Embedded Systems

Prof. Myung-Eui Lee (F-102)

melee@kut.ac.kr

한국기술교육대학교
korea University of
Technology and Education

GPIO

- **general-purpose I/O (GPIO) controller**
  - » **121 highly-multiplexed general-purpose I/O (GPIO) pins for use in generating and capturing application-specific input and output signals.**
  - » **Each pin can be programmed as an output, an input, or as bidirectional for certain alternate functions.**
    - ◆ Programmable for: Inputs or outputs, Interrupts or wake-up sources, Rising or falling edge, GPIO or one of several alternate functions
  - » **Input : the GPIOs can be sampled or programmed to generate interrupts from either rising or falling edges.**
  - » **Output : the GPIOs can be individually cleared or set.**
  - » **Alternate function : serve either as GPIO or as an alternate function, but not as both at the same time, providing system flexibility.**
  - » **All GPIO pins are configured as inputs during the assertion of all resets, and they remain inputs until configured otherwise.**
  - » **GPIO35, GPIO36, GPIO37, and GPIO41 for target user**

GPIO

- ## GPIO Block Diagram
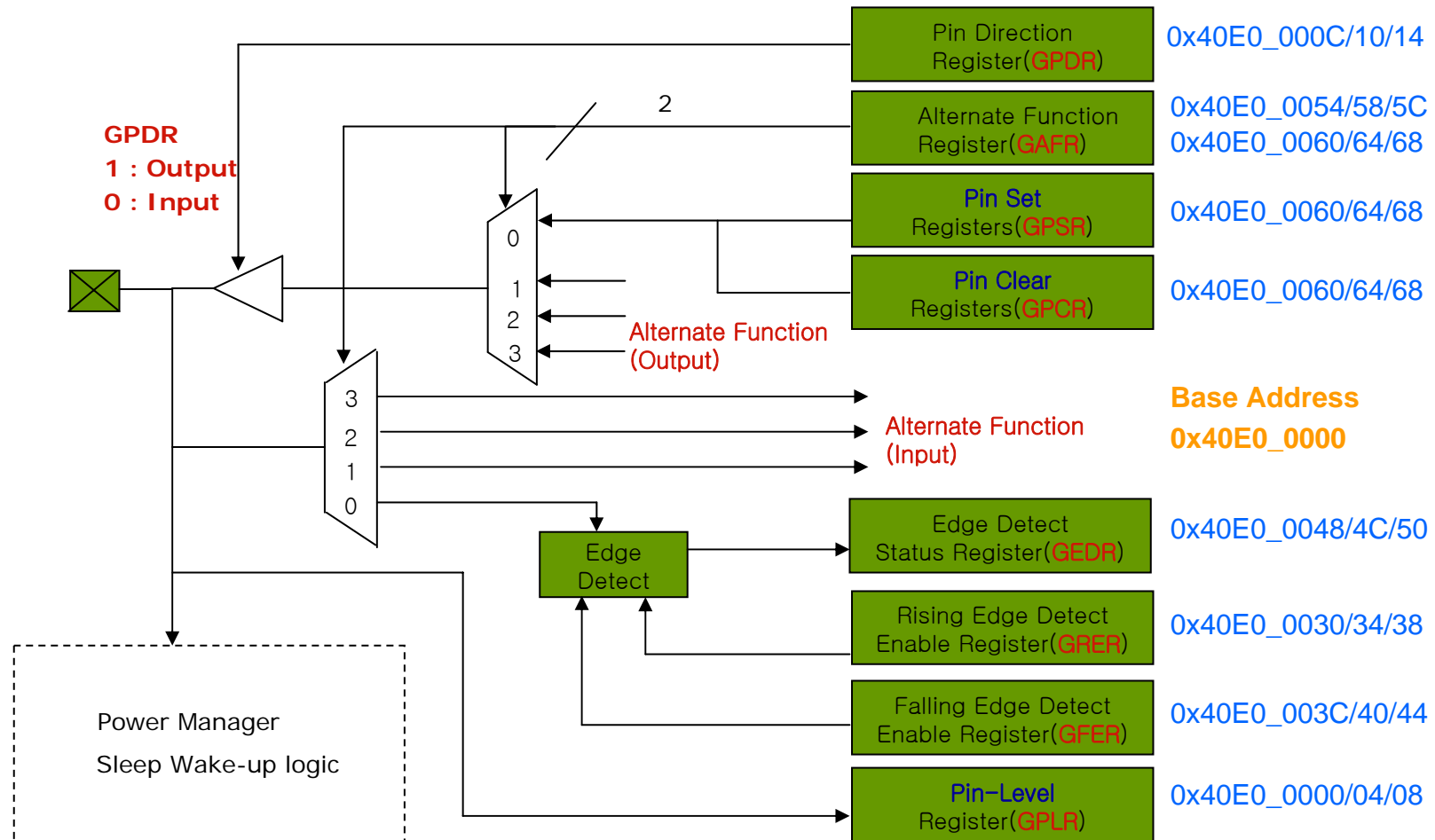
GPIO

- **Alternate function :** 0 = GPIO 1/2/3 = Alt. func.

GPIO Pin	Pin Name	Alternate Function 1 (In)	Alternate Function 2 (In)	Alternate Function 3 (In)	Alternate Function 1 (Out)	Alternate Function 2 (Out)	Alternate Function 3 (Out)
0	GPIO<0>						
1	GPIO<1>/ nRESET_GPIO[4]						
2	SYS_EN[5]						
3	GPIO<3>/ PWR_SCL						
4	GPIO<4>/ PWR_SDA						
5	PWR_CAP<0>[5]						
6	PWR_CAP<1>[5]						
7	PWR_CAP<2>[5]						
8	PWR_CAP<3>[5]						
9	GPIO<9>			FFCTS	HZ_CLK		CHOUT<0>
10	GPIO<10>	FFDCD		USB_P3_5[7]	HZ_CLK		CHOUT<1>
11	GPIO<11>	EXT_SYNC<0>	SSPRXD2	USB_P3_1	CHOUT<0>	PWM_OUT<2>	48_MHz

