Lab4. Interrupt

Applications using Switch/Timer Interrupt Drivers

fpga_int0 driver (1)

```
module_init(init_module2);
module_exit(close_module2);

struct file_operations FPGA_INT0_PS_Reg_FOPS =
{
	.owner = THIS_MODULE,
	.open = fpga_int0_ps_reg_open,
	.write = fpga_int0_ps_reg_write,
	.read = fpga_int0_ps_reg_read,
};

int init_module2(void)
{
	int FPGA_DEV;
	FPGA_DEV = register_chrdev(FPGA_INT0_MAJOR_NUMBER, FPGA_INT0_DRIVER_NAME,
	&FPGA_INT0_PS_Reg_FOPS);
	if(FPGA_DEV < 0)
	{
		printk(KERN_WARNING "Can't get any major\n");
		return FPGA_DEV;
	}
	fpga_int0_ps_reg_major = FPGA_INT0_MAJOR_NUMBER;
	fpga_int0_ps_reg_addr = ioremap(FPGA_INT0_PS_Reg_ADDRESS,0x2);
	printk("init module, %s major
	number: %d\n",FPGA_INT0_DRIVER_NAME,fpga_int0_ps_reg_major);
```

fpga_int0 driver (2)

```
	set_irq_type(IRQ_GPIO(83),IRQT_FALLING);
	if(request_irq(IRQ_GPIO(83),fpga_int0_interrupt,SA_INTERRUPT,FPGA_INT0_DRIVER_NA
	ME,NULL))
	{
		printk("FPGA INT 0 Interrupt init fail...!\n");
	}
	return 0;
}

void close_module2(void)
{
	if(unregister_chrdev(fpga_int0_ps_reg_major,FPGA_INT0_DRIVER_NAME))
	{
		printk(KERN_WARNING "%s DRIVER CLEANUP FALLED\n",
	FPGA_INT0_DRIVER_NAME);
	}
}

int fpga_int0_ps_reg_open(struct inode *fpga_int0_ps_reg_inode, struct file
	*fpga_int0_ps_reg_file)
{
	return 0;
}
```

fpga_int0 driver (3)

```
irqreturn_t fpga_int0_interrupt(int irq,void *dev_id,struct pt_regs *regs)
{
	printk("FPGA INT 0 Interrupt Success\n");
	kill_proc(PID_ID,SIGUSR1,1);
	return IRQ_HANDLED;
}

ssize_t fpga_int0_ps_reg_write(struct file *fpga_int0_ps_reg_inode,int *gdata, size_t length,
	loff_t *off_what)
{
	PID_ID = *gdata;
	return length;
}

ssize_t fpga_int0_ps_reg_read(struct file *fpga_int0_ps_reg_inode,int  *gdata, size_t length,
	loff_t *off_what)
{
	*gdata = inw(fpga_int0_ps_reg_addr);
	copy_to_user(gdata,gdata,sizeof(gdata));
	return length;
}
```

fpga_int0 application (1)

