(low pin count) device */
g_drvr_data.pdev = pci_find_device(LPC_VENDOR_ID, LPC_DEVICE_ID);
```

Get the base address for accessing the GPIO registers. Use this base plus the register offsets to access the GPIO registers.

```
/* Get base address from the LPC configuration space. */
g_drvr_data.mem_base = pci_read_config(g_drvr_data.pdev,
                          GPIO_BAR_OFFSET, DWORD);

/* Clear all but address bits */
g_drvr_data.mem_base &= 0x0000ff80;
```

2.2.2 GPIO_USE_SEL Register

Select GPIO or alternative.

```
/* offsets for first set of gpio registers */
#define  GPIO_USE_SEL   0x00
#define  GP_IO_SEL      0x04
#define  GP_LVL         0x0c
#define  GPO_BLINK      0x18
#define  GPI_INV        0x2c

/* read the register first */
ULONG    bitstr, pin_num;

/* get the register contents */
bitstr = inl(g_drvr_data.mem_base + GPIO_USE_SEL);

if ( cmd == IOCTL_SET_TO_ALTERNATIVE )
{
    /* clear bit to use pin for alternative function */
    bitstr &= ~(1 << pin_num);
}
else
{
    /* set bit to use pin for gpio */
    bitstr |= (1 << pin_num);
}

/* write the modified string back to the gpio register */
outl(g_drvr_data.mem_base + GPIO_USE_SEL, bitstr);
```

2.2.3 GP_IO_SEL Register

Select input or output.

```
/* get the register contents */
bitstr = inl(g_drvr_data.mem_base + GP_IO_SEL);

if ( cmd == IOCTL_SET_AS_OUTPUT )
{
    /* clear bit to use pin as output */
    bitstr &= ~(1 << pin_num);
}
else
{
    /* set bit to use pin as input */
    bitstr |= (1 << pin_num);
}

/* write the modified string back to the gpio register */
outl(g_drvr_data.mem_base + GP_IO_SEL, bitstr);
```

2.2.4 GP_LVL Register

Set the output level.

```
/* get the register contents */
bitstr = inl(g_drvr_data.mem_base + GP_LVL);

if ( cmd == IOCTL_SET_LOW )
{
    /* write a 0 to output pin */
    bitstr &= ~(1 << pin_num);
}
else
{
    /* write a 1 to output pin */
    bitstr |= (1 << pin_num);
}

/* write the modified string back to the gpio register */
outl(g_drvr_data.mem_base + GP_LVL, bitstr);
```

2.2.5 GPO_BLINK Register

Enable or disable output blink.

```
/* get the register contents */
bitstr = inl(g_drvr_data.mem_base + GPO_BLINK);

if ( cmd == IOCTL_STOP_BLINK )
{
    /* set pin to non-blink mode */
    bitstr &= ~(1 << pin_num);
}
else
{
    /* set pin to blink mode */
    bitstr |= (1 << pin_num);
}

/* write the modified string back to the gpio register */
outl(g_drvr_data.mem_base + GPO_BLINK, bitstr);
```

2.2.6 GPI_INV Register

Enable or disable input inversion.

```
/* get the register contents */
bitstr = inl(g_drvr_data.mem_base + GPI_INV);

if ( cmd == IOCTL_NONINVERTED_INPUT )
{
    /* set pin to noninverted mode */
    bitstr &= ~(1 << pin_num);
}
else
{
    /* set pin to inverted mode */
    bitstr |= (1 << pin_num);
}

/* write the modified string back to the gpio register */
outl(g_drvr_data.mem_base + GPI_INV, bitstr);
```

2.3 Linux* Code Examples

2.3.1 Module Init

The init function should include something like the following:

```
/* vendor and device IDs of LPC device */
#define LPC_VENDOR_ID 0x8086
#define LPC_DEVICE_ID 0x5031
```

```
/* offset into low pin count (LPC) config space of the GPIO base
address */
#define GPIO_BAR_OFFSET 0x48

/* GPIO register memory size in bytes */
#define GPIO_MEM_SIZE 64

#define DRIVERNAME "gpio_ref"

typedef struct drvr_data
{
   gpio_regs_t regs;
   dev_t devnum;
   struct cdev cdev;
   void *mem_virt;
   unsigned int mem_base;
   unsigned int mem_resrvd;
} drvr_data_t;

dev_t devno;
struct pci_dev *pdev = NULL;

/* request and reserve a device number */
ret = alloc_chrdev_region(&g_drvr_data.devnum, 0, 1, DRIVERNAME);