⋮

116[3]	GPIO<116>	CIF_DD<2>	AC97_SDATA_IN_0	UDET	DVAL<0>	nUVS2	MBGNT
117	GPIO<117>	SCL			SCL		
118	GPIO<118>	SDA			SDA		
119	GPIO<119>	USBHPWR<2>					
120	GPIO<120>					USBHPEN<2>	

GPIO

- **Register Definition**

Register Type	Register Function	GPIO <15:0>	GPIO <31:16>	GPIO <47:32>	GPIO <63:48>	GPIO <79:64>	GPIO <95:80>	GPIO <111:96>	GPIO <120:112>
GPLR	Monitor Pin State	GPLR0		GPLR1		GPLR2		GPLR3	
GPSR	Control Output Pin State	GPSR0		GPSR1		GPSR2		GPSR3	
GPCR		GPCR0		GPCR1		GPCR2		GPCR3	
GPDR	Set Pin Direction	GPDR0		GPDR1		GPDR2		GPDR3	
GRER	Detect Rising/Falling Edge	GRER0		GRER1		GRER2		GRER3	
GFER		GFER0		GFER1		GFER2		GFER3	
GEDR	Detect Edge Type	GEDR0		GEDR1		GEDR2		GEDR3	
GAFR	Set Alternate Functions	GAFR0_L	GAFR0_U	GAFR1_L	GAFR1_U	GAFR2_L	GAFR2_U	GAFR3_L	GAFR3_U

GPIO

- ## GAFR1_L Register

Physical Address
0x40E0_005C

GAFR1_L

GPIO Controller

User Settings																															

Bit 31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

AF47	AF46	AF45	AF44	AF43	AF42	AF41	AF40	AF39	AF38	AF37	AF36	AF35	AF34	AF33	AF32

Reset 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Bits	Access	Name	Description
<31:0>	R/W	AFx	GPIO 'x' Alternate Function Select Bits (where x = 32 through 47) A bit-pair in this register determines the corresponding GPIO pin's functionality as one of the alternate functions that is mapped to it or as a generic GPIO pin. 0b00 = The corresponding GPIO pin (GPIO<x>) is used as a general-purpose I/O. 0b01 = The corresponding GPIO pin (GPIO<x>) is used for its alternate function 1. 0b10 = The corresponding GPIO pin (GPIO<x>) is used for its alternate function 2. 0b11 = The corresponding GPIO pin (GPIO<x>) is used for its alternate function 3.

