Embedded Systems

Ch 5B.
Parallel Interface (II)

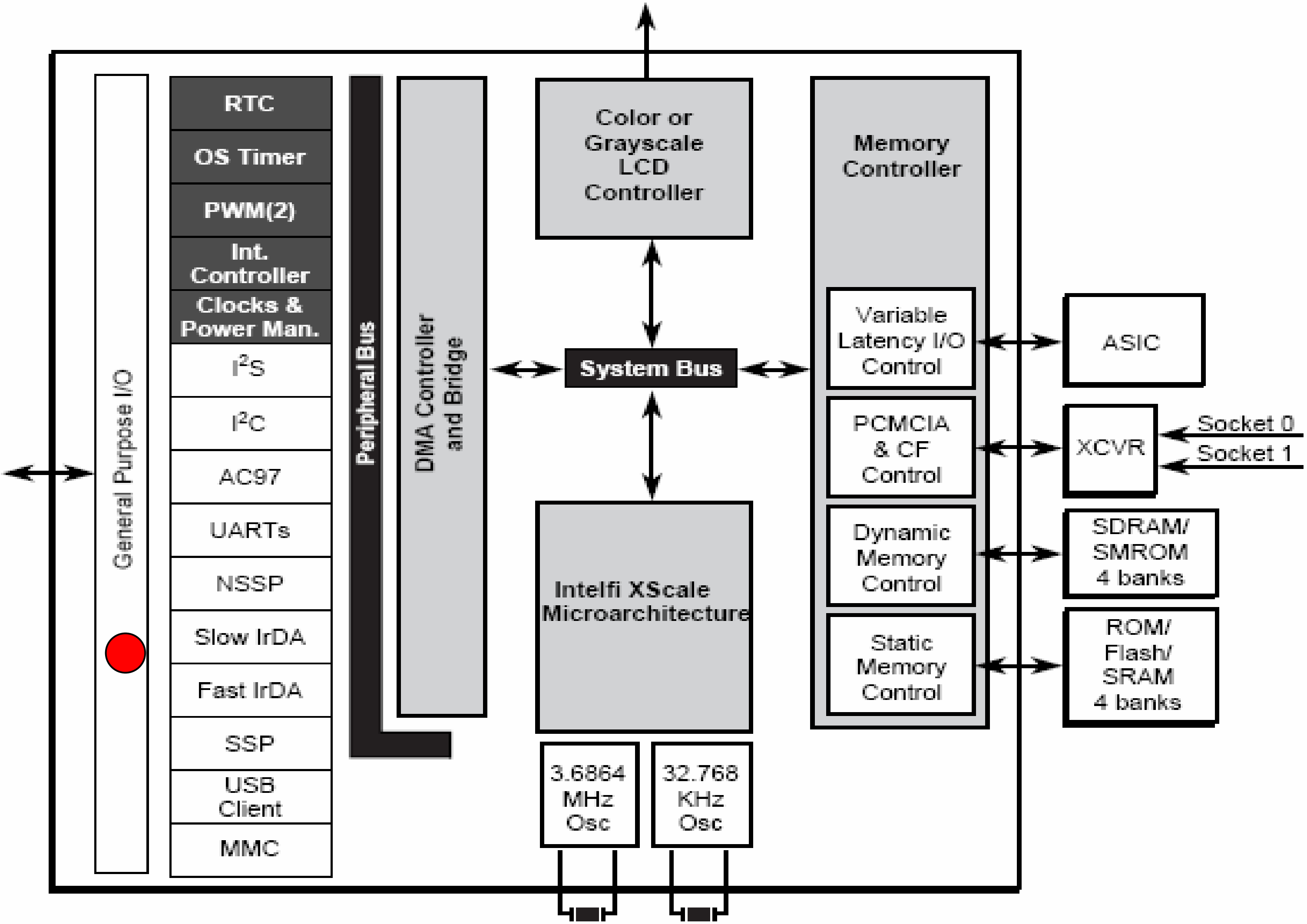
Byung Kook Kim
Dept of EECS
Korea Advanced Institute of Science and Technology

Overview

- 1. Introduction to Parallel Ports
- 2. Input/Output Mechanisms
- 3. IEEE 1284
- 4. Centronix Interface
- 5. GPIO (General Purpose Input/Output) Interface
- 6. GPIO Driver
- 7. DIO (Digital Input/Output) Driver

5. GPIO Interface

- Block Diagram of PXA255



GPIO Interface (II)

- **GPIO (General Purpose Input/Output)**
  - Each GPIO pin can be individually programmed as an output or an input.
  - Inputs can cause interrupts on rising or falling edges.
  - Primary GPIO pins are not shared with peripherals while secondary GPIO pins have alternate functions which can be mapped to the peripherals.

- **GPIO란?**
  - GPIO(General Purpose Input/Output)**란 일반적인 용도로 사용 가능한 디지털 입출력 기능의** Port pins **이다**.
  - PXA255**의** GPIO**는 총** 85**개이며 각각이** pin **들은** input/output**으로 프로그램 되거나 인터럽트** source**로 사용될 수 있다**.
  - PXA255**의 대부분의** GPIO**는 단순히 디지털 입출력 뿐만 아니라 부가적인 기능을 갖고 있다. 그래서 다른 기능을 사용하다 보면 처음 생각보다 단순** DIO**로 사용할** GPIO**가 적어진다**.

GPIO Interface (III)

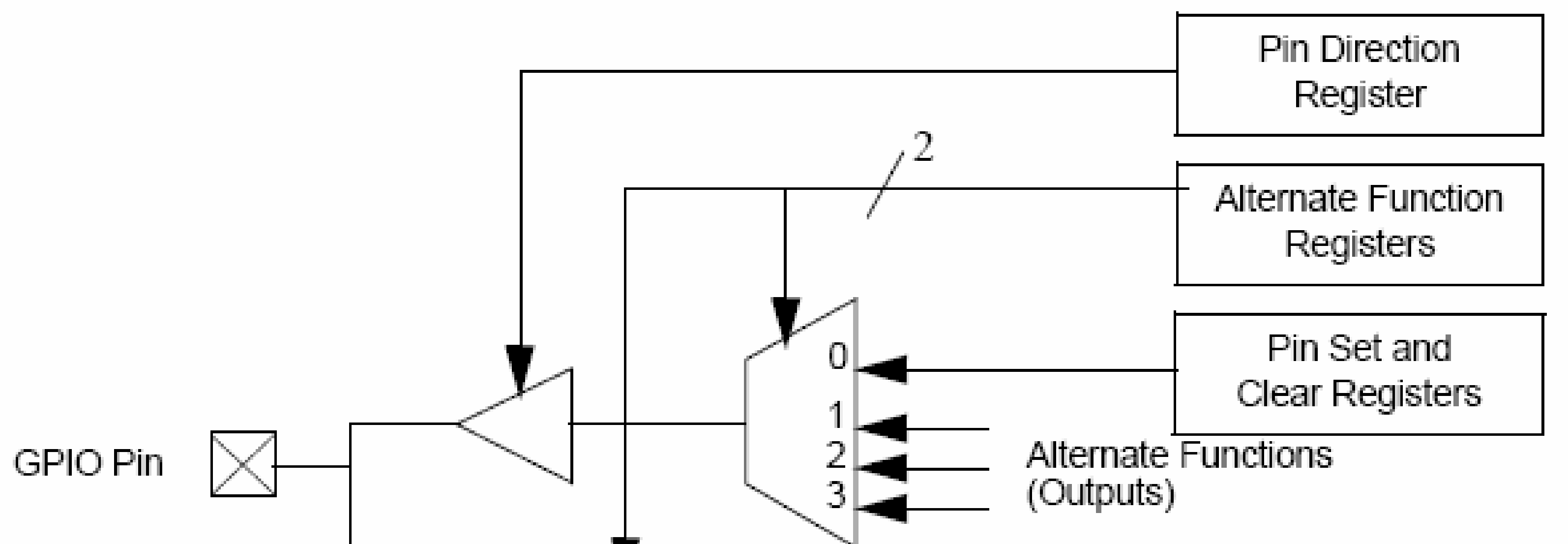
- Remarks on GPIO
  - The PXA255 processor enables and controls its 85 GPIO pins through the use of 27 registers which configure the pin direction (input or output), pin function, pin state (outputs only), pin level detection (inputs only), and selection of alternate functions.
  - A portion of the GPIOs can be used to bring the processor out of Sleep mode.
  - Take care when choosing which GPIO pin is assigned as a GPIO function because many of the GPIO pins have alternate functions and can be configured to support processor peripherals.
  - Configure all unused GPIOs as outputs to minimize power consumption.

