practice06

Kernel porting | Timer Settings & Implement Hardware Interrupt Handler

TIMER

Code Implementation

timer.c

Change the timer to be accessible in USER mode.

```
void vh_timer_init(void)
{
    write_cntkctl(CNTKCTL_PL0P); // change the timer to be accessible in USER mode.
    vk_timer_flag = 0;
}
```

kernel_start.c

- Enter 1/16 of the current value in `CNTP_TVAL` register, through `write_cntp_tval()`
- Read 10 times over, with printing `CTNP_TVAL` value through `read_cntp_tval()`
- Read the value of physical counter through `read_cntpct()`, after exiting the loop

```
void TIMER_test() {
    int timebuf[10];
    int i;

    write_cntp_tval(read_cntp_tval() / 16);

    for (i = 0; i < 10; i++) {
        timebuf[i] = read_cntp_tval();
        printk("timebuffer[%d] = %d\n", i, timebuf[i]);
    }

    read_cntpct();
}

void VPOS_kernel_main( void )
{
...
    //Timer Test...
    TIMER_test();
...
}
```

Result

```
2021011158> go 40000000
## Starting application at 0x40000000 ...

*******************************************************************************
*  QURIX version 3.0    xx/10/2012                                            *
*******************************************************************************
timebuffer[0] = -119033345
timebuffer[1] = -119120395
timebuffer[2] = -119124574
timebuffer[3] = -119128368
timebuffer[4] = -119132145
timebuffer[5] = -119135904
timebuffer[6] = -119139660
timebuffer[7] = -119143408
timebuffer[8] = -119147645
timebuffer[9] = -119151529

Race condition value = 0

Shell>typing is still avaliable
```

Hardware Interrupt

Code Implementation

Set GIC Register address

vh_io_hal.h

```
/*****************************************************************
   GIC address
 *****************************************************************/

#define GICD_BASE_ADDR      0x08000000
#define GICC_BASE_ADDR   0x08010000

#define GICD_CTRL      (*(volatile unsigned int*)(GICD_BASE_ADDR + 0x0))
#define GICD_ISENABLER(x)    (*(volatile unsigned int*)(GICD_BASE_ADDR + 0x100 + x*4))
#define GICD_ICENABLER(x)    (*(volatile unsigned int*)(GICD_BASE_ADDR + 0x180 + x*4))
#define GICD_ISPENDR(x)    (*(volatile unsigned int*)(GICD_BASE_ADDR + 0x200 + x*4))
#define GICD_ICPENDR(x)    (*(volatile unsigned int*)(GICD_BASE_ADDR + 0x280 + x*4))
#define GICD_ICACTIVER(x)    (*(volatile unsigned int*)(GICD_BASE_ADDR + 0x380 + x*4))
#define GICD_ITARGETSR(x)    (*(volatile unsigned int*)(GICD_BASE_ADDR + 0x800 + x*4))
#define GICD_ICFGR(x)    (*(volatile unsigned int*)(GICD_BASE_ADDR + 0xC00 + x*4))

#define GICC_CTRL      (*(volatile unsigned int*)(GICC_BASE_ADDR + 0x0))
#define GICC_PMR       (*(volatile unsigned int*)(GICC_BASE_ADDR + 0x4))
#define GICC_IAR       (*(volatile unsigned int*)(GICC_BASE_ADDR + 0xC))
#define GICC_EOIR      (*(volatile unsigned int*)(GICC_BASE_ADDR + 0x10))

#define INTERRUPT_ID_SGI_BEGIN 0
#define INTERRUPT_ID_PPI_BEGIN 16
#define INTERRUPT_ID_SPI_BEGIN 32
#define INTERRUPT_ID_UART (INTERRUPT_ID_SPI_BEGIN + 1)
#define INTERRUPT_ID_TIMER 30    // physical timer
#define INTERRUPT_ID_END 1019
```