GPIO

- ## GPDR1 Register

Bits	Access	Name	Description
<31:0>	R/W	PDx	GPIO Pin Direction 'x' (where x = 32 to 63) This field configures the direction of each GPIO. 0 = Pin configured as an input. 1 = Pin configured as an output.

GPIO

- ## GPSR1 Register

Physical Address 0x40E0_001C   **GPSR1**   **GPIO Controller**

Bits	Access	Name	Description
<31:0>	W	PSx	GPIO Pin Set 'x' (where x = 32 to 63) This field configures the output level of each GPIO. 0 = Pin level unaffected. 1 = If pin configured as an output, set pin level high (one).

GPIO

- ## GPCR1 Register

Physical Address 0x40E0_0028			GPCR1			GPIO Controller		

Bits	Access	Name	Description
<31:0>	W	PCx	GPIO Pin Clear 'x' (where x = 32 to 63) This field configures the output level of each GPIO. 0 = Pin level unaffected. 0 = If pin configured as an output, clear pin level low (zero).

GPIO

- ## GPER1 Register

Physical Address 0x40E0_0040

GFER1

GPIO Controller

User Settings																															
Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	FE63	FE62	FE61	FE60	FE59	FE58	FE57	FE56	FE55	FE54	FE53	FE52	FE51	FE50	FE49	FE48	FE47	FE46	FE45	FE44	FE43	FE42	FE41	FE40	FE39	FE38	FE37	FE36	FE35	FE34	FE33	FE32
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	Access	Name	Description
<31:0>	R/W	FEx	GPIO Pin Falling Edge detect Enable bit 'x' (where x = 32 to 63) This field configures the output level of each GPIO. 0 = Disables rising-edge detect enable. 1 = Sets corresponding GEDR status bit when a falling edge is detected on the GPIO pin.

GPIO Test Program

- ## Gpioled-test.c

```c
int main(int argc, char **argv)
{                int dev;          char buff;
if(argc <= 1) {   printf("please input the parameter! ex)./test on or off\n");
                                return -1;
                }
dev = open("/dev/gpioled", O_WRONLY);
if (dev != -1)     {                if(argv[1][0] == 'o' && argv[1][1] == 'n') buff = 1;
                                else if(argv[1][0] == 'o' && argv[1][1] == 'f' && argv[1][2] == 'f') buff = 0;
                                write(dev,&buff,1);
                                close(dev);      }                    else
                {                printf( "Device Open ERROR!\n");
                                exit(-1);        }
return(0);
}
```

- ## Gpioled.c

```c
#define GPIOLED_MAJOR   0
#define GPIOLED_NAME    "gpio"

int gpioled_open(struct inode *minode, struct file *mfile) {

        GAFR1_L = GAFR1_L & (~0x000000c0);
        GPDR1 = GPDR1 | 0x00000008;
        return 0;
}

int gpioled_release(struct inode *minode, struct file *mfile) {
        return 0;
}

ssize_t gpioled_write(struct file *inode, const char *gdata, size_t length, loff_t *off_what) {
        unsigned char  c;

        get_user(c,gdata);
        // c= 0: off /  c= 1: on
        if (c != 0) GPCR1 = GPCR1 | 0x000008;           // clear(0)= led on
        else  GPSR1 = GPSR1 | 0x000008;                                 // set(1)= led off

        return length;
}

static struct file_operations gpioled_fops = {
        write : gpioled_write,
        open : gpioled_open,
        release : gpioled_release,
};
```

```
int gpioled_init(void) {
        int result; // major number
        result = register_chrdev(GPIOLED_MAJOR,GPIOLED_NAME,&gpioled_fops);
        if(result < 0) {
                printk(KERN_WARNING"Can't get any major\n");
                return result;
        }
        printk("init module, major number: %d\n", result);
        return 0;
}

void gpioled_exit(void) {
        if(unregister_chrdev(GPIOLED_MAJOR,GPIOLED_NAME))
                printk(KERN_WARNING"%s DRIVER CLEANUP FAILED\n",GPIOLED_NAME);
}
module_init(gpioled_init);
module_exit(gpioled_exit);
```

LED test run

- **Make module :**
  - » **sample modules in microcom homepage**
- **Load the module**
  - » **insmod gpioled.ko**
- **Check that the module is loaded**
  - » **cat /proc/modules**
- **Make special file**
  - » **mknod /dev/gpioled c xxx 0**
- **Test run**
  - » **./gpioled-test on or off**
- **Remove the module**
  - » **rmmod gpioled**