GPIO Interface (IV)

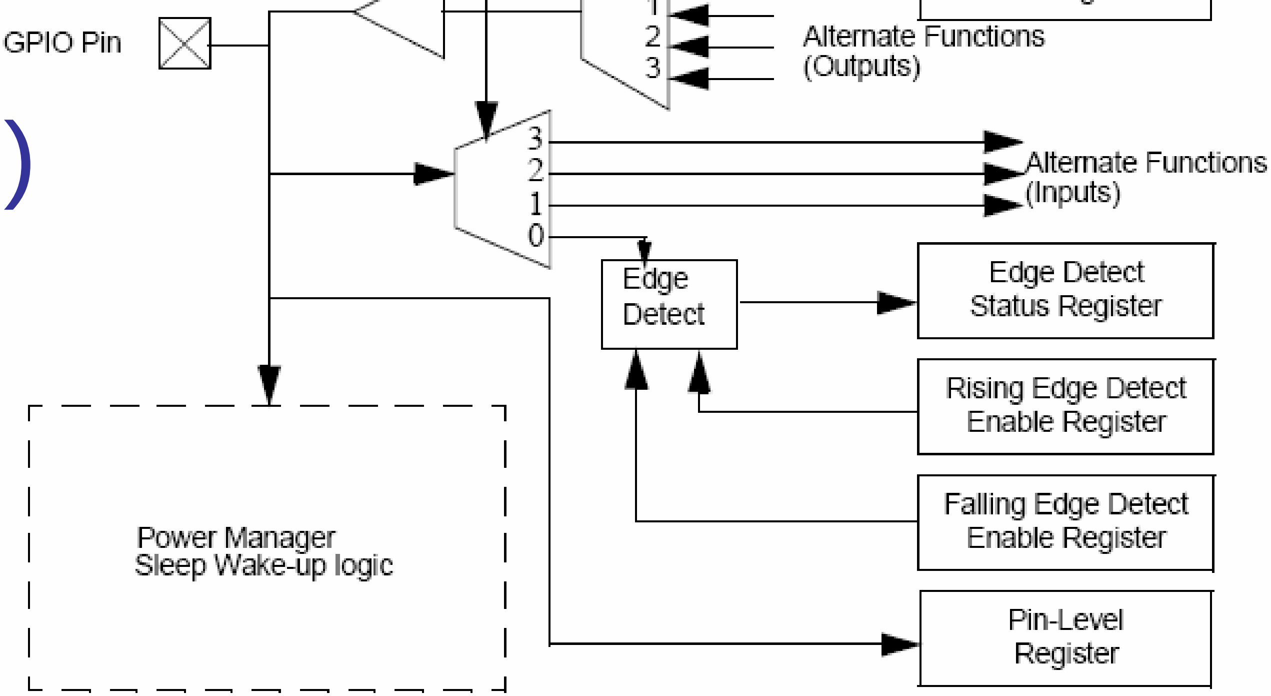
- **GPIO Operation**
  - The PXA255 processor provides 85 GPIO pins for use in generating and capturing application-specific input and output signals.
  - Each pin can be programmed as either an input or output.
  - When programmed to be an input, a GPIO can also serve as an interrupt source.
  - All 85 pins are configured as inputs during the assertion of all resets, and remain as inputs until they are configured otherwise.

GPIO Interface (V)

- **GPIO Operation (II)**
  - Use the GPIO Pin Direction Register (GPDR) to set whether the GPIO pins are outputs or inputs.
  - When programmed as an output, the pin can be set high by writing to the GPIO Pin Output Set Register (GPSR) and cleared low by writing to the GPIO Pin Output Clear Register (GPCR).
  - The set and clear registers can be written to regardless of whether the pin is configured as an input or an output.
  - If a pin is configured as an input, the programmed output state occurs when the pin is reconfigured to be an output.


Pin Direction Register
2
Alternate Function Registers
Pin Set and Clear Registers
0
1
2
3
Alternate Functions (Outputs)
GPIO Pin

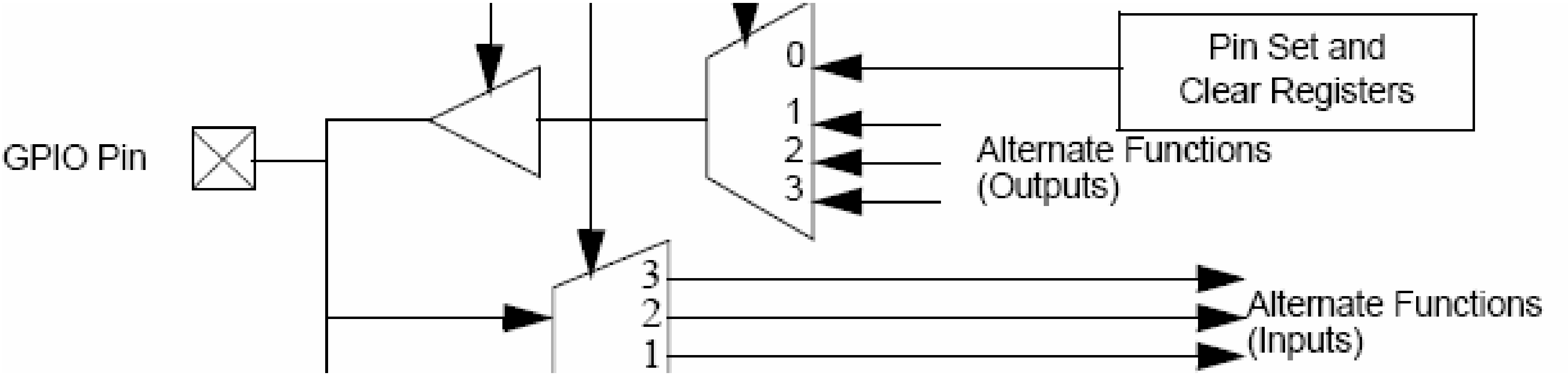
GPIO Interface (VI)


GPIO Pin
1
2
3
Alternate Functions (Outputs)
3
2
1
0
Alternate Functions (Inputs)
Edge Detect
Edge Detect Status Register
Rising Edge Detect Enable Register
Falling Edge Detect Enable Register
Pin-Level Register
Power Manager Sleep Wake-up logic

- **GPIO Operation (III)**
  - Validate each GPIO pin’s state by reading the GPIO Pin Level Register (GPLR).
  - You can read this register any time to confirm the state of a pin.
  - In addition, use the GPIO Rising Edge Detect Enable Register (GRER) and GPIO Falling Edge Detect Enable Register (GFER) to detect either a rising edge or falling edge on each GPIO pin.
  - Use the GPIO Edge Detect Status register (GEDR) to read edge detect state.
  - These edge detects can be programmed to generate interrupts.
  - Also use GPIO[15:0] to generate wake-up events that bring the PXA255 processor out of sleep mode.

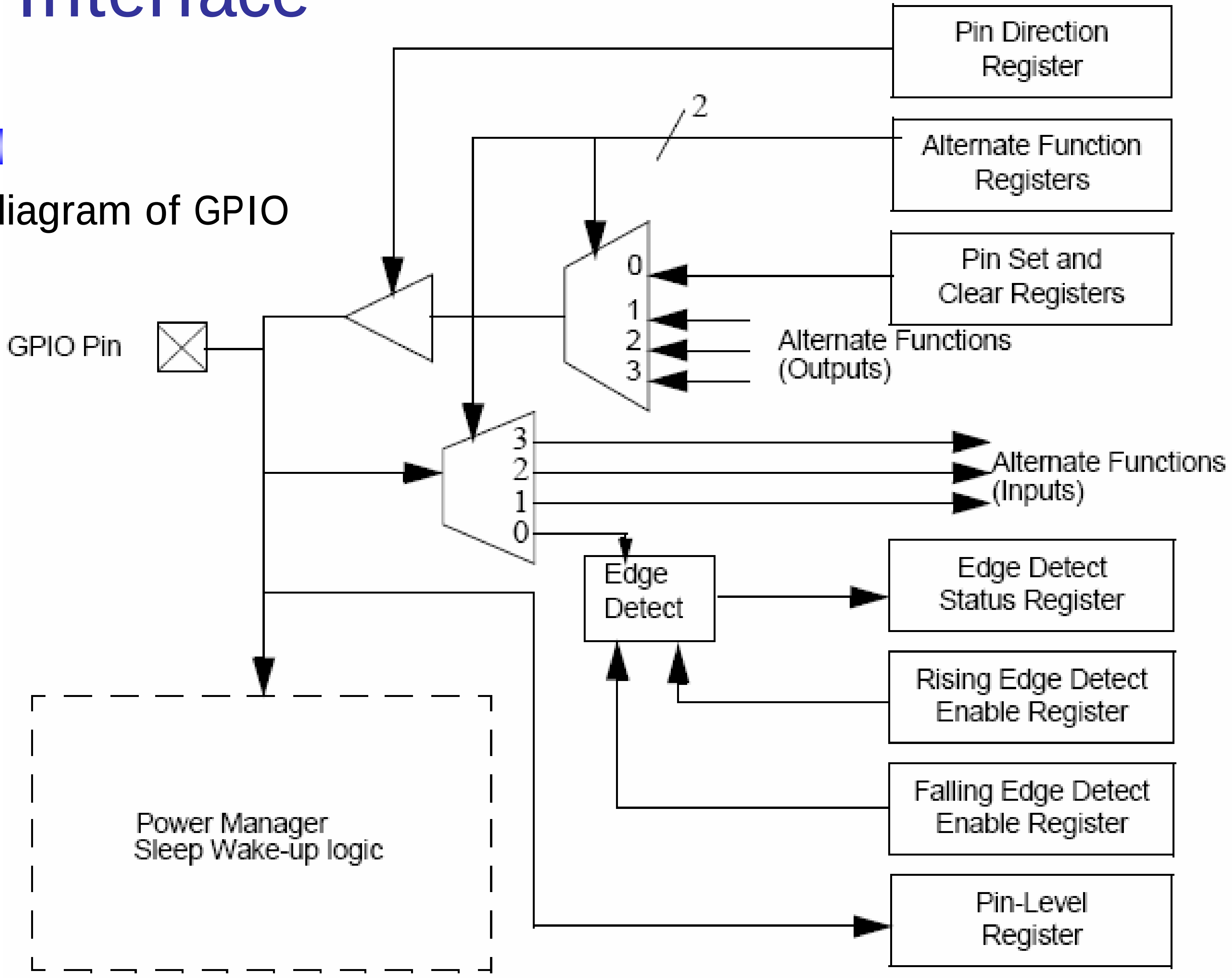
GPIO Interface (VII)

- **GPIO Operation (IV)**
  - Most GPIO pins can also serve an alternate function within the processor.
  - Certain modes within the serial controllers and LCD controller require extra pins.
  - These functions are hardwired into specific GPIO pins.
  - Even though a GPIO pin is used for an alternate function, you must still program the proper direction of that pin through the GPDR.