- Set GIC with checking `qemu/hw/arm/virt.c`

```
static const MemMapEntry base_memmap[] = {
    /* Space up to 0x8000000 is reserved for a boot ROM */
    [VIRT_FLASH] =              {        0, 0x08000000 },
    [VIRT_CPUPERIPHS] =       { 0x08000000, 0x00020000 },
    /* GIC distributor and CPU interfaces sit inside the CPU peripheral space */
    [VIRT_GIC_DIST] =          { 0x08000000, 0x00010000 },
    [VIRT_GIC_CPU] =           { 0x08010000, 0x00010000 },
```

```
    [VIRT_GIC_V2M] =          { 0x08020000, 0x00001000 },
    [VIRT_GIC_HYP] =          { 0x08030000, 0x00010000 },
    [VIRT_GIC_VCPU] =         { 0x08040000, 0x00010000 },
    /* The space in between here is reserved for GICv3 CPU/vCPU/HYP */
    [VIRT_GIC_ITS] =          { 0x08080000, 0x00020000 },
    /* This redistributor space allows up to 2*64kB*123 CPUs */
    [VIRT_GIC_REDIST] =       { 0x080A0000, 0x00F60000 },
    ...
};
```

Enable interrupt

kernel_start.c

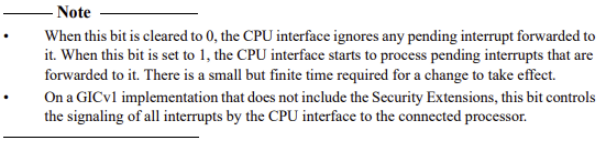
```
void set_interrupt(void)
{
    // disable  interrupts
    GICC_CTRL &= 0;
    GICD_CTRL &= 0;

    // interrupt setting
    vh_serial_irq_enable();

    // set priority mask to the lowest level (to accept interrupts of any priority level)
    GICC_PMR = 0xff;

    // enable interrupts
    GICC_CTRL |= 1;
    GICD_CTRL |= 1;
}
```

- Set `GICC_CTRL` control through referencing 'ARM Generic Interrupt Controller Architecture Specification' (p.125~126)



Note

- When this bit is cleared to 0, the CPU interface ignores any pending interrupt forwarded to it. When this bit is set to 1, the CPU interface starts to process pending interrupts that are forwarded to it. There is a small but finite time required for a change to take effect.
- On a GICv1 implementation that does not include the Security Extensions, this bit controls the signaling of all interrupts by the CPU interface to the connected processor.

serial.c

```
char getc(void)
{
    // char c;
    // unsigned long rxstat;

    // /* Write getc func */
    // while (UARTFR & UARTFR_RXFE);

    // c = (char) UARTDR;

    // rxstat = UARTRSR & UART_ERR_MASK;

    /* End getc func */

    while (pop_idx == push_idx);

    char c = serial_buff[pop_idx++];

    if (pop_idx == SERIAL_BUFF_SIZE)
        pop_idx = 0;
```

```
    return c;
}

void vh_serial_init(void)
{
    // set baud rate
    unsigned int idiv, fdiv;
    /*  baud rate Here  */
    float divisor = (float) UART_CLK / (16 * UART_BAUDRATE);
    idiv = (unsigned int) divisor;
    fdiv = (unsigned int) ((divisor - idiv) * 64 + 0.5);

    /*  baud rate End  */
    UARTIBRD = idiv;
    UARTFBRD = fdiv;

    // set UART ctrl regs
    UARTLCR_H = ~0x10;
    UARTCR = 0x301;
    // UARTIMSC = 0xF9EF;
    // UARTIFLS = 0x4;

    // clear buffer
    push_idx = 0;
    pop_idx = 0;
    for(int i=0; i<SERIAL_BUFF_SIZE; i++)
        serial_buff[i] = '\0';
}

void vh_serial_irq_enable(void)
{
    /* enable GIC & interrupt */

    // clear active & pending status
    GICD_ICPENDR(INTERRUPT_ID_UART / 32) = (1 << (INTERRUPT_ID_UART % 32));
  GICD_ICACTIVER(INTERRUPT_ID_UART / 32) = (1 << (INTERRUPT_ID_UART % 32));

    // enable interrupt
    GICD_ISENABLER(INTERRUPT_ID_UART / 32) |= (1 << (INTERRUPT_ID_UART % 32));

    // set interrupt target (to cpu0)
    GICD_ITARGETSR(INTERRUPT_ID_UART / 4) |= 1 << ((INTERRUPT_ID_UART % 4) * 8);

    // set interrupt triggering type (edge-triggering)
    GICD_ICFGR(INTERRUPT_ID_UART / 16) |= 0x2 << ((INTERRUPT_ID_UART % 16) * 2);

    // clear uart interrupts
    UARTICR |= 0x1ff;

    // enable UART interrupts Mask RX
    UARTIMSC |= (1 << 4);
}
```