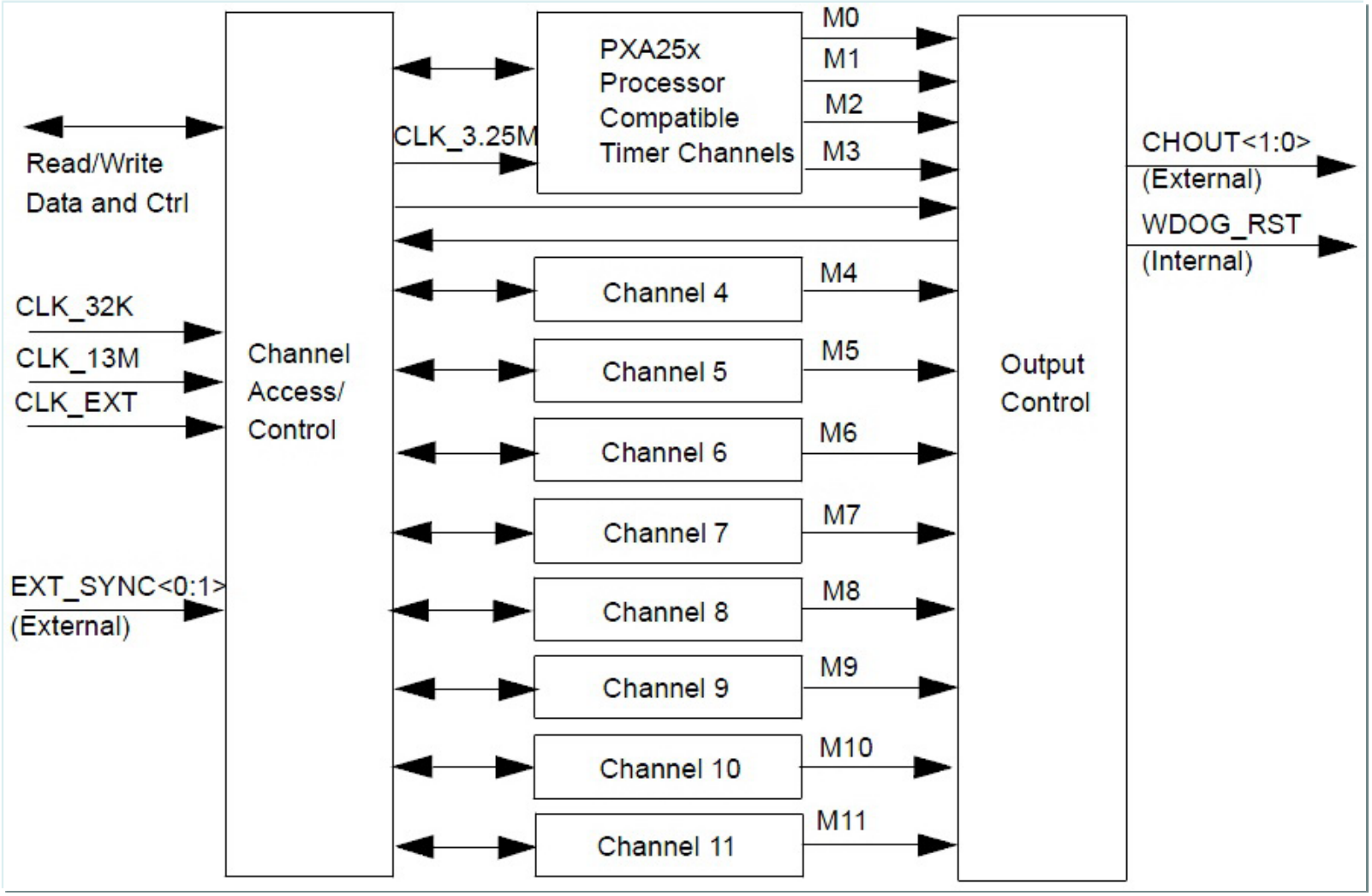
```
int INT0_DEV;
int INT0_DEV_PID;
int temp = 0;

void usrsignal(int sig)
{
	read(INT0_DEV,&temp,sizeof(temp));
	switch(temp)
	{
		case 0x001: if(temp == 0x001) printf("Push SW1\n"); break;
		case 0x002: if(temp == 0x002) printf("Push SW2\n"); break;
		case 0x004: if(temp == 0x004) printf("Push SW3\n"); break;
		case 0x008: if(temp == 0x008) printf("Push SW4\n"); break;
		case 0x010: if(temp == 0x010) printf("Push SW5\n"); break;
		case 0x020: if(temp == 0x020) printf("Push SW6\n"); break;
		case 0x040: if(temp == 0x040) printf("Push SW7\n"); break;
		case 0x080: if(temp == 0x080) printf("Push SW8\n"); break;
		case 0x100: if(temp == 0x100) printf("Push SW9\n"); break;
		case 0x200: if(temp == 0x200) printf("Push SW10\n"); break;
		case 0x400: if(temp == 0x400) printf("Push SW11\n"); break;
		case 0x800: if(temp == 0x800) printf("Push SW12\n"); break;
		default : printf(" What ? \n"); break;
	}
}
```