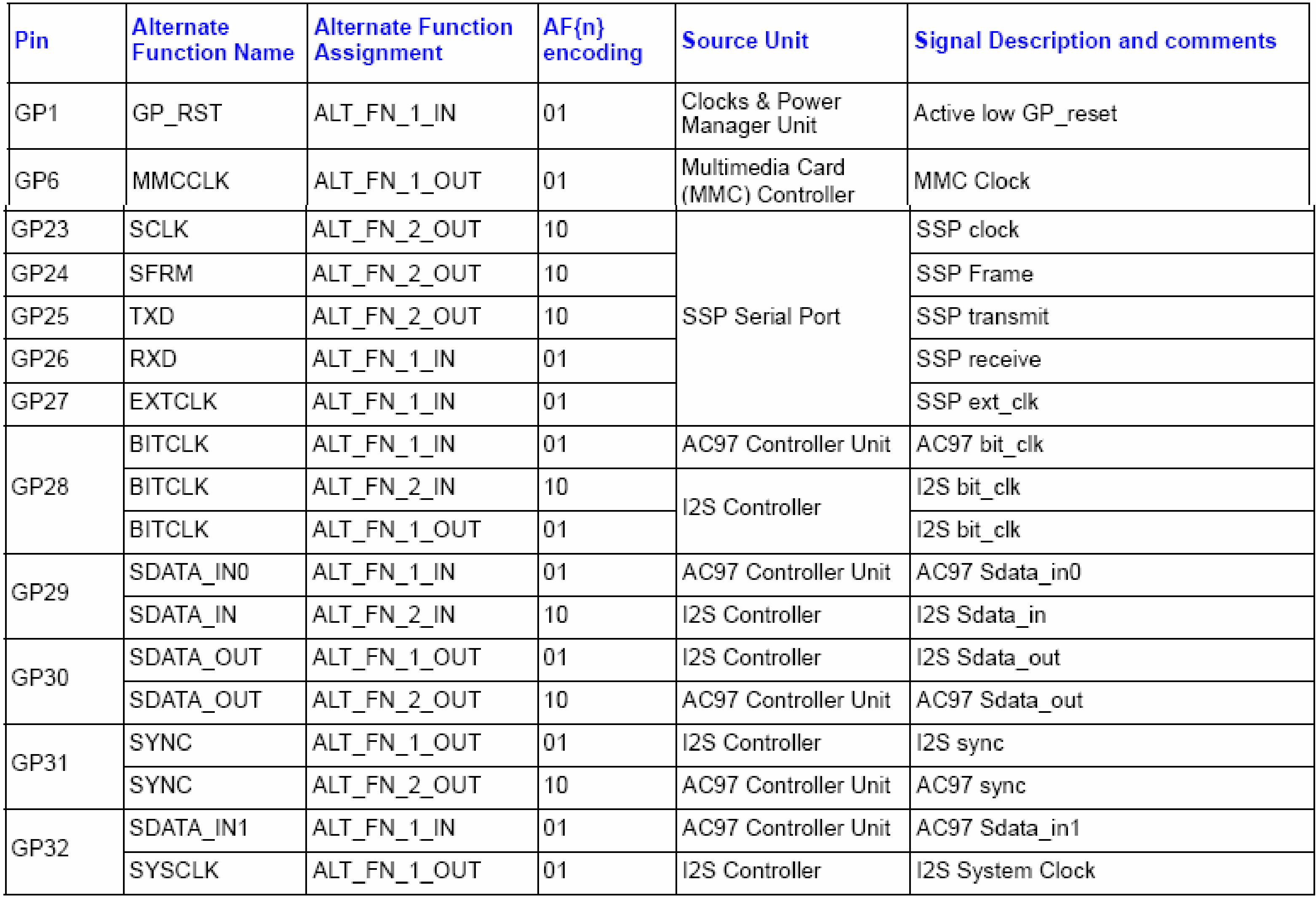
GPIO Interface (VIII)

- Block diagram of GPIO


Pin Direction Register
2
Alternate Function Registers
Pin Set and Clear Registers
0
1
2
3
Alternate Functions (Outputs)
GPIO Pin
3
2
1
0
Alternate Functions (Inputs)
Edge Detect
Edge Detect Status Register
Rising Edge Detect Enable Register
Falling Edge Detect Enable Register
Power Manager Sleep Wake-up logic
Pin-Level Register

GPIO Interface (IX)

- GPIO Alternate Functions (Partial)



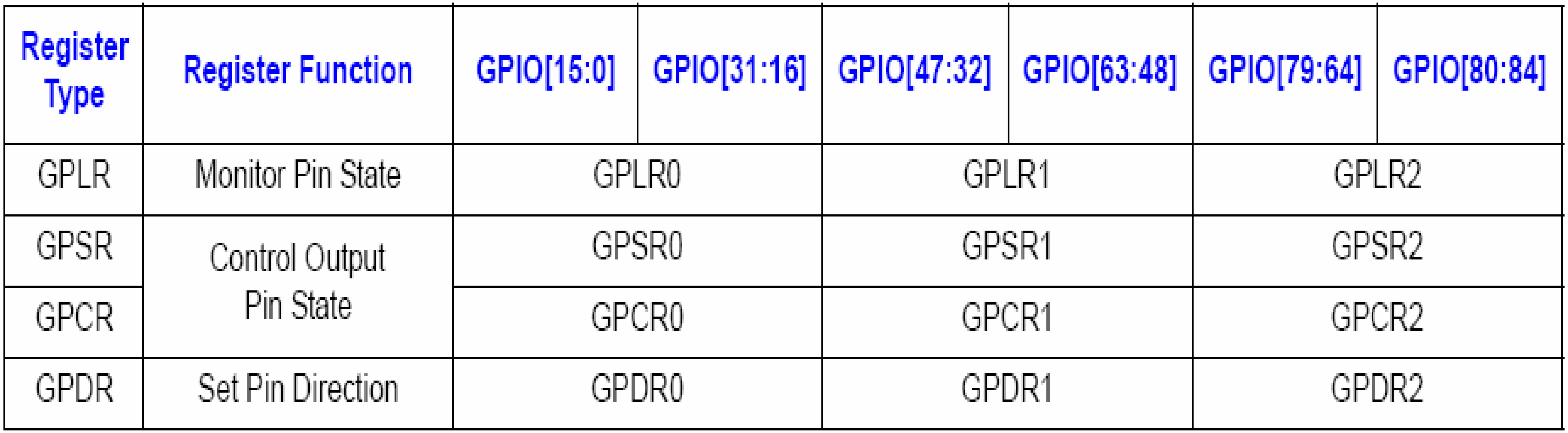
Pin	Alternate Function Name	Alternate Function Assignment	AF{n} encoding	Source Unit	Signal Description and comments
GP1	GP_RST	ALT_FN_1_IN	01	Clocks & Power Manager Unit	Active low GP_reset
GP6	MMCCLK	ALT_FN_1_OUT	01	Multimedia Card (MMC) Controller	MMC Clock
GP23	SCLK	ALT_FN_2_OUT	10	SSP Serial Port	SSP clock
GP24	SFRM	ALT_FN_2_OUT	10		SSP Frame
GP25	TXD	ALT_FN_2_OUT	10		SSP transmit
GP26	RXD	ALT_FN_1_IN	01		SSP receive
GP27	EXTCLK	ALT_FN_1_IN	01		SSP ext_clk
GP28	BITCLK	ALT_FN_1_IN	01	AC97 Controller Unit	AC97 bit_clk
	BITCLK	ALT_FN_2_IN	10	I2S Controller	I2S bit_clk
	BITCLK	ALT_FN_1_OUT	01		I2S bit_clk
GP29	SDATA_IN0	ALT_FN_1_IN	01	AC97 Controller Unit	AC97 Sdata_in0
	SDATA_IN	ALT_FN_2_IN	10	I2S Controller	I2S Sdata_in
GP30	SDATA_OUT	ALT_FN_1_OUT	01	I2S Controller	I2S Sdata_out
	SDATA_OUT	ALT_FN_2_OUT	10	AC97 Controller Unit	AC97 Sdata_out
GP31	SYNC	ALT_FN_1_OUT	01	I2S Controller	I2S sync
	SYNC	ALT_FN_2_OUT	10	AC97 Controller Unit	AC97 sync
GP32	SDATA_IN1	ALT_FN_1_IN	01	AC97 Controller Unit	AC97 Sdata_in1
	SYSCLK	ALT_FN_1_OUT	01	I2S Controller	I2S System Clock

GPIO Interface (X)

