Lab4. Interrupt

Applications using Timer Interrupt Drivers
**Enhanced Periodic Interrupt Timer (EPIT)**

- EPIT is a 32-bit set-and-forget timer that is capable of providing precise interrupts at regular intervals with minimal processor intervention. EPIT begins counting after it is enabled by software.
<table>
<thead>
<tr>
<th>Absolute address (hex)</th>
<th>Register name</th>
<th>Width (in bits)</th>
<th>Access</th>
<th>Reset value</th>
<th>Section/page</th>
</tr>
</thead>
<tbody>
<tr>
<td>20D_0000</td>
<td>Control register (EPIT1_CR)</td>
<td>32</td>
<td>R/W</td>
<td>0000_0000h</td>
<td>24.6.1/1216</td>
</tr>
<tr>
<td>20D_0004</td>
<td>Status register (EPIT1_SR)</td>
<td>32</td>
<td>R/W</td>
<td>0000_0000h</td>
<td>24.6.2/1219</td>
</tr>
<tr>
<td>20D_0008</td>
<td>Load register (EPIT1_LR)</td>
<td>32</td>
<td>R/W</td>
<td>FFFF_FFFFh</td>
<td>24.6.3/1219</td>
</tr>
<tr>
<td>20D_000C</td>
<td>Compare register (EPIT1_CMPR)</td>
<td>32</td>
<td>R/W</td>
<td>0000_0000h</td>
<td>24.6.4/1220</td>
</tr>
<tr>
<td>20D_0010</td>
<td>Counter register (EPIT1_CNR)</td>
<td>32</td>
<td>R</td>
<td>FFFF_FFFFh</td>
<td>24.6.5/1220</td>
</tr>
<tr>
<td>20D_4000</td>
<td>Control register (EPIT2_CR)</td>
<td>32</td>
<td>R/W</td>
<td>0000_0000h</td>
<td>24.6.1/1216</td>
</tr>
<tr>
<td>20D_4004</td>
<td>Status register (EPIT2_SR)</td>
<td>32</td>
<td>R/W</td>
<td>0000_0000h</td>
<td>24.6.2/1219</td>
</tr>
<tr>
<td>20D_4008</td>
<td>Load register (EPIT2_LR)</td>
<td>32</td>
<td>R/W</td>
<td>FFFF_FFFFh</td>
<td>24.6.3/1219</td>
</tr>
<tr>
<td>20D_400C</td>
<td>Compare register (EPIT2_CMPR)</td>
<td>32</td>
<td>R/W</td>
<td>0000_0000h</td>
<td>24.6.4/1220</td>
</tr>
<tr>
<td>20D_4010</td>
<td>Counter register (EPIT2_CNR)</td>
<td>32</td>
<td>R</td>
<td>FFFF_FFFFh</td>
<td>24.6.5/1220</td>
</tr>
</tbody>
</table>
Control Register (EPITx_CR)

Address: Base address + 0h offset

Bit 31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
R 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
W 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Reset 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Bit 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
R 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
W 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Reset 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

PRESCALAR

RLD 0 0
OCIE 0 0
ENMOD 0 0
EN 0 0

## Control Register (EPITx_CR)

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31–26</td>
<td>This read-only field is reserved and always has the value 0.</td>
</tr>
<tr>
<td></td>
<td><strong>Reserved</strong></td>
</tr>
<tr>
<td>25–24</td>
<td>Select clock source</td>
</tr>
<tr>
<td>CLKSRC</td>
<td>These bits determine which clock input is to be selected for running the counter. This field value should only be changed when the EPIT is disabled by clearing the EN bit in this register. For other programming requirements while changing clock source, refer to Change of Clock Source.</td>
</tr>
<tr>
<td>00</td>
<td>Clock is off</td>
</tr>
<tr>
<td>01</td>
<td>Peripheral clock</td>
</tr>
<tr>
<td>10</td>
<td>High-frequency reference clock</td>
</tr>
<tr>
<td>11</td>
<td>Low-frequency reference clock</td>
</tr>
<tr>
<td>23–22</td>
<td>EPIT output mode. This bit field determines the mode of EPIT output on the output pin.</td>
</tr>
<tr>
<td>OM</td>
<td>00 EPIT output is disconnected from pad</td>
</tr>
<tr>
<td></td>
<td>01 Toggle output pin</td>
</tr>
<tr>
<td></td>
<td>10 Clear output pin</td>
</tr>
<tr>
<td></td>
<td>11 Set output pin</td>
</tr>
<tr>
<td>21</td>
<td>EPIT stop mode enable. This read/write control bit enables the operation of the EPIT during stop mode. This bit is reset by a hardware reset and unaffected by software reset.</td>
</tr>
<tr>
<td>STOPEN</td>
<td>0 EPIT is disabled in stop mode</td>
</tr>
<tr>
<td></td>
<td>1 EPIT is enabled in stop mode</td>
</tr>
</tbody>
</table>
## Control Register (EPITx_CR)