GPIO Test Program

- ## Gpioin-test.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <errno.h>

static int dev;
int main(void)
{           char buff[40];

            dev = open("/dev/gpioin",O_RDWR);

            if(dev < 0) {
                        printf( "Device Open ERROR!\n");
                        exit(1);
            }

            printf("Please push the GPIO 36 button!\n");
            read(dev,buff,40);
            printf("%s\n",buff);

            close(dev);
            return 0;
}
```

GPIO Device Driver

- ## Gpioin.c : interrupt

```c
#define IRQ_BUTTON                              IRQ_GPIO(36)
static int GPIO_MAJOR = 0;
static DECLARE_WAIT_QUEUE_HEAD(wait_queue); //  include/linux/wait.h

static irqreturn_t button_interrupt(int irq, void *dev_id, struct pt_regs *regs){
            wake_up_interruptible(&wait_queue);
            return 0;
}

static ssize_t gpioin_read(struct file *filp, char *buf, size_t count, loff_t *l){
            char hello[] = "GPIO 36 port was pushed!";

            interruptible_sleep_on(&wait_queue);
            copy_to_user(buf,&hello,sizeof(hello)); // printf in main() in application gpioin-test.c
            return 0;
}

static int gpioin_open(struct inode *inode, struct file *filp){
            int res;

            GAFR1_L &= ~(0x300);
            GPDR1 &= ~(0x10);               // input
            GFER1 |= 0x10;                                   // enable falling edge
            set_irq_type(IRQ_BUTTON,IRQT_FALLING);
            res = request_irq(IRQ_BUTTON,&button_interrupt,SA_INTERRUPT,"Button",NULL);
```

```
if(res < 0)
                                printk(KERN_ERR "%s: Request for IRQ %d failed\n",__FUNCTION__,IRQ_BUTTON);
                else {
                                enable_irq(IRQ_BUTTON);
                }
                return 0;
}

static int gpioin_release(struct inode *inode, struct file *filp)
{               free_irq(IRQ_BUTTON,NULL);
                disable_irq(IRQ_BUTTON);

                return 0;
}

static struct file_operations gpioin_fops = {
                read:           gpioin_read,
                open:           gpioin_open,
                release:        gpioin_release,
};

int gpioin_init(void){
                int result;
                result = register_chrdev(GPIO_MAJOR,"GPIO INTERRUPT",&gpioin_fops);

                if(result < 0) {
                                printk(KERN_WARNING"Can't get major %d\n",GPIO_MAJOR);
                                return result;
                }
```

```
        if(GPIO_MAJOR == 0) GPIO_MAJOR = result;
        printk("init module, GPIO major number : %d\n",result);

        return 0;
}

void gpioin_exit(void){
        unregister_chrdev(GPIO_MAJOR,"GPIO INTERRUPT");
        return;
}

module_init(gpioin_init);
module_exit(gpioin_exit);
```

Interrupt

- ## /include/asm-arm/irq.h

```
37 #define IRQT_NOEDGE    (0)
38 #define IRQT_RISING    (__IRQT_RISEDGE)
39 #define IRQT_FALLING   (__IRQT_FALEDGE)
40 #define IRQT_BOTHEDGE  (__IRQT_RISEDGE|__IRQT_FALEDGE)
41 #define IRQT_LOW       (__IRQT_LOWLVL)
42 #define IRQT_HIGH      (__IRQT_HIGHLVL)
43 #define IRQT_PROBE     (1 << 4)
```

- ## /arch/arm/kernel/irq.c

```
171 void enable_irq(unsigned int irq)
172 {
173     struct irqdesc *desc = irq_desc + irq;
174     unsigned long flags;
175
176     spin_lock_irqsave(&irq_controller_lock, flags);
177     if (unlikely(!desc->disable_depth)) {
178         printk("enable_irq(%u) unbalanced from %p\n", irq,  __builtin_return_address(0));
```

Input test run

- **Make module :**
  - » **sample modules in microcom homepage**
- **Load the module**
  - » **insmod gpioin.ko**
- **Check that the module is loaded**
  - » **cat /proc/modules**
- **Make special file**
  - » **mknod /dev/gpioin c xxx 0**
- **Test run**
  - » **./gpioin-test**
- **Remove the module**
  - » **rmmod gpioin**