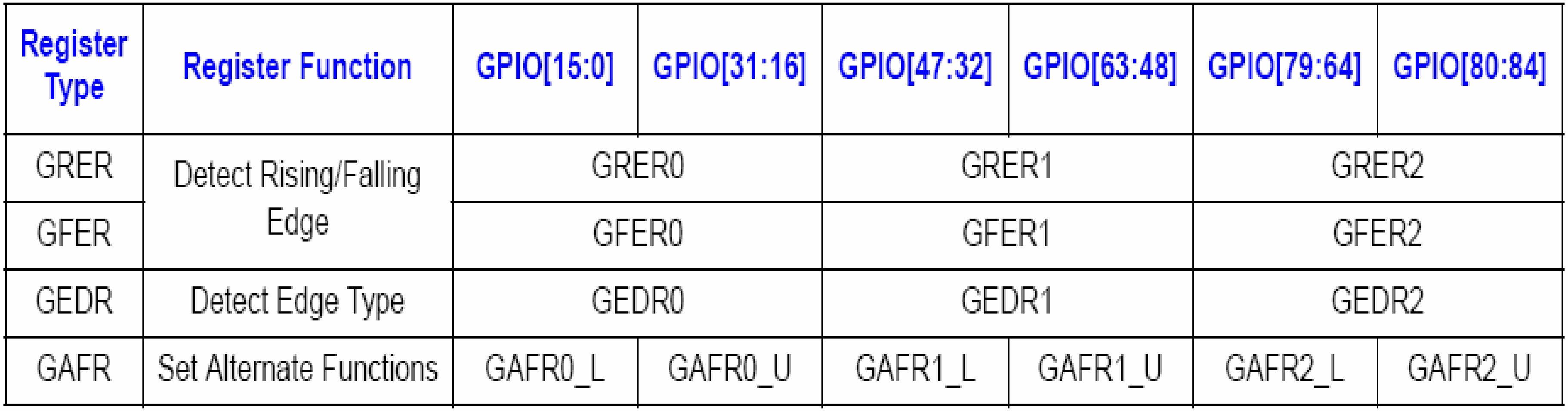
- **GPIO Register Definitions**
  - Twenty-seven 32-bit registers within the GPIO control block.
    - Nine distinct register functions
    - Three sets of each of the nine registers to serve the 85 GPIOs.
  - Registers
    - Three monitor pin state (GPLR)
    - Six control output pin state (GPSR, GPCR)
    - Three control pin direction (GPDR)
    - Six control whether rising edges and/or falling edges are detected (GRER & GFER)
    - Three indicate when specified edge types have been detected on pins (GEDR).
    - Six determine whether a pin is used as a normal GPIO or whether it is to be taken over by one of three possible alternate functions (GAFR_L, GAFR_U).

GPIO Interface (XI)

- GPIO Register Definitions Table



Register Type	Register Function	GPIO[15:0]	GPIO[31:16]	GPIO[47:32]	GPIO[63:48]	GPIO[79:64]	GPIO[80:84]
GPLR	Monitor Pin State	GPLR0		GPLR1		GPLR2	
GPSR	Control Output Pin State	GPSR0		GPSR1		GPSR2	
GPCR		GPCR0		GPCR1		GPCR2	
GPDR	Set Pin Direction	GPDR0		GPDR1		GPDR2	



Register Type	Register Function	GPIO[15:0]	GPIO[31:16]	GPIO[47:32]	GPIO[63:48]	GPIO[79:64]	GPIO[80:84]
GRER	Detect Rising/Falling Edge	GRER0		GRER1		GRER2	
GFER		GFER0		GFER1		GFER2	
GEDR	Detect Edge Type	GEDR0		GEDR1		GEDR2	
GAFR	Set Alternate Functions	GAFR0_L	GAFR0_U	GAFR1_L	GAFR1_U	GAFR2_L	GAFR2_U

GPIO Interface (XII)

- **GPIO Pin-Level Registers (GPLR0, GPLR1, GPLR2)**
  - Check the state of each of the GPIO pins by reading the GPIO Pin Level register (GPLR).
  - Each bit in the GPLR corresponds to one pin in the GPIO.
    - GPLR0[31:0] correspond to GPIO[31:0],
    - GPLR1[31:0] correspond to GPIO[63:32] and
    - GPLR2[16:0] correspond to GPIO[84:64].
    - Use the GPLR0–2 read-only registers to determine the current value of a particular pin (regardless of the programmed pin direction).
    - For reserved bits, reads return zero.

- **GPIO Pin Direction Registers (GPDR0, GPDR1, GPDR2)**
  - The GPDR contain one direction control bit for each of the 85 GPIO pins.
  - If a direction bit is programmed to a one, the GPIO is an output.
  - If it is programmed to a zero, it is an input.
  - Reserved bits must be written to zeros and reads to the reserved bits must be ignored.

GPIO Interface (XIII)

- **GPIO Pin Output Set Registers (GPSR0, GPSR1, and GPSR2) and Pin Output Clear Registers (GPCR0, GPCR1, GPCR2)**
  - When a GPIO is configured as an output, the state of the pin can be controlled by writing to either the GPSR or GPCR.
    - An output pin is set high by writing a one to its corresponding bit within the GPSR.
    - To clear an output pin, a one is written to the corresponding bit within the GPCR.
  - Remarks
    - GPSR and GPCR are write-only registers: Reads return unpredictable values.
    - Writing a zero to any of the GPSR or GPCR bits has no effect on the state of the pin.
    - Writing a one to a GPSR or GPCR bit corresponding to a pin that is configured as an input is effective only after the pin is configured as an output.
    - Reserved bits must be written with zeros and reads must be ignored.

GPIO Interface (XIV)

- **GPIO Rising Edge Detect Enable Registers (GRER0, GRER1, GRER2) and Falling Edge Detect Enable Registers (GFER0, GFER1, GFER2)**
  - Each GPIO can also be programmed to detect a rising-edge, falling-edge, or either transition on a pin.
  - When an edge is detected that matches the type of edge programmed for the pin, a status bit is set.
  - The interrupt controller can be programmed so that an interrupt is signalled to the core when any of these status bits are set.
  - Additionally, the interrupt controller can be programmed so that a subset of the status bits causes the processor to wake from Sleep mode when they are set.

GPIO Interface (XV)

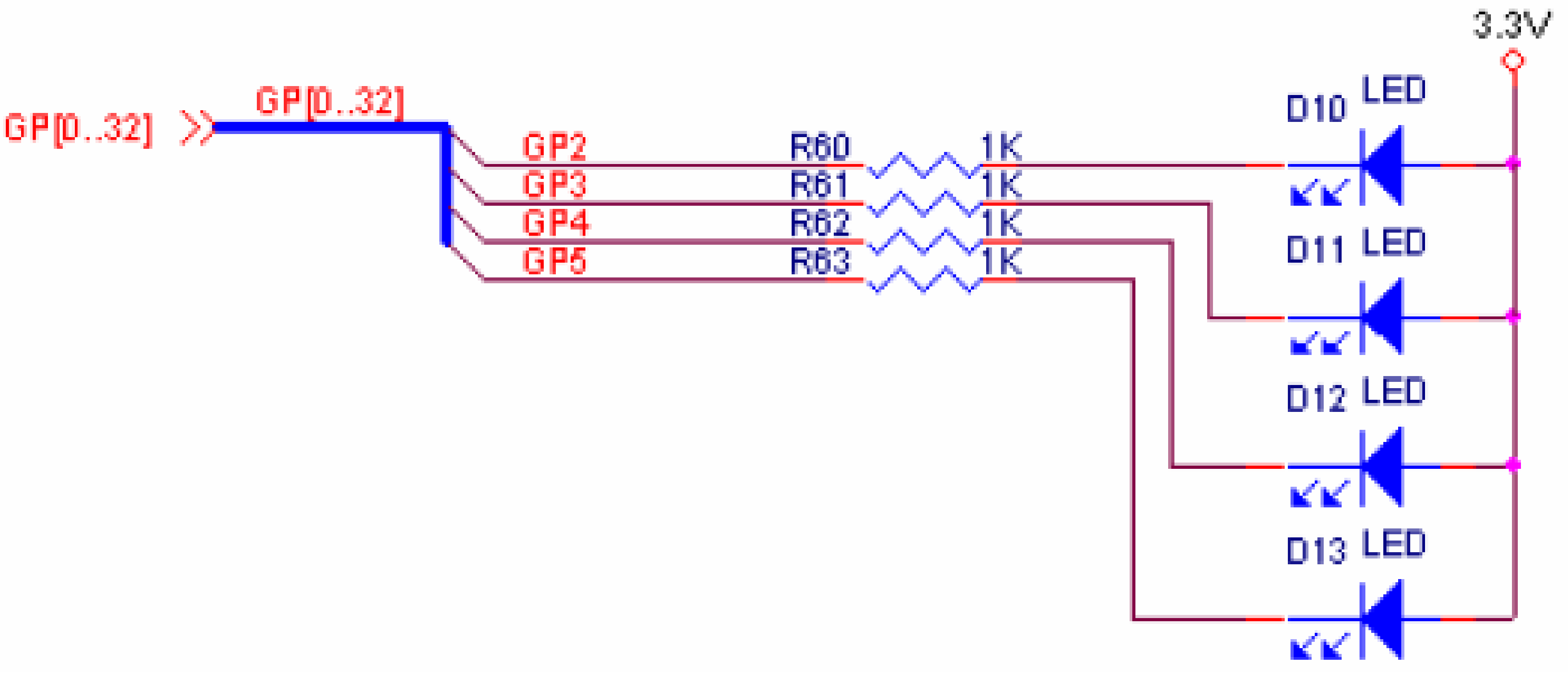
- **GPIO Edge Detect Status Register (GEDR0, GEDR1, GEDR2)**
  - When an edge detect occurs on a pin that matches the type of edge programmed in the GRER and/or GFER registers, the corresponding status bit is set in GEDR.
  - Once a GEDR bit is set by an edge event, the bit remains set until the user clears it by writing a one to the status bit. Writing a zero to a GEDR status bit has no effect.
  - Each edge detect that sets the corresponding GEDR status bit for GPIO[84:0] can trigger an interrupt request. GPIO[84:2] together form a group that can cause one interrupt request to be triggered when any one of GEDR[84:2] are set. GPIO[0] and GPIO[1] cause independent first-level interrupts.