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>20 Reserved</td>
<td>This read-only field is reserved and always has the value 0.</td>
</tr>
</tbody>
</table>
| 19 WAITEN | This read/write control bit enables the operation of the EPIT during wait mode. This bit is reset by a hardware reset. A software reset does not affect this bit.  
0 EPIT is disabled in wait mode  
1 EPIT is enabled in wait mode |
| 18 DBGEN | This bit is used to keep the EPIT functional in debug mode. When this bit is cleared, the input clock is gated off in debug mode. This bit is reset by hardware reset. A software reset does not affect this bit.  
0 Inactive in debug mode  
1 Active in debug mode |
| 17 IOVW | EPIT counter overwrite enable. This bit controls the counter data when the modulus register is written. When this bit is set, all writes to the load register overwrites the counter contents and the counter starts subsequently counting down from the programmed value.  
0 Write to load register does not result in counter value being overwritten.  
1 Write to load register results in immediate overwriting of counter value. |
## Control Register (EPITx_CR)

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
</table>
| **16** SWR (Software reset) | The EPIT is reset when this bit is set to 1. It is a self clearing bit. This bit is set when the block is in reset state and is cleared when the reset procedure is over. Setting this bit resets all the registers to their reset values, except for the EN, ENMOD, STOPEN, WAITEN and DBGEN bits in this control register.  
  
  | 0 | EPIT is out of reset  
  | 1 | EPIT is undergoing reset |
| **15–4** PRESCALAR (Counter clock prescaler value) | This bit field determines the prescaler value by which the clock is divided before it goes to the counter.  
  
  | 0x000 | Divide by 1  
  | 0x001 | Divide by 2...  
  | 0xFFF | Divide by 4096 |
| **3** RLD (Counter reload control) | This bit is cleared by hardware reset. It decides the counter functionality, whether to run in free-running mode or set-and-forget mode.  
  
  | 0 | When the counter reaches zero it rolls over to 0xFFFF_FFFF (free-running mode)  
  | 1 | When the counter reaches zero it reloads from the modulus register (set-and-forget mode) |
## Control Register \((EPITx\_CR)\)

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>OCIE&lt;sub&gt;N&lt;/sub&gt;</td>
</tr>
<tr>
<td></td>
<td>Output compare interrupt enable.</td>
</tr>
<tr>
<td></td>
<td>This bit enables the generation of interrupt on occurrence of compare event.</td>
</tr>
<tr>
<td>0</td>
<td>Compare interrupt disabled</td>
</tr>
<tr>
<td>1</td>
<td>Compare interrupt enabled</td>
</tr>
<tr>
<td>1</td>
<td>ENMOD</td>
</tr>
<tr>
<td></td>
<td>EPIT enable mode.</td>
</tr>
<tr>
<td></td>
<td>When EPIT is disabled ((EN=0)), both main counter and prescaler counter freeze their count at current count values. ENMOD bit is a r/w bit that determines the counter value when the EPIT is enabled again by setting EN bit. If ENMOD bit is set, then main counter is loaded with the load value ((If\ RLD=1)) / (0xFFFF_FFFF) ((If\ RLD=0)) and prescaler counter is reset, when EPIT is enabled ((EN=1)). If ENMOD is programmed to 0 then both main counter and prescaler counter restart counting from their frozen values when EPIT is enabled ((EN=1)). If EPIT is programmed to be disabled in a low-power mode ((STOP/WAIT/DEBUG)), then both the main counter and the prescaler counter freeze at their current count values when EPIT enters low-power mode. When EPIT exits the low-power mode, both main counter and prescaler counter start counting from their frozen values irrespective of the ENMOD bit. This bit is reset by a hardware reset. A software reset does not affect this bit.</td>
</tr>
<tr>
<td>0</td>
<td>Counter starts counting from the value it had when it was disabled.</td>
</tr>
<tr>
<td>1</td>
<td>Counter starts count from load value ((RLD=1)) or (0xFFFF_FFFF) ((If\ RLD=0))</td>
</tr>
<tr>
<td>0</td>
<td>EN</td>
</tr>
<tr>
<td></td>
<td>This bit enables the EPIT. EPIT counter and prescaler value when EPIT is enabled ((EN = 1)), is dependent upon ENMOD and RLD bit as described for ENMOD bit. It is recommended that all registers be properly programmed before setting this bit. This bit is reset by a hardware reset. A software reset does not affect this bit.</td>
</tr>
<tr>
<td>0</td>
<td>EPIT is disabled</td>
</tr>
<tr>
<td>1</td>
<td>EPIT is enabled</td>
</tr>
</tbody>
</table>
EPIT clock source

- 24MHz*22=528MHz
- 528MHz/(4*2)=66MHz
The Clock Control Module (CCM) generates and controls clocks to the various modules in the design and manages low power modes. This module uses the available clock sources to generate the clock roots.