/* init cdev struct for adding device to kernel */
cdev_init(&g_drvr_data.cdev, &file_ops);
g_drvr_data.cdev.owner = THIS_MODULE;
g_drvr_data.cdev.ops = &file_ops;

devno = MKDEV(MAJOR(g_drvr_data.devnum), 0);

/* Get dev struct for the LPC device. The GPIO BAR is located in
the LPC device config space */
pdev = pci_get_device(LPC_VENDOR_ID, LPC_DEVICE_ID, NULL);

/* Get base address from the LPC configuration space. */
pci_read_config_dword(pdev, GPIO_BAR_OFFSET,
                        &(g_drvr_data.mem_base));

/* Clear all but address bits */
g_drvr_data.mem_base &= 0x0000ff80;

/* release reference to device */
pci_dev_put(pdev);

/* obtain exclusive access to GPIO memory space */
request_region(g_drvr_data.mem_base, GPIO_MEM_SIZE, DRIVERNAME);
```

2.3.2 Module Exit

The exit function should include something like the following:

```
/* remove cdev struct from system */
```

```
cdev_del(&g_drvr_data.cdev);

/* unregister driver module */
unregister_chrdev_region(g_drvr_data.devnum, 1);

/* release the reserved IO memory space */
if ( g_drvr_data.mem_resrvd )
{
    release_region(g_drvr_data.mem_base, GPIO_MEM_SIZE);
}
```

2.3.3 GPIO_USE_SEL Register

Select GPIO or alternative.

```
/* offsets for first set of gpio registers */
#define  GPIO_USE_SEL   0x00
#define  GP_IO_SEL      0x04
#define  GP_LVL         0x0c
#define  GPO_BLINK      0x18
#define  GPI_INV        0x2c

/* read the register first */
ULONG    bitstr, pin_num;

/* get the register contents */
bitstr = inl(g_drvr_data.mem_base + GPIO_USE_SEL);

if ( cmd == IOCTL_SET_TO_ALTERNATIVE )
{
    /* clear bit to use pin for alternative function */
    clear_bit(pin_num, (void*)&bitstr);
}
else
{
    /* set bit to use pin for gpio */
    set_bit(pin_num, (void*)&bitstr);
}

/* write the modified string back to the gpio register */
outl(bitstr, g_drvr_data.mem_base + GPIO_USE_SEL);
```

2.3.4 GP_IO_SEL Register

Select input or output.

```
/* get the register contents */
bitstr = inl(g_drvr_data.mem_base + GP_IO_SEL);

if ( cmd == IOCTL_SET_AS_OUTPUT )
{
    /* clear bit to use pin as output */
    clear_bit(pin_num, (void*)&bitstr);
}
else
{
    /* set bit to use pin as input */
    set_bit(pin_num, (void*)&bitstr);
}

/* write the modified string back to the gpio register */
outl(bitstr, g_drvr_data.mem_base + GP_IO_SEL);
```

2.3.5 GP_LVL Register

Set the output level.

```
/* get the register contents */
bitstr = inl(g_drvr_data.mem_base + GP_LVL);

if ( cmd == IOCTL_SET_LOW )
{
    /* write a 0 to output pin */
    clear_bit(pin_num, (void*)&bitstr);
}
else
{
    /* write a 1 to output pin */
    set_bit(pin_num, (void*)&bitstr);
}

/* write the modified string back to the gpio register */
outl(bitstr, g_drvr_data.mem_base + GP_LVL);
```

2.3.6 GPO_BLINK Register

Enable or disable output blink.

```
/* get the register contents */
bitstr = inl(g_drvr_data.mem_base + GPO_BLINK);

if ( cmd == IOCTL_STOP_BLINK )
{
    /* set pin to non-blink mode */
    clear_bit(pin_num, (void*)&bitstr);
}
else
{
    /* set pin to blink mode */
    set_bit(pin_num, (void*)&bitstr);
}

/* write the modified string back to the gpio register */
outl(bitstr, g_drvr_data.mem_base + GPO_BLINK);
```

2.3.7 GPI_INV Register

Enable or disable input inversion.

```
/* get the register contents */
bitstr = inl(g_drvr_data.mem_base + GPI_INV);

if ( cmd == IOCTL_NONINVERTED_INPUT )
{
    /* set pin to noninverted mode */
    clear_bit(pin_num, (void*)&bitstr);
}
else
{
    /* set pin to inverted mode */
    set_bit(pin_num, (void*)&bitstr);
}

/* write the modified string back to the gpio register */
outl(bitstr, g_drvr_data.mem_base + GPI_INV);
```

2.4 Microsoft Windows* Code Examples

2.4.1 Find Device and Get GPIO Base Address

Find the LPC interface and get the GPIO base address. Use this base address plus the register offsets to access the GPIO registers. (HalGetBusDataByOffset() is an older function, but it is used here for instructional purposes and clarity.)