GPIO Interface (XVI)

- **GPIO Alternate Function Register (GAFR0_L, GAFR0_U, GAFR1_L, GAFR1_U, GAFR2_L, GAFR2_U)**
  - Each GPIO can be configured to be either a generic GPIO pin, one of 3 alternate input functions, or one of 3 alternate output functions.
  - To select any of the alternate input functions, the GPDR register must configure the GPIO to be an input. Similarly, only GPIOs configured as outputs by the GPDR can be configured for alternate output functions.
  - Each GPIO pin has a pair of bits assigned to it whose values determine which function (normal GPIO, alternate function 1, alternate function 2 or alternate function 3) the GPIO performs.
    - "00" indicates normal GPIO function
    - "01" selects alternate input function 1 (ALT_FN_1_IN) or alternate output function 1 (ALT_FN_1_OUT)
    - "10" selects alternate input function 2 (ALT_FN_2_IN) or alternate output function 2 (ALT_FN_2_OUT)
    - "11" selects alternate input function 3 (ALT_FN_3_IN) or alternate output function 3 (ALT_FN_3_OUT)

6. GPIO Driver

- GPIO LED driver
  - Circuit diagram



GPIO Driver (II)

- Installing GPIO Driver
  - Makefile

```
KERNELDIR = /project/ez-x5/test/kernel/linux
DEV_INCLUDEDIR = $(KERNELDIR)/include -I./ -I../include
include $(KERNELDIR)/.config
CFLAGS += -Wall –D__KERNEL__ -DMODULE $(DEV_INCLUDEDIR)
            $(DEBFLAGS)
TARGET = gpio_dev
OBJS = $(TARGET).o
SRCS = gpio.c
CFLAGS += -O2 –g

All: $(TARGET).o
$(TARGET).o: $(SRCS:.c=.o)
            $(LD) –r $^ -o $@

Clean:
            rm –f *.o *~ core .depend

Dep:
            gccmakeup $(DEV_INCLUDEDIR) $(SRCS)

#DO NOT DELETE
```

GPIO Driver (III)

- Installing GPIO Driver (II)
  - Edit GPIO driver
    - Makefile (for driver)
    - vi gpio.c
    - vi gpio.h
  - Edit application
    - Makefile (for application)
    - vi test.c
    - make
      - Generated: test_app
  - Compile driver
    - make clean
    - make dep
    - make
      - Generated: gpio.o gpio_dev.o

GPIO Driver (IV)

```
int init_module( void )
{
        int result;

        // 장치를 등록한다.
        result = register_chrdev( GPIO_MAJOR, DEVICE_NAME, &gpio_fops );
        …
        printk(" Init madule, Succeed. This Device is %s and Major Number is
[%d]\n", DEVICE_NAME, GPIO_MAJOR);

        // GPIO 제어를 위한 GPIO 초기화
        GPIO_IO_Init();

        return 0; /* 성공 */
}
```

```
void cleanup_module(void)
{
        // 모듈을 해제한다..
        if ( !unregister_chrdev( GPIO_MAJOR, DEVICE_NAME) )
                printk("%s Device Exit Sucess...\n", DEVICE_NAME);
        else
                printk("%s Device Exit Fail...\n", DEVICE_NAME);
}
```

GPIO Driver (V)

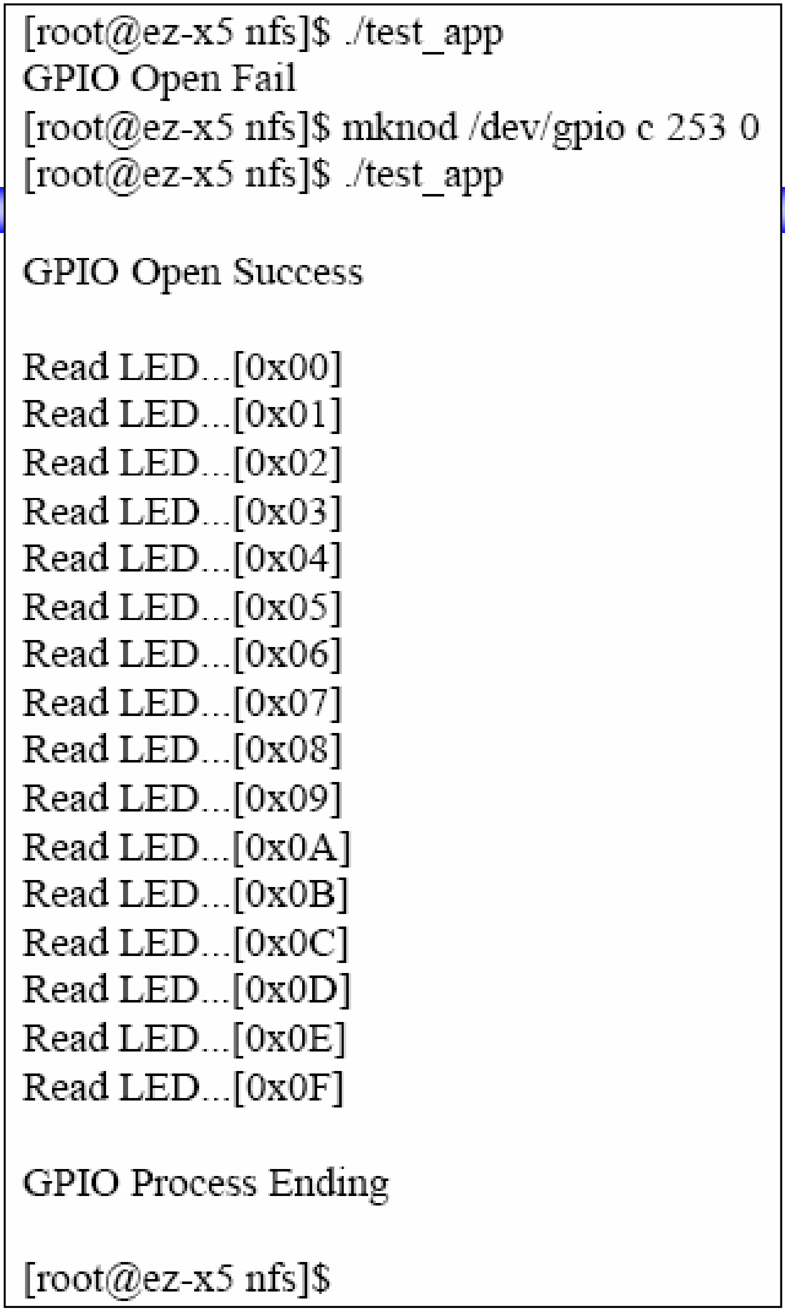
```
void GPIO_IO_Init( void )
{
            // 입력 정의
            GAFR0_L &= ~( GPIO_INPUT_MASK );        // Disable Alternative
Function
//          GPDR0 &= ~( GPIO_INPUT_MASK );          // 입력 전용으로 설정
            GRER0 &= ~( GPIO_INPUT_MASK );          // Clear Rising edge trigger.
            GFER0 &= ~( GPIO_INPUT_MASK );          // Set as Falling Edge Detect
}
```

GPIO Driver (VI)

```
ssize_t gpio_write(struct file *filp, const char *buf, size_t count, loff_t *f_pos )
{
            const unsigned char *gpiodata = buf;
            int data=0;

            get_user( data, gpiodata );
            gpio_outb( data );

            return count;
}

int gpio_outb( int data )
{
            // 출력전용
            GPDR0 |= (GPIO_OUPPUT_MASK);

            //GPSR은 출력 SET 레지스터
            GPSR0 |=  ( GPIO_OUTPUT_MASK );

            //GPCR은 출력 Clear 레지스터
            GPCR0  = GPCR0 | (data << 8);
            return 0;
}
```

GPIO Driver (VII)

- Installing GPIO Driver (III)
  - Download to EZ-X5
  - Install GPIO module
    - $ insmod gpio_dev.o
  - Set filesystem
    - $ mknod /dev/GPIO c 253 0
  - Check module
    - $ lsmod
  - Run application program
    - ./test_app →
  - Remove GPIO module
    - $ rmmod gpio_dev.o