The Clock Controller Module controls the following functions:

- Uses the available clock sources to generate clock roots to various parts of the chip:
  - PLL1 also referenced as ARM PLL
  - PLL2 also referenced as System PLL
  - PLL3 also referenced as USB1 PLL
  - PLL4 also referenced as Audio PLL
  - PLL5 also referenced as Video PLL
  - PLL6 also referenced as ENET PLL
  - PLL7 also referenced as USB2 PLL (This PLL is only used by the USB UTM interface through a direct connection.)
  - PLL8 also referenced as MLB PLL

- Uses programmable bits to control frequencies of the clock roots.
- Controls the low power mechanism.
- Provides control signals to LPCG for gating clocks.
- Provides handshake with SRC for reset performance.
- Provides handshake with GPC for support of power gating operations.
CCM Clock Tree
## CCM Clock Gating Register 1 (CCM_CCGR1)

**Address:** 20C_4000h base + 6Ch offset = 20C_406Ch

<table>
<thead>
<tr>
<th>Bit</th>
<th>CG15</th>
<th>CG14</th>
<th>CG13</th>
<th>CG12</th>
<th>CG11</th>
<th>CG10</th>
<th>CG9</th>
<th>CG8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reset</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bit</th>
<th>CG7</th>
<th>CG6</th>
<th>CG5</th>
<th>CG4</th>
<th>CG3</th>
<th>CG2</th>
<th>CG1</th>
<th>CG0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reset</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>17–16</td>
<td>esai clocks (esai_clk_enable)</td>
</tr>
<tr>
<td>CG8</td>
<td></td>
</tr>
<tr>
<td>15–14</td>
<td>epit2 clocks (epit2_clk_enable)</td>
</tr>
<tr>
<td>CG7</td>
<td></td>
</tr>
<tr>
<td>13–12</td>
<td>epit1 clocks (epit1_clk_enable)</td>
</tr>
<tr>
<td>CG6</td>
<td></td>
</tr>
<tr>
<td>11–10</td>
<td>enet clock (enet_clk_enable)</td>
</tr>
<tr>
<td>CG5</td>
<td></td>
</tr>
<tr>
<td>9–8</td>
<td>ecspi5 clocks (ecspi5_clk_enable)</td>
</tr>
<tr>
<td>CG4</td>
<td></td>
</tr>
<tr>
<td>7–6</td>
<td>ecspi4 clocks (ecspi4_clk_enable)</td>
</tr>
<tr>
<td>CG3</td>
<td></td>
</tr>
<tr>
<td>5–4</td>
<td>ecspi3 clocks (ecspi3_clk_enable)</td>
</tr>
<tr>
<td>CG2</td>
<td></td>
</tr>
<tr>
<td>3–2</td>
<td>ecspi2 clocks (ecspi2_clk_enable)</td>
</tr>
<tr>
<td>CG1</td>
<td></td>
</tr>
<tr>
<td>CG0</td>
<td>ecspi1 clocks (ecspi1_clk_enable)</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
#include <linux/module.h>
#include <linux/cdev.h>
#include <linux/fs.h>
#include <linux/gpio.h>
#include <linux/delay.h>
#include <linux/kdev_t.h>
#include <linux/interrupt.h>
#include <linux/signal.h>
#include <linux/sched.h>

#include <asm/io.h>
#include <mach/epit.h>

MODULE_LICENSE("GPL");

static int epit_major=0, epit_minor=0;
static int result, res;
static dev_t epit_dev;
static struct cdev epit_cdev;
int dev_pid;

__iomem void *epit_base;
__iomem void *ccm_ccgr1_address;
void kill_proc(int pid, int sig)
{
    struct task_struct* p;
    struct task_struct* t = NULL;
    struct pid* pspid;
    rcu_read_lock();
    p = &init_task;
    do {
        if (p->pid == pid){
            t = p;
            break;
        }
        p=next_task(p);
    } while(p != &init_task);
    if(t != NULL) {
        pspid = t->pids[PIDTYPE_PID].pid;
        if (pspid != NULL) kill_pid(pspid,sig,1);
    }
    rcu_read_unlock();
}

