Reference Information of CSC1004 – Computer Org & Arch Assignment 2

Thank you so much for providing the feedback on some technical issues of the assignment 2, with the most common issue as Segmentation Fault caused by the memory access.

For your info, we are not able to access the physical memory space of the ARM processor, we have to map the physical memory into a virtual memory mapping of the operating systems first, then we can access the virtual memory mapping, which can indirectly access the physical memory space.

Please refer to the article of this website on how to configure the GPIO (general purpose input output) peripherals. I revised the sample codes based on this website (https://www.pieter-jan.com/node/15).

You can also reference to this website to get some example codes on (https://www.raspberrypi.org/forums/viewtopic.php?t=186295).

For example the ARM timer's physical memory address with offset 0xB400 + Base 0x3F000000, and interrupt controller's physical memory address with offset 0xB200 + Base 0x3F000000. We can use the functions of map_peripheral ( ) and unmap_peripheral ( ) to achieve it. Please see the examples below for your modifications.

This sample codes have some bugs: we can read and write the registers to configure the Timer and Interrupt. But the interrupt will cause the system hung. And also the interrupt handler function is not invoked. Please help solve the bugs.

Please copy and paste the .c file and RPi_timer3.h file into your PC. And then compile them at Raspberry Pi using gcc. Finally please use the command below to execute the code:

sudo ./your_compiled_file

The reason we need use sudo to execute this program is that the memory mapping function need the super-user rights to access.

```
#include "RPi_timer3.h"
#include <stdio.h>

//struct bcm2835_peripheral gpio = {GPIO_BASE};
//struct bcm2835_peripheral bsc0 = {BSC0_BASE};
struct bcm2835_peripheral timer0 = {TIMER_BASE};

// Exposes the physical address defined in the passed structure using mmap on /de
v/mem
```

```
int map_peripheral(struct bcm2835_peripheral *p)
{
   // Open /dev/mem
   if ((p->mem_fd = open("/dev/mem", O_RDWR|O_SYNC) ) < 0) {
      printf("Failed to open /dev/mem, try checking permissions.\n");
      return -1;
   }

   p->map = mmap(
      NULL,
      BLOCK_SIZE,
      PROT_READ|PROT_WRITE,
      MAP_SHARED,
      p->mem_fd,  // File descriptor to physical memory virtual file '/dev/mem'
      p->addr_p      // Address in physical map that we want this memory block to
 expose
   );

   if (p->map == MAP_FAILED) {
        perror("mmap");
        return -1;
   }

   p->addr = (volatile unsigned int *)p->map;

   return 0;
}

void unmap_peripheral(struct bcm2835_peripheral *p) {

    munmap(p->map, BLOCK_SIZE);
    close(p->mem_fd);
}



/*-Enable_Interrupts-----------------------------------------------------
 Self explanatory enables ARM core processor interrupts
 ------------------------------------------------------------------------*/
void Enable_Interrupts (void) {
    asm(    "mrs     r0, cpsr   \n\t"
        "bic     r0, r0, #0x80  \n\t"
        "msr     cpsr_c, r0"
//      "msr     cpsr_c, r0 \n\t"   // added on 7 Oct 2019
//      "mov     pc, lr"    // added on 7 Oct 2019
```

```
    );
}

/* The interrupt service routine (ISR) we will call */
void __attribute__((interrupt("IRQ"))) c_irq_handler (void)
{
    TIMER_IRQCLEAR0 = 0;                    // Write any value to register to cle
ar irq ... PAGE 198
//    printf("\nTimer interrupt ISR is triggered");           // printf a message
  when the ARM timer interrupt is triggered.

}


int main()
{
    if(map_peripheral(&timer0) == -1)
    {
            printf("Failed to map the physical timer0 registers into the virtual
memory space.\n");
            return -1;
        }
    printf("\n step1\n");


        // read the values before operatings
        printf("\nIRQ_BASIC_PEND0 = : \n%X", IRQ_BASIC_PEND0);
        printf("\nENABLE_BASIC_IRQ0 = : \n%X", ENABLE_BASIC_IRQ0);
        printf("\nDISABLE_BASIC_IRQ0 = : \n%X", DISABLE_BASIC_IRQ0);
        printf("\nTIMER_LOAD0 = : \n%X", TIMER_LOAD0);
//        printf("\nTIMER_VALUE0 = : \n%X", TIMER_VALUE0);
        printf("\nTIMER_CONTROL0 = : \n%X", TIMER_CONTROL0);
        printf("\nTIMER_IRQCLEAR0 = : \n%X", TIMER_IRQCLEAR0);
        printf("\nTIMER_MASK_IRQ0 = : \n%X", TIMER_MASK_IRQ0);
        printf("\nTIMER_PRE_DIVD0 = : \n%X", TIMER_PRE_DIVD0);

        //write TIMER_LOAD0 register (offset 0x400), with a new value
        TIMER_LOAD0 = 0x500;
        printf("\nWrite value into Timer Load register, TIMER_LOAD0 = : \n%X",TIM
ER_LOAD0);

        //write TIMER_pre_divider1 register (offset 0x41C), with a new value
        TIMER_PRE_DIVD0 = 0x3F0;
        printf("\nWrite a pre_divider value into TIMER_PRE_DIVD0 register, TIMER_
PRE_DIVD0 = : \n%X", TIMER_PRE_DIVD0);
```

```
        //write DISABLE_BASIC_IRQ1 register (offset 0x224), to disable other IRQ
//        DISABLE_BASIC_IRQ0 |= ((3 << 1) & 0x00000003);   // set Bit 2-
3 of DISABLE_BASIC_IRQ register (offset 0x224) to 1, to disable IRQ.
//        printf("\nWrite Bit 0 of DISABLE BASIC IRQ register (disable IRQ), DISA
BLE_BASIC_IRQ0 = : \n%X",DISABLE_BASIC_IRQ0);

        printf("\nENABLE_BASIC_IRQ0 = : \n%X",ENABLE_BASIC_IRQ0);

////////////////////////////////////

        //write TIMER_CONTROL1 register (offset 0x408)
        TIMER_CONTROL0 |= ((2 << 2) & 0x0000000C);   // set Bit 2-
3 of TIMER_CONTROL1 register (offset 0x408) to 10, pre-scale is clock / 256.
        printf("\nWrite Bit 2-3 of Timer Control Register (pre-
scale clock/256), TIMER_CONTROL0 = : \n%X",TIMER_CONTROL0);

        //write TIMER_CONTROL1 register (offset 0x408)
        TIMER_CONTROL0 |= ((1 << 1) & 0x00000002);   // set Bit 1 of TIMER_CONTRO
L1 register (offset 0x408) to 1, to choose 23-bit counter.
        printf("\nWrite Bit 1 of Timer Control Register (choose 23-
bit counter), TIMER_CONTROL0 = : \n%X",TIMER_CONTROL0);

        //write TIMER_CONTROL1 register (offset 0x408)
        TIMER_CONTROL0 |= ((1 << 7) & 0x00000080);   // set Bit 7 of TIMER_CONTRO
L1 register (offset 0x408) to 1, to enable timer.
        printf("\nWrite Bit 7 of Timer Control Register (enable timer), TIMER_CON
TROL0 = : \n%X",TIMER_CONTROL0);

        //write TIMER_CONTROL1 register (offset 0x408)
        TIMER_CONTROL0 |= ((1 << 5) & 0x00000020);   // set Bit 5 of TIMER_CONTRO
L1 register (offset 0x408) to 1, to enable timer interrupt.
        printf("\nWrite Bit 5 of Timer Control Register (enable timer interrupt),
 TIMER_CONTROL0 = : \n%X",TIMER_CONTROL0);
////////////

        printf("\nbefore enable IRQ, IRQ_BASIC_PEND0 = : \n%X",IRQ_BASIC_PEND0);

        printf("\nIRQ_BASIC_PEND0 = : \n%X",IRQ_BASIC_PEND0);

        //// Enable interrupts!
        Enable_Interrupts();

        //write ENABLE_BASIC_IRQ1 register (offset 0x218), to enable IRQ
```

```
        ENABLE_BASIC_IRQ0 |= ((1 << 0) & 0x00000001);   // set Bit 0 of ENABLE_BA
SIC_IRQ register (offset 0x218) to 1, to enable IRQ.
        printf("\nWrite Bit 0 of ENABLE BASIC IRQ register (enable IRQ), ENABLE_B
ASIC_IRQ0 = : \n%X",ENABLE_BASIC_IRQ0);

//        printf("\nafter clear Timer IRQ register, IRQ_BASIC_PEND0 = : \n%X",IRQ
_BASIC_PEND0);

    printf("\nstep 3\n");

//        while (1) // don't change the while loop code.
//        {

//              printf("\n step2\n");

//        }

////////////////////////////////////
    return 0;
}
```