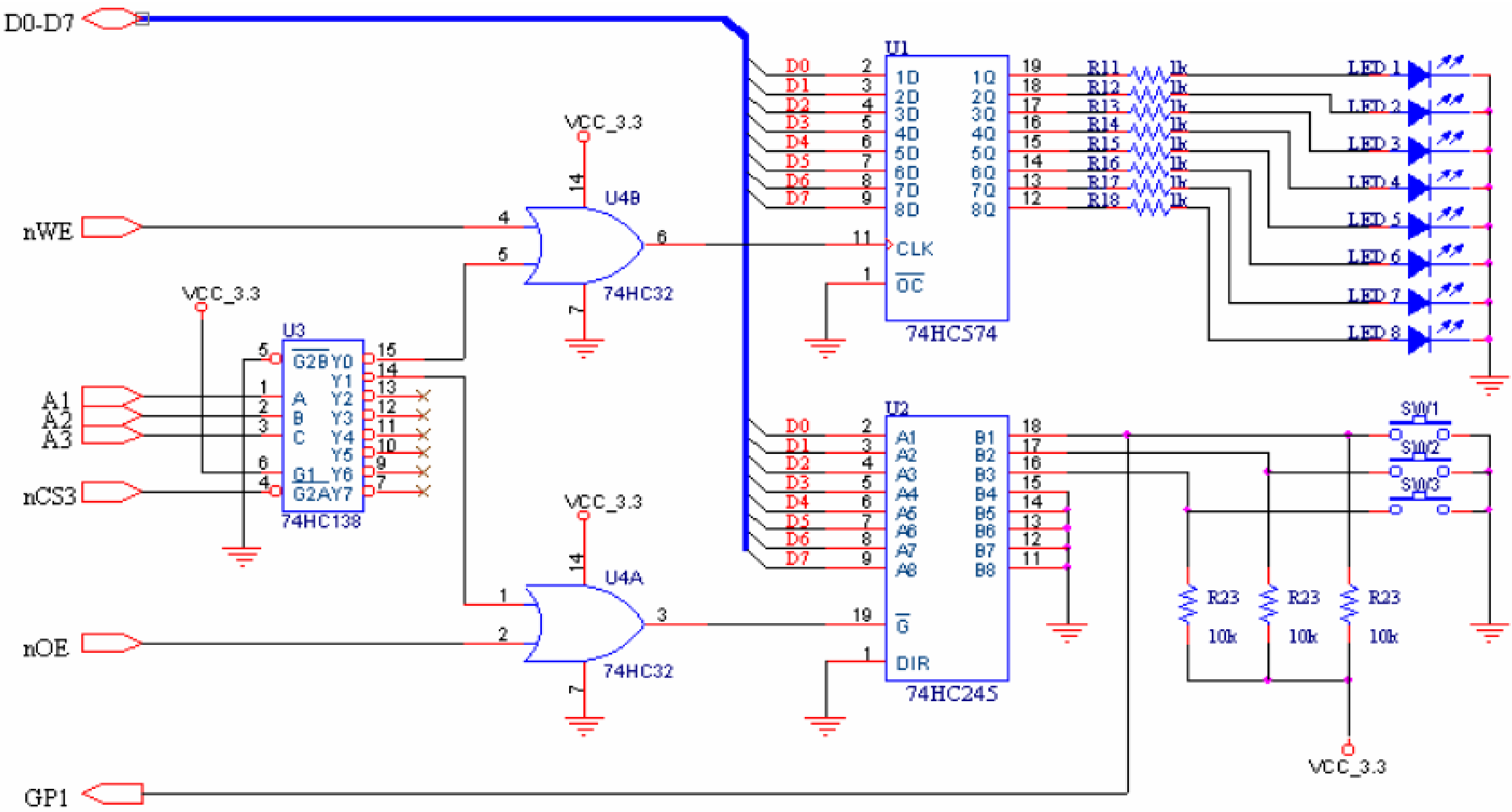
```
[root@ez-x5 nfs]$ ./test_app
GPIO Open Fail
[root@ez-x5 nfs]$ mknod /dev/gpio c 253 0
[root@ez-x5 nfs]$ ./test_app

GPIO Open Success

Read LED...[0x00]
Read LED...[0x01]
Read LED...[0x02]
Read LED...[0x03]
Read LED...[0x04]
Read LED...[0x05]
Read LED...[0x06]
Read LED...[0x07]
Read LED...[0x08]
Read LED...[0x09]
Read LED...[0x0A]
Read LED...[0x0B]
Read LED...[0x0C]
Read LED...[0x0D]
Read LED...[0x0E]
Read LED...[0x0F]

GPIO Process Ending

[root@ez-x5 nfs]$
```

7. DIO Driver

- Overview
  - EZ-X5 Baord에 add-on board 추가
    - 목표: LED lamp output and push-button switch input
  - Xscale의 데이터버스를 이용하여 LED 및 스위치 동작을 제어하는 방법
    - Xscale의 데이터버스는 32개의 입출력 신호 (D0-31)가 있으며, 입력신호로 할것인지, 출력신호로 할 것인지를 선택하는 1개의 선택신호 (nOE)가 있다.

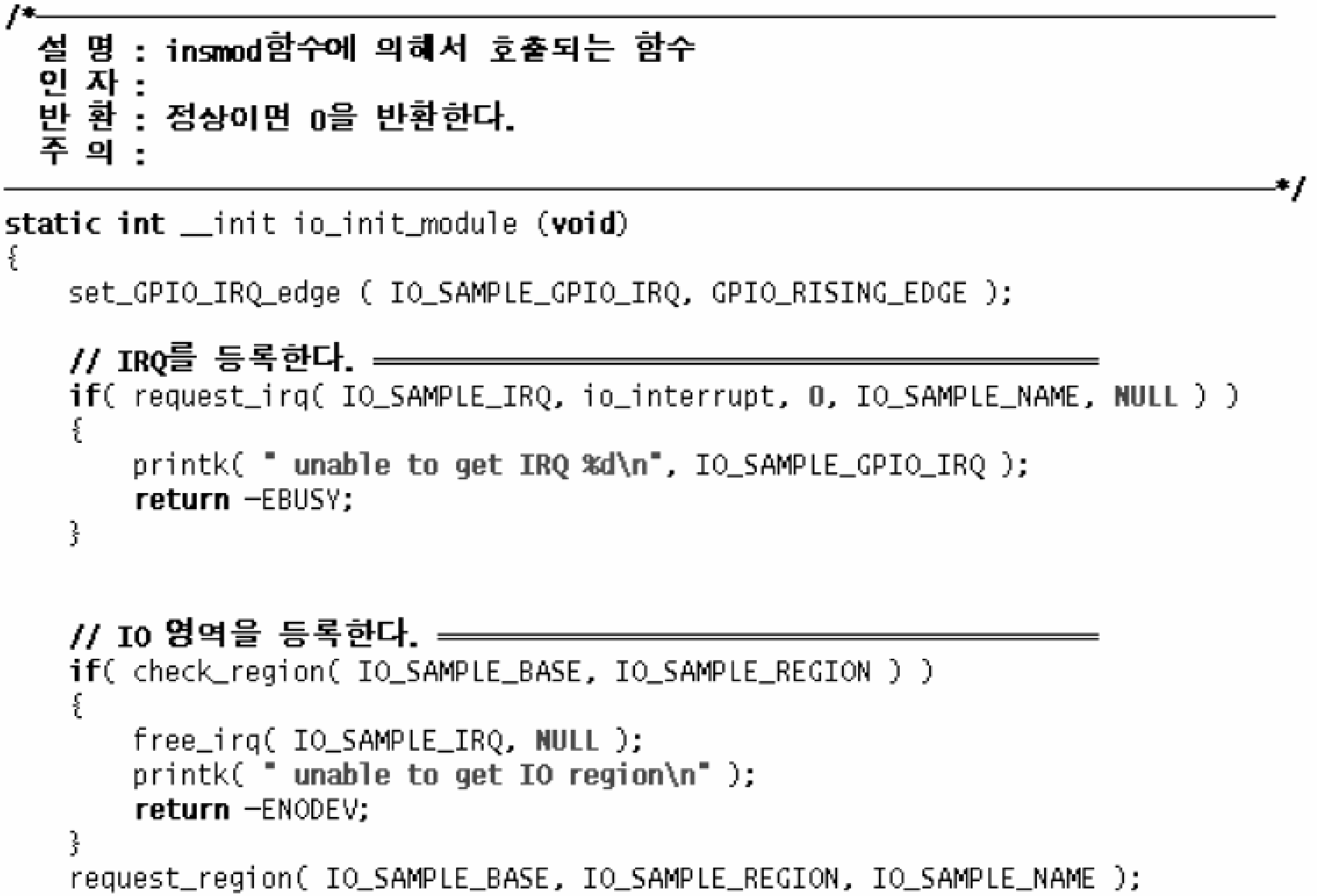
DIO Driver (II)

- Circuit diagram



DIO Driver (III)

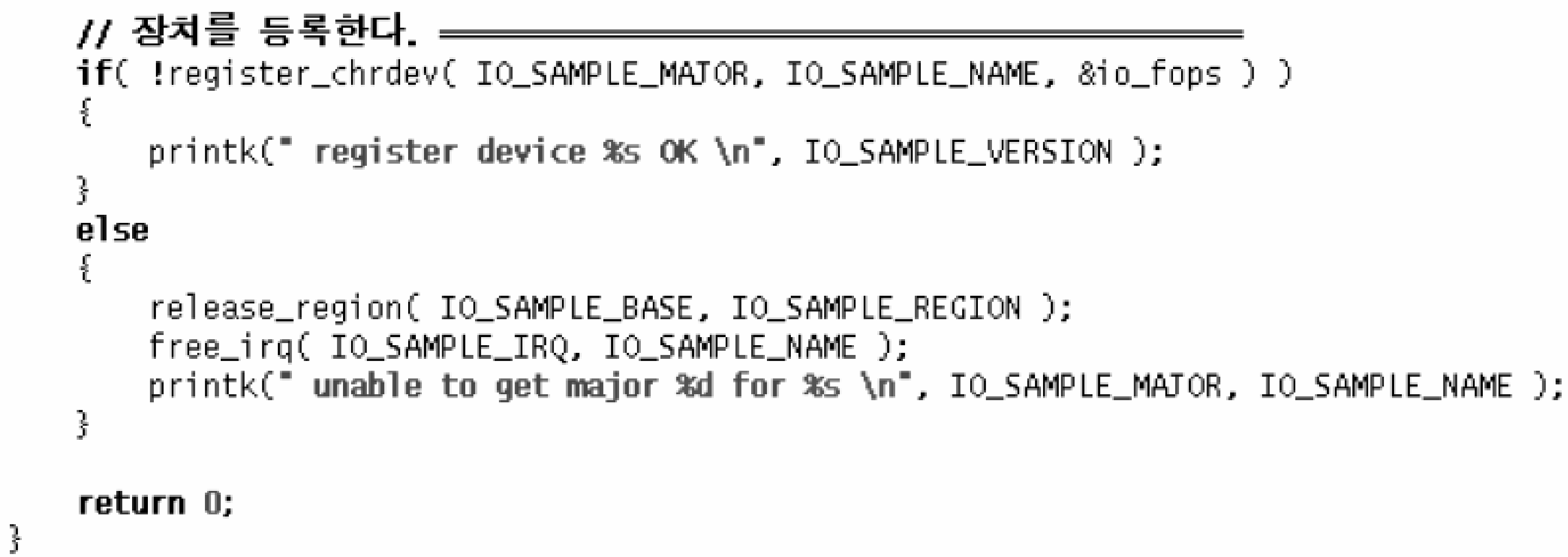
- Init_module



```
/*-----------------------------------------------------------------
  설 명 : insmod함수에 의해서 호출되는 함수
  인 자 :
  반 환 : 정상이면 0을 반환한다.
  주 의 :
-----------------------------------------------------------------*/
static int __init io_init_module (void)
{
    set_GPIO_IRQ_edge ( IO_SAMPLE_GPIO_IRQ, GPIO_RISING_EDGE );

    // IRQ를 등록한다. ===========================================
    if( request_irq( IO_SAMPLE_IRQ, io_interrupt, 0, IO_SAMPLE_NAME, NULL ) )
    {
        printk( " unable to get IRQ %d\n", IO_SAMPLE_GPIO_IRQ );
        return -EBUSY;
    }


    // IO 영역을 등록한다. =======================================
    if( check_region( IO_SAMPLE_BASE, IO_SAMPLE_REGION ) )
    {
        free_irq( IO_SAMPLE_IRQ, NULL );
        printk( " unable to get IO region\n" );
        return -ENODEV;
    }
    request_region( IO_SAMPLE_BASE, IO_SAMPLE_REGION, IO_SAMPLE_NAME );
```