```
/* Location of the LPC device on the PCI bus */
#define LPC_BUS_NUM 0
#define LPC_DEVICE_NUM 31
#define LPC_FUNCTION_NUM 0

/* offset into low pin count (LPC) config space of the GPIO base
address */
#define GPIO_BAR_OFFSET 0x48
#define DWORD              4

ULONG              result, gpio_base;
PCI_SLOT_NUMBER    slotNum;

/* Make sure that this system has the right LPC controller */
slotNum.u.bits.Reserved       = 0;
slotNum.u.bits.DeviceNumber   = LPC_DEVICE_NUM;
slotNum.u.bits.FunctionNumber = LPC_FUNCTION_NUM;

result = HalGetBusDataByOffset( PCIConfiguration, LPC_BUS_NUM,
                                slotNum.u.AsULONG, gpio_base,
                                GPIO_BAR_OFFSET, DWORD );

/* Clear all but address bits */
```

```
gpio_base &= 0x0000ff80;
```

2.4.2 GPIO_USE_SEL Register

Select GPIO or alternative.

```
/* offsets for first set of gpio registers */
#define  GPIO_USE_SEL   0x00
#define  GP_IO_SEL      0x04
#define  GP_LVL         0x0c
#define  GPO_BLINK      0x18
#define  GPI_INV        0x2c

/* read the register first */
ULONG    val, pin_num;

val = READ_PORT_ULONG((PULONG) gpio_base + GPIO_USE_SEL);

if( IoControlCode == IOCTL_SET_TO_ALTERNATIVE )
{
    /* clear bit to use pin for alternative function */
    val &= ~(1 << pin_num);
}
else
{
    /* set bit to use pin for gpio */
    val |= (1 << pin_num);
}

/* write the data back with modified bit */
WRITE_PORT_ULONG((PULONG) gpio_base + GPIO_USE_SEL, val);
```

2.4.3 GP_IO_SEL Register

Select input or output.

```
/* read the register first */
val = READ_PORT_ULONG((PULONG) gpio_base + GP_IO_SEL);

if( IoControlCode == IOCTL_SET_AS_OUTPUT )
{
    /* clear bit to use pin as output */
    val &= ~(1 << pin_num);
}
else
{
    /* set bit to use pin as input */
    val |= (1 << pin_num);
}

/* write the data back with modified bit */
WRITE_PORT_ULONG((PULONG) gpio_base + GP_IO_SEL, val);
```

2.4.4 GP_LVL Register

Set the output level.

```
/* read the register first */
val = READ_PORT_ULONG((PULONG) gpio_base + GP_LVL);

if( IoControlCode == IOCTL_SET_LOW )
{
    /* write a 0 to pin */
    val &= ~(1 << pin_num);
}
else
{
    /* write a 1 to pin */
    val |= (1 << pin_num);
}

/* write the data back with modified bit */
WRITE_PORT_ULONG((PULONG) gpio_base + GP_LVL, val);
```

2.4.5 GPO_BLINK Register

Enable or disable output blink.

```
/* read the register first */
val = READ_PORT_ULONG((PULONG) gpio_base + GPO_BLINK);

if( IoControlCode == IOCTL_STOP_BLINK )
{
    /* set pin to non-blink mode */
    val &= ~(1 << pin_num);
}
else
{
    /* set pin to blink mode */
    val |= (1 << pin_num);
}

/* write the data back with modified bit */
WRITE_PORT_ULONG((PULONG) gpio_base + GPO_BLINK, val);
```

2.4.6 GPI_INV Register

Enable or disable input inversion.

```
/* read the register first */
val = READ_PORT_ULONG((PULONG) gpio_base + GPI_INV);

if( IoControlCode == IOCTL_NONINVERTED_INPUT )
{
    /* set pin to noninverted mode */
    val &= ~(1 << pin_num);
}
else
{
    /* set pin to inverted mode */
    val |= (1 << pin_num);
}

/* write the data back with modified bit */
WRITE_PORT_ULONG((PULONG) gpio_base + GPI_INV, val);
```

3 Conclusion

Not all GPIO pins are the same. Pins can be defined as input or output, input only, output only, capable of generating an interrupt or not, etc. For the specifics of each GPIO pin, please refer to the GPIO section of the datasheet or external design specification for the particular device.