The RPi_timer3.h file is shown below:

```
#ifndef _INC_PJ_GPIO_H
#define _INC_PJ_GPIO_H

#include <stdio.h>

#include <string.h>
#include <stdlib.h>
#include <dirent.h>
#include <fcntl.h>
#include <assert.h>

#include <sched.h>      // To set the priority on linux

#include <sys/mman.h>
#include <sys/types.h>
#include <sys/stat.h>

#include <unistd.h>
```

```
// Define which Raspberry Pi board are you using. Take care to have defined only
one at time.
//#define RPI
#define RPI3

#ifdef RPI
#define BCM2708_PERI_BASE       0x20000000
#define GPIO_BASE       (BCM2708_PERI_BASE + 0x200000)  // GPIO controller
#define BSC0_BASE       (BCM2708_PERI_BASE + 0x205000)  // I2C controller
#define TIMER_BASE      (BCM2708_PERI_BASE + 0xB000)    // Timer controller base
address = 0xB400; interrupt base address = 0xB200;
#endif

#ifdef RPI3
#define BCM2708_PERI_BASE       0x3F000000
#define GPIO_BASE       (BCM2708_PERI_BASE + 0x200000)  // GPIO controller.
#define BSC0_BASE       (BCM2708_PERI_BASE + 0x804000)  // I2C controller
#define TIMER_BASE      (BCM2708_PERI_BASE + 0xB000)    // Timer controller base
address = 0xB400; interrupt base address = 0xB200;
#endif

#define PAGE_SIZE       (4*1024)
#define BLOCK_SIZE      (4*1024)

// IO Acces
struct bcm2835_peripheral {
    unsigned long addr_p;
    int mem_fd;
    void *map;
    volatile unsigned int *addr;
};

extern struct bcm2835_peripheral timer0;    // for timer + interrupt

//////////////////////////////
// timer and interrupt macros

// offset address on Datasheet of IRQ_BASIC_PENDING register is 0xB200. The mappe
d memory address starts from 0xB000.
// hence the offset in the memory mapping is (0xB200 - 0xB000) / 4 = 0x80; (4 byt
es in datasheet = one 32-bit in mapping)
#define IRQ_BASIC_PEND0     *(timer0.addr + 0x80) // Bit 0 of IRQ Basic Pending r
egister (offset 0xB200) is for ARM Timer IRQ pending status (read only)
```

```
// offset address on Datasheet of Base Interrupt Enable register is 0xB218. The m
apped memory address starts from 0xB000.
// hence the offset in the memory mapping is (0xB218 - 0xB000) / 4 = 0x86; (4 byt
es in datasheet = one 32-bit in mapping)
#define ENABLE_BASIC_IRQ0   *(timer0.addr + 0x86) // Bit 0 of Base Interrupt Enab
le register (offset 0xB218) is to enable ARM Timer IRQ (write/read)

// offset address on Datasheet of Base Interrupt Disable register is 0xB224. The
mapped memory address starts from 0xB000.
// hence the offset in the memory mapping is (0xB224 - 0xB000) / 4 = 0x86; (4 byt
es in datasheet = one 32-bit in mapping)
#define DISABLE_BASIC_IRQ0  *(timer0.addr + 0x89) // Base Interrupt Enable regist
er (offset 0xB224); disable other IRQ (write/read), except for timer.


// offset address on Datasheet of TIMER_LOAD register is 0xB400. The mapped memor
y address starts from 0xB000.
// hence the offset in the memory mapping is (0xB400 - 0xB000) / 4 = 0x100; (4 by
tes in datasheet = one 32-bit in mapping)
#define TIMER_LOAD0     *(timer0.addr + 0x100) // Bit 0-
31 of TIMER_LOAD register (offset 0xB400) is to sets the time to count down (writ
e/read)

// offset address on Datasheet of TIMER_CONTROL register is 0xB408. The mapped me
mory address starts from 0xB000.
// hence the offset in the memory mapping is (0xB408 - 0xB000) / 4 = 0x102; (4 by
tes in datasheet = one 32-bit in mapping)
#define TIMER_CONTROL0  *(timer0.addr + 0x102) // TIMER_CONTROL register (offset
0xB408) is to configure the Timer behaviors (write/read)
// Bit 1: 0 for 16 bits counter, 1 for 23 bits counter
// Bit 2-3: 00 for clock/1; 01 for clock/16; 10 for clock/256; 11 for clock/1;
// Bit 5: 0 disable timer interrupt, 1 enable timer interrupt
// Bit 7: 0 timer disable; 1 timer enable;

// offset address on Datasheet of TIMER_IRQCLEAR register is 0xB40C. The mapped m
emory address starts from 0xB000.
// hence the offset in the memory mapping is (0xB40C - 0xB000) / 4 = 0x103; (4 by
tes in datasheet = one 32-bit in mapping)
#define TIMER_IRQCLEAR0     *(timer0.addr + 0x103) // Write any value to TIMER_IR
QCLEAR register (offset 0xB408) to clear IRQ ... PAGE 198 (write)

// offset address on Datasheet of TIMER_MASK_IRQ register is 0xB414. The mapped m
emory address starts from 0xB000.
// hence the offset in the memory mapping is (0xB414 - 0xB000) / 4 = 0x105; (4 by
tes in datasheet = one 32-bit in mapping)
```

```
#define TIMER_MASK_IRQ0     *(timer0.addr + 0x105) // Read TIMER_MASK_IRQ registe
r (offset 0xB414) (read only), Bit 0: 0 interrupt line not asserted. 1 asserted.

// offset address on Datasheet of TIMER_PRE_DIVIDER register is 0xB41C. The mappe
d memory address starts from 0xB000.
// hence the offset in the memory mapping is (0xB41C - 0xB000) / 4 = 0x107; (4 by
tes in datasheet = one 32-bit in mapping)
#define TIMER_PRE_DIVD0     *(timer0.addr + 0x107) // TIMER_PRE_DIVIDER register
(offset 0xB41C) (write and read), Bit 0-9.

////////////////////////////

#endif
```