DIO Driver (IV)

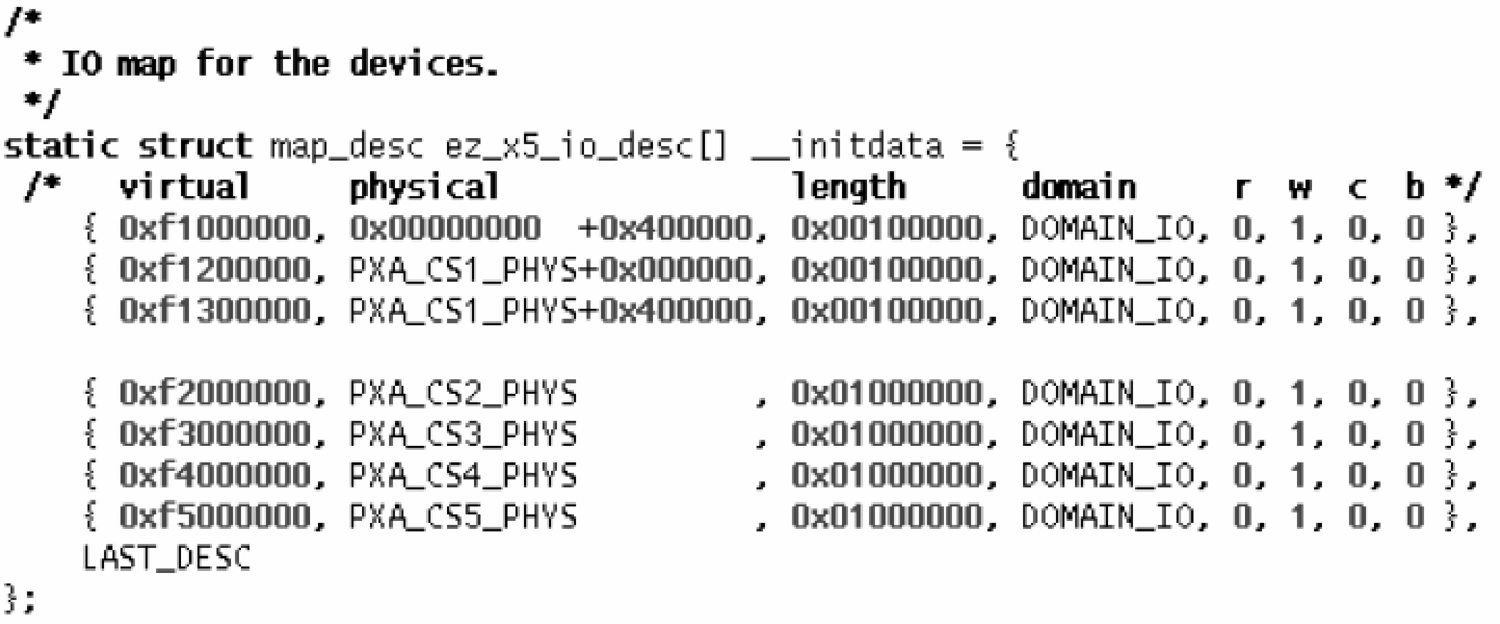
- Init_module (Cont'd)



```
    // 장치를 등록한다.
    if( !register_chrdev( IO_SAMPLE_MAJOR, IO_SAMPLE_NAME, &io_fops ) )
    {
        printk(" register device %s OK \n", IO_SAMPLE_VERSION );
    }
    else
    {
        release_region( IO_SAMPLE_BASE, IO_SAMPLE_REGION );
        free_irq( IO_SAMPLE_IRQ, IO_SAMPLE_NAME );
        printk(" unable to get major %d for %s \n", IO_SAMPLE_MAJOR, IO_SAMPLE_NAME );
    }

    return 0;
}
```

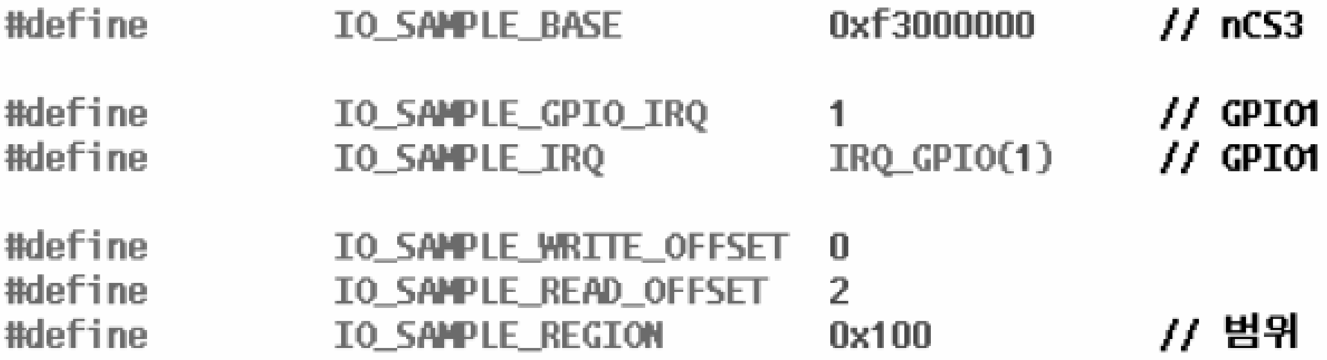
DIO Driver (V)

- I/O region in ez-x5.c



```
/*
 * IO map for the devices.
 */
static struct map_desc ez_x5_io_desc[] __initdata = {
 /*    virtual     physical                 length      domain     r  w  c  b */
    { 0xf1000000, 0x00000000  +0x400000, 0x00100000, DOMAIN_IO, 0, 1, 0, 0 },
    { 0xf1200000, PXA_CS1_PHYS+0x000000, 0x00100000, DOMAIN_IO, 0, 1, 0, 0 },
    { 0xf1300000, PXA_CS1_PHYS+0x400000, 0x00100000, DOMAIN_IO, 0, 1, 0, 0 },

    { 0xf2000000, PXA_CS2_PHYS         , 0x01000000, DOMAIN_IO, 0, 1, 0, 0 },
    { 0xf3000000, PXA_CS3_PHYS         , 0x01000000, DOMAIN_IO, 0, 1, 0, 0 },
    { 0xf4000000, PXA_CS4_PHYS         , 0x01000000, DOMAIN_IO, 0, 1, 0, 0 },
    { 0xf5000000, PXA_CS5_PHYS         , 0x01000000, DOMAIN_IO, 0, 1, 0, 0 },
    LAST_DESC
};
```