fpga_int0 application (2)

```
int main(void)
{
	system("clear");
	INT0_DEV = open("/dev/fpga_int0",O_RDWR);
	if(INT0_DEV < 0)
	{
		printf(" deriver open fail\n");
		return -1;
	}
	(void)signal(SIGUSR1,usrsignal);
	INT0_DEV_PID = getpid();
	write(INT0_DEV,&INT0_DEV_PID,sizeof(INT0_DEV_PID));
	printf("int0_test\n");
	printf("stop = Ctrl+C\n");
	while(1)
	{
	}
	close(INT0_DEV);
	return 0;
}
```

Operating System Timers



ostimer driver (1)

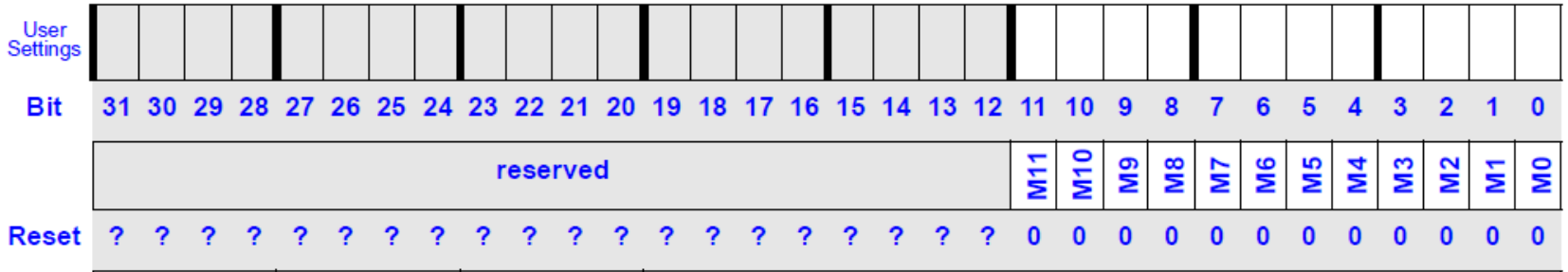
```
#define  DEVICE_NAME   "ostimer"
#define  OSTIMER_MAJOR        252

static int Major;
static int Device_Open = 0;
static int app_id;

static irqreturn_t ostimer_handler1(int irq, void *dev_id, struct pt_regs *regs )
{
      kill_proc(app_id,SIGUSR2,1);
      OSSR |= XLLP_OSSR_M4;
      return IRQ_HANDLED;
}
static int ostimer_ioctl(struct inode *inode, struct file *file,
                                          unsigned int cmd,unsigned long gdata)
{
      int id;
      copy_from_user(&id,(char *)gdata,4);
      app_id = id;
      return 0;
}
```

Table 22-10. OSSR Bit Definitions

0x40A0_0014 — OSSR — OS TImer



Bits	Access	Name	Description
31:12	reserved	—	reserved
11:0	R/W	M{*n*}	Match Status Channel *n* If OIER(*n*) is set then: 0 = No OSMR*n* match has occurred since the last clear. 1 = An OSMR*n* match has occurred since the last clear.

ostimer driver (2)

```
ssize_t ostimer_write (struct file *file,
                       const char *buffer,/* buffer */
                       size_t length,              /* length of buffer */
                       loff_t * offset)            /* offset in the file */
{
      return (length);
}
static struct file_operations led_fops =
{
      open:             ostimer_open,
      write:            ostimer_write,
      release:          ostimer_release,
      ioctl:            ostimer_ioctl,
};
static int __init ostimer_init(void)
{
 /*
  * Register the character device
  */
    int return_val1;
    Major = register_chrdev (OSTIMER_MAJOR, DEVICE_NAME, &led_fops);
    if (Major < 0) {
        printk ("ostimer init_module: failed with %d\n", OSTIMER_MAJOR);
        return Major;
    }
    printk ("ostimer driver registred: major = %d\n", OSTIMER_MAJOR);
```