- Reference from 'ARM Generic Interrupt Controller Architecture Specification' to implement `vh_serial_irq_enable()` :
  1. clear active & pending status
     - Set the corresponding bit to 1

Table 4-12 GICD_ICPENDR bit assignments

Bits	Name	Function		
[31:0]	Clear-pending bits	For each bit:		
		Reads	**0**	The corresponding interrupt is not pending on any processor.
			1	• For SGIs and PPIs, the corresponding interrupt is pending[a] on this processor. • For SPIs, the corresponding interrupt is pending[a] on at least one processor.

For interrupt ID *m*, when DIV and MOD are the integer division and modulo operations:

- the corresponding GICD_ICPENDR number, *n*, is given by *n* = *m* DIV 32
- the offset of the required GICD_ICPENDR is (0x280 + (4*n))
- the bit number of the required Set-pending bit in this register is *m* MOD 32.

2. enable interrupt
   - Set the corresponding bit to 1.

Table 4-9 GICD_ISENABLER bit assignments

Bits	Name	Function		
[31:0]	Set-enable bits	For SPIs and PPIs, each bit controls the forwarding of the corresponding interrupt from the Distributor to the CPU interfaces:		
		Reads	**0**	Forwarding of the corresponding interrupt is disabled.
			1	Forwarding of the corresponding interrupt is enabled.

For interrupt ID *m*, when DIV and MOD are the integer division and modulo operations:

- the corresponding GICD_ISENABLER number, *n*, is given by *n* = *m* DIV 32
- the offset of the required GICD_ISENABLER is (0x100 + (4*n))
- the bit number of the required Set-enable bit in this register is *m* MOD 32.

3. set interrupt target (to cpu 0)
   - Make the interrupt target CPU interface number 0.

For interrupt ID *m*, when DIV and MOD are the integer division and modulo operations:

- the corresponding GICD_ITARGETSRn number, *n*, is given by *n* = *m* DIV 4
- the offset of the required GICD_ITARGETSR is (0x800 + (4*n))
- the byte offset of the required Priority field in this register is *m* MOD 4, where:
  - byte offset 0 refers to register bits [7:0]
  - byte offset 1 refers to register bits [15:8]
  - byte offset 2 refers to register bits [23:16]
  - byte offset 3 refers to register bits [31:24].

4. set interrupt triggering type (edge-triggering)
   - Set all the bits in the corresponding area to 1.

For interrupt ID *m*, when DIV and MOD are the integer division and modulo operations:

- the corresponding GICD_ICFGR number, *n*, is given by *n* = *m* DIV 16
- the offset of the required GICD_ICFGRn is (0xC00 + (4*n))
- the required Priority field in this register, *F*, is given by *F* = *m* MOD 16, where field 0 refers to register bits [1:0], field 1 refers to bits [3:2], up to field 15 that refers to bits [31:30], see Figure 4-15 on page 4-109.

Implement interrupt entry routine

HAL_arch_startup.S

```
vh_irq:
   sub     lr, lr, #4
   str     sp, vk_save_irq_mode_stack_ptr
   stmfd   sp, {r14}^
   sub     sp, sp, #4
   stmfd   sp, {r13}^
   sub     sp, sp, #4
   stmfd   sp!, {r0-r12}
   mrs     r0, spsr_all
   stmfd   sp!, {r0, lr}

   str     sp, vk_save_irq_current_tcb_bottom

   bl    vk_irq_handler

vh_leaving_irq:
   ldmfd   sp!, {r0, lr}
   msr     spsr_cxsf, r0
   ldmfd   sp!, {r0-r12}
   ldmfd   sp, {r13}^
```

```
    add     sp, sp, #4
    ldmfd   sp, {r14}^
    add     sp, sp, #4

    movs    pc, lr
```