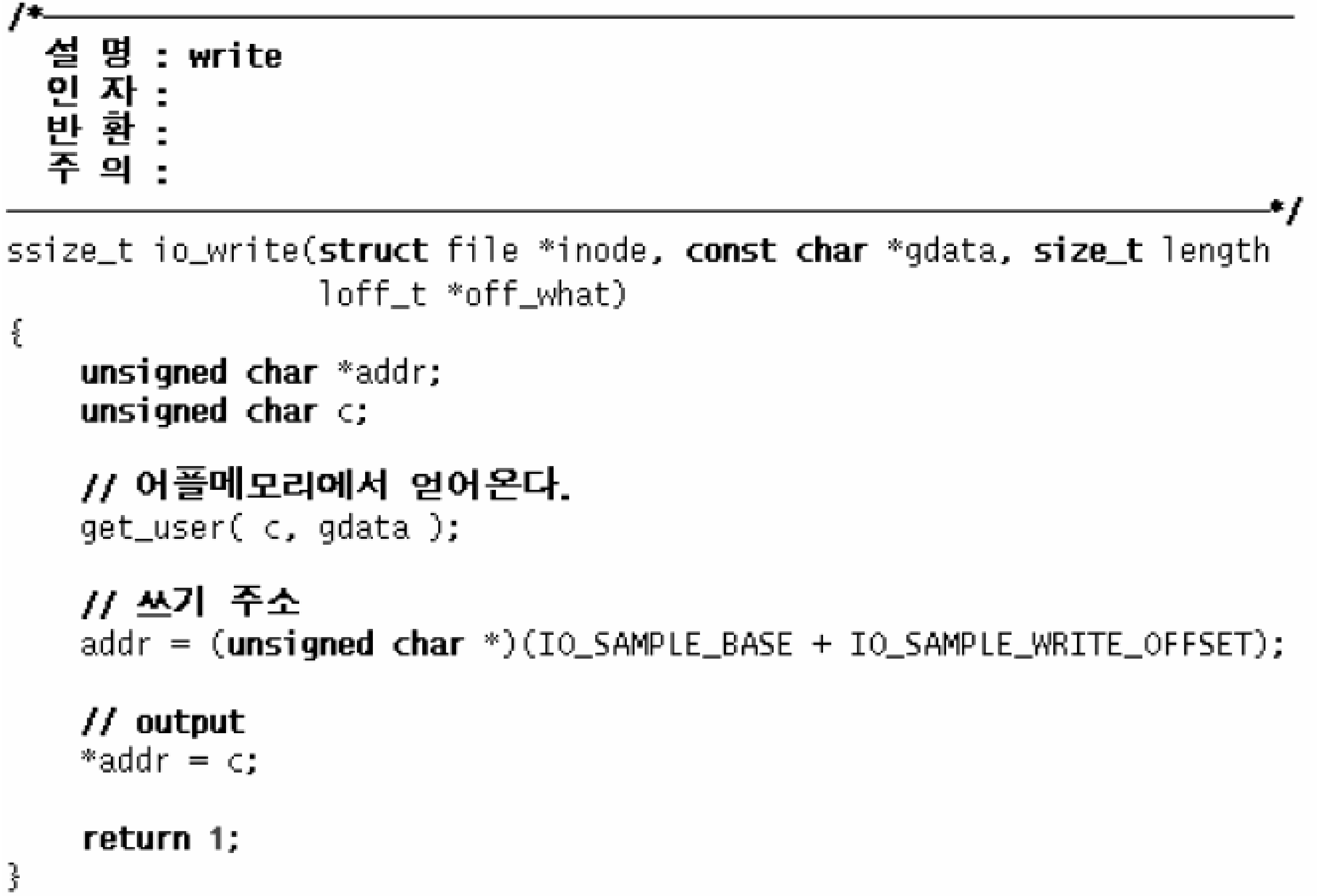
- I/O address in io.h



```
#define         IO_SAMPLE_BASE            0xf3000000        // nCS3

#define         IO_SAMPLE_GPIO_IRQ        1                 // GPIO1
#define         IO_SAMPLE_IRQ             IRQ_GPIO(1)       // GPIO1

#define         IO_SAMPLE_WRITE_OFFSET    0
#define         IO_SAMPLE_READ_OFFSET     2
#define         IO_SAMPLE_REGION          0x100             // 범위
```

DIO Driver (VI)

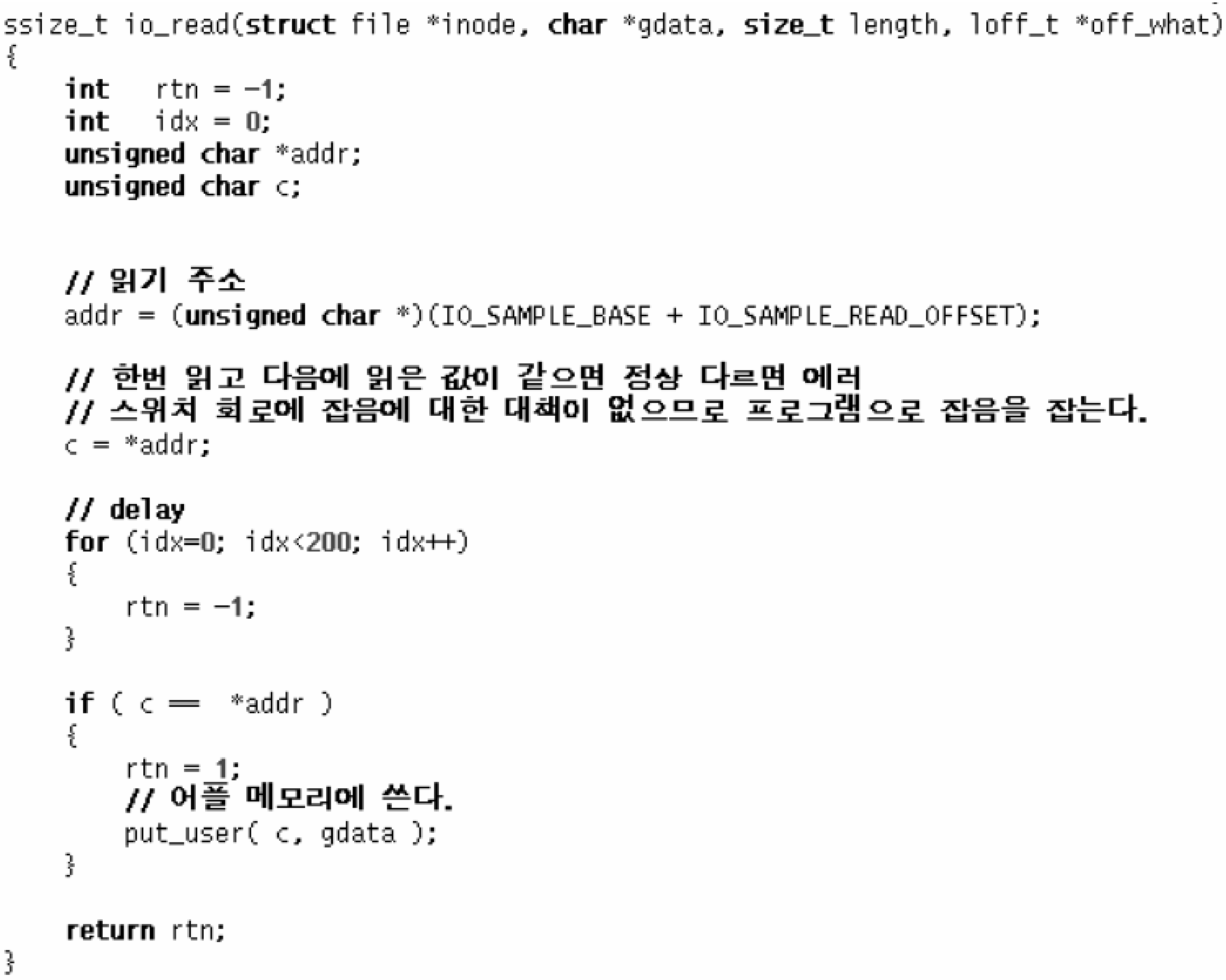
- Io_write function in io.c



```
/*--------------------------------------------------------------------------
  설 명 : write
  인 자 :
  반 환 :
  주 의 :
---------------------------------------------------------------------------*/
ssize_t io_write(struct file *inode, const char *gdata, size_t length
                 loff_t *off_what)
{
    unsigned char *addr;
    unsigned char c;

    // 어플메모리에서 얻어온다.
    get_user( c, gdata );

    // 쓰기 주소
    addr = (unsigned char *)(IO_SAMPLE_BASE + IO_SAMPLE_WRITE_OFFSET);

    // output
    *addr = c;

    return 1;
}
```

DIO Driver (VII)

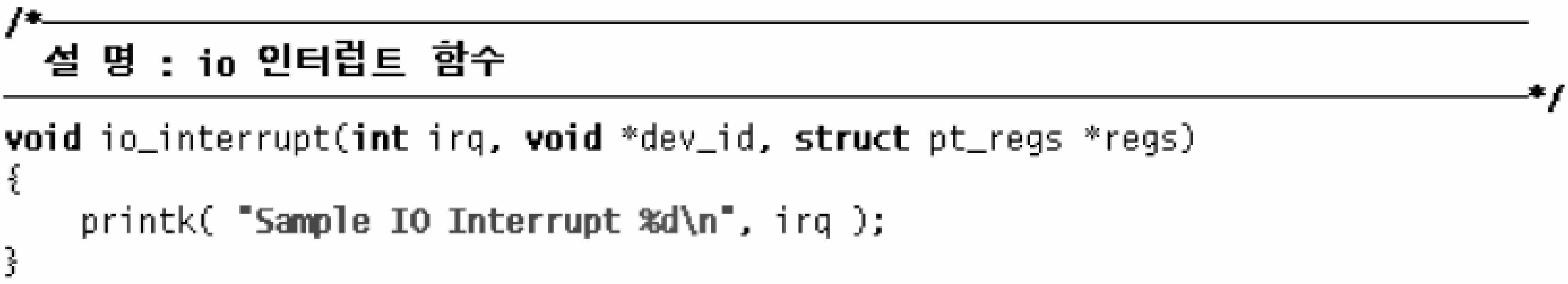
- Io_read function in io.c



```
ssize_t io_read(struct file *inode, char *gdata, size_t length, loff_t *off_what)
{
    int     rtn = -1;
    int     idx = 0;
    unsigned char *addr;
    unsigned char c;


    // 읽기 주소
    addr = (unsigned char *)(IO_SAMPLE_BASE + IO_SAMPLE_READ_OFFSET);

    // 한번 읽고 다음에 읽은 값이 같으면 정상 다르면 에러
    // 스위치 회로에 잡음에 대한 대책이 없으므로 프로그램으로 잡음을 잡는다.
    c = *addr;

    // delay
    for (idx=0; idx<200; idx++)
    {
        rtn = -1;
    }

    if ( c ==  *addr )
    {
        rtn = 1;
        // 어플 메모리에 쓴다.
        put_user( c, gdata );
    }

    return rtn;
}
```

DIO Driver (VIII)

- I/O interrupt routine
  - Each time you press the SW1!



```
/*
  설 명 : io 인터럽트 함수
                                                              */
void io_interrupt(int irq, void *dev_id, struct pt_regs *regs)
{
    printk( "Sample IO Interrupt %d\n", irq );
}
```

References

- GPIO Interface
  - PXA255 Developer's Manual, http://developer.intel.com
- GPIO Driver & DIO Driver
  - EZ-X5 User's Manual, http://www.falinux.com