static irqreturn_t epit_interrupt(int irq, void *dev_id, struct pt_regs *regs)
{
    printk(KERN_ALERT"EPIT interrupt!\n ");
    kill_proc(dev_pid,SIGUSR1);
    writel(EPITSR_OCIF, epit_base+EPITSR); /* epit_irq_acknowledge */
    return IRQ_HANDLED;
static int epit_init(void)
{
    printk(KERN_ALERT "< EPIT Module is up > \n");
    if((result = epit_register_cdev())<0)
    {
        printk(KERN_ALERT "< EPIT Register Fail > \n");
        return result;
    }
    res = interrupt_init();
    if(res < 0)
        return -1;
    return 0;
}

static void epit_exit(void)
{
    printk(KERN_ALERT "< EPIT Module is down > \n");
    free_irq(MXC_INT_EPIT1, NULL);
    cdev_del(&epit_cdev);
    unregister_chrdev_region(epit_dev,1);
int interrupt_init(void)
{
    int rtc;

    ccm_ccgr1_address = ioremap(0x020c406c, SZ_16K);
    writel(readl(ccm_ccgr1_address) | 0xf000, ccm_ccgr1_address); /* epit clock enable */
    epit_base = ioremap(0x020d0000, SZ_16K); /* epit base address */
    writel(EPITCR_EN | EPITCR_ENMOD | EPITCR_OCIEN | EPITCR_RLD | EPITCR_IOVW |
            EPITCR_CLKSRC_PERIPHERAL, epit_base+EPITCR);
    writel(660000, epit_base+EPITLR);
    writel(660000, epit_base+EPITCMPR);

    rtc = request_irq(MXC_INT_EPIT1, (void*)epit_interrupt, IRQF_DISABLED , "epit", NULL);
    if(rtc) {
        printk(KERN_ALERT "<EPIT irq Register Fail>
        goto fail;
    }

    printk(KERN_INFO "EPIT Enable\n");
    return 0;
fail:
    return -1;
}
static int epit_register_cdev(void) {
    int error;

    if(epit_major) {
        epit_dev = MKDEV(epit_major, epit_minor);
        error = register_chrdev_region(epit_dev, 1, "epit");
    } else {
        error = alloc_chrdev_region(&epit_dev, epit_minor, 1, "epit");
        epit_major = MAJOR(epit_dev);
    }

    if(error<0) {
        printk(KERN_WARNING "EPIT: can't get major %d\n", epit_major);
        return result;
    }

    printk(KERN_ALERT "major number = %d\n", epit_major);

    cdev_init(&epit_cdev, &epit_fops);
    epit_cdev.owner = THIS_MODULE;
    epit_cdev.ops = &epit_fops;
    error = cdev_add(&epit_cdev, epit_dev, 1);

    if(error) {
        printk(KERN_NOTICE "EPIT Register Error %d\n", error);
    }

    return 0;
}
```c
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <fcntl.h>
#include <sys/ioctl.h>
#include <signal.h>

int dev_pid;
int counter=0;
int sec=0, min=0;

void usrsignal(int sig)
{
    counter++;
    if (counter >= 100)
    {
        sec++;
        if (sec >= 60)
        {
            min++;
        }
    }
    counter=0;
    printf("%d %d\n", min, sec);
}
```
int main(void)
{
    int fd;
    int retn;
    char buf[10] = {0};
    int loop = 0;

    (void)signal(SIGUSR1, usrsignal);
    fd = open("/dev/epit", O_RDWR);
    printf("fd = %d\n", fd);
    if(fd<0){
        perror("/dev/epit error");
        exit(-1);
    }else{
        printf("< epit device has been detected >\n");
    }
    dev_pid=getpid();
    printf("dev_pid=%d\n", dev_pid);
    buf[0] = 0xff & dev_pid;
    buf[1] = 0xff & (dev_pid >> 8);
    write(fd,buf,sizeof(dev_pid));

    while(1){
        /*printf("%d\n",buf[0]);
        usleep(10000);
        //for(loop = 0; loop < 100000; loop++) {*/
    }
    close(fd);
    return 0;
}
초(second)값과 1/100 초를 나타낼 수 있는 stopwatch를 만든다.
시간은 4자리 (초, 1/00초 각 2자리씩)로 seven segment display에 나타낸다.
프로그램이 시작되면 0000을 디스플레이하고, 버튼 입력을 기다린다.
버튼을 누르면 stopwatch가 가기(go) 시작한다.
Stopwatch가 가는 중 버튼을 누르면 stopwatch가 정지(stop)한다.
버튼 입력을 thread, 디스플레이용 thread 등 적어도 2개 이상의 thread를 사용한다.
Exercise

- Clear Button을 누르면 stopwatch를 정지하고, 0000을 디스플레이한 후, 버튼 입력을 기다린다.
- Stopwatch의 stop & go 는 Clear Button을 제외한 8개의 버튼 중 아무 버튼을 눌러도 동작해야 한다.