ostimer driver (3)

```
    OMCR4 |=  XLLP_OMCR_CRES_1MSEC;
    OSMR4 = 9;
    OMCR4 |=  XLLP_OMCR_C;
    OMCR4 |=  XLLP_OMCR_P;
    OMCR4 |=  XLLP_OMCR_R;
    OIER |=  XLLP_OIER_E4;
    OSCR4 = 0; /* start OS timer 4 */

    set_irq_type( IRQ_OST_4_11, IRQT_FALLING );
    return_val1 = request_irq(IRQ_OST_4_11, ostimer_handler1, SA_INTERRUPT, "ostimer INT",
      NULL );
    if( return_val1 < 0 ) {
        printk(KERN_ERR "Acumen270_push_init() : Can't reqeust irq %#010x\n",
      IRQ_OST_4_11 );
        return -1;
    }
    return 0;
}

module_init(ostimer_init);
module_exit(ostimer_exit);
```

ostimer application (1)

```
int I_LED_DEV;
unsigned int counter = 0;

void usrsignal(int sig)
{     int temp = 31;
      counter++;
      if (counter >= 100) counter=0;
      if (counter == 0)
      {
            printf("On\n");
            temp=0;write(I_LED_DEV,&temp,sizeof(temp));
      }
      if (counter == 50)
      {
            printf("Off\n");
            temp=0xff;write(I_LED_DEV,&temp,sizeof(temp));
      }
}
```

ostimer application (2)

```
int main(void)
{
	int fd,id;
	unsigned char c;
	I_LED_DEV = open("/dev/fpga_led",O_RDWR);
	if(I_LED_DEV < 0)
	{
		printf(" fpga_led driver open fail\n");
		return -1;
	}
	fd = open("/dev/ostimer",O_WRONLY);
	if (fd < 0){
		printf("ostimer Device Open Error\n");
		exit(1);
	}
	 (void)signal(SIGUSR2,usrsignal);
	  id = getpid();
	 ioctl(fd,0,&id,4);
	for(;;){}
	close(fd);
}
```

Lab5. Digital Clock on Embedded Linux

Applications using Switch/Timer Interrupt Drivers

Exercise(1)

- **1/100** 초를 나타낼 수 있는 디지털 시계를 만든다.
- 현재 시각은 **8**자리 **(**시,분,초,**1/00**초 각 **2**자리씩**)**로 나타낸다.
- 프로그램이 시작되면 **“Enter Time”**메시지를 **LCD**에 내보낸 후 초기 설정 시각의 입력을 기다린다.
- 초기 시각은 **8**자리 **(**시,분,초,**1/00**초 각 **2**자리씩**)**로 **1**번 부터 **10**번까지의 버튼 키를 이용하여 입력한다**.(**숫자 **0**은 **10**번 버튼 사용**)** 한자리씩 입력할 때 마다 입력된 숫자가 **seven segment display**에 나타나야 한다**.** 새로 누른 숫자는 가장 오른쪽 자리에 나타나며**,** 다음 숫자를 누르면 왼쪽으로 한자리씩 이동한다**.** 버튼 누르는 횟수에 관계 없이 숫자는 계속 나타나야 한다**. Enter Key(11**번 버튼**)**를 누르면 초기 시각 입력이 끝나고 시계가 가기 시작한다.

Exercise(2)

- 시계가 가는 동안은 **“Running”** 메시지가 **LCD**에 나타난다**.**
- 시계가 가는 중 **12**번 버튼 키를 누르면 시계가 정지되고 **“Stopped”**메시지가 **LCD**에 나타난다**.** 다시 **12**번 버튼 키를 누르면 **“Running”**메시지가 **LCD**에 나타나고 시계가 간다**.**
- 시계가 가는 동안 **11**번 버튼 키를 누르면 시계가 정지되고 **“Enter Time”**메시지를 **LCD**에 내보낸 후 초기 설정 시각의 입력을 기다린다**.**
- 적어도 **2**개 이상의 **thread**를 사용한다**.**