README

其他班考题

- 周一晚信卓：按下`U`键，读入开关低四位状态，以16进制显示在数码管最右位上；每按下一次`L`键，显示位置左移一格，左移可以循环
- 周二晚电信：按下`U`键，读入开关低八位状态，以16进制显示在数码管最右两位上；按下`L`键，数字以1Hz频率增长，至255后归零再循环
- 周三晚电信：按下`U`键，读入开关低四位状态，以16进制显示在数码管最右位上；按下`L`键，数字以1Hz频率显示左移，左移可以循环；再次按下`U`键，数字重新回到最低位，保持不动，以等待下一次`L`键的按下

分析

可以看到，三个晚上的考题都并不算难，且都涉及到的操作有：

- 按下某键，读入开关状态（低四位或者低八位）
- 开关状态以16进制显示在数码管上
- 按下另一键，对**数字本身**或是**数字显示位置**做出改变，改变无非就是1Hz频率增加或是左移

大胆预测，我们的考题不会和他们的差太多，涉及到的有：**按*button*读取*switch*状态**、***16*进制显示**、***1Hz*频率操作**、**数码管左移操作。**

接下来将以周三晚电信的考题为例，结合我自己写的代码进行分析

代码分析

介于周三晚电信班的题目我是在八点四十左右拿到的，花费了不到5分钟就写完了需求，代码可能不是很美观：

```c
/*
 * elec_Wed.c
 *
 *  Created on: 2023年5月17日
 *      Author: 畅想未来
 */

#include "xil_io.h"
#include "stdio.h"
#include "xgpio_l.h"
#include "xintc_l.h"
#include "xtmrctr_l.h"
#define RESET_VALUE 100000

void display_seg();
void initial_seg();
void btn_handler();
void My_ISR() __attribute__((interrupt_handler));

int i = 0;
int j = 0;
int hasClickedLeftBtn = 0;
unsigned short currentBtn, lastBtn, realBtn;
unsigned short num;
unsigned short pos = 0xfffe;
unsigned short undefinedCode = 0xff;
//segCode为0-f的七段数码管译码显示码，以8421BCD码为下标，如segCode[3]=0xb0，
segCode[e]=0x86;
unsigned short segCode[] =
{0xc0,0xf9,0xa4,0xb0,0x99,0x92,0x82,0xf8,0x80,0x90,0x88,0x83,0xa7,0xa1,0x86,0x8e};


int main() {
    print("main func on run");
    // 和程序控制一样，设置LED为输出，switch为输入
    Xil_Out32(XPAR_AXI_GPIO_0_BASEADDR + XGPIO_TRI2_OFFSET, 0x0);
    Xil_Out32(XPAR_AXI_GPIO_0_BASEADDR + XGPIO_TRI_OFFSET, 0xffff);
// GPIO_0使能中断：
// XGPIO_IER_OFFSET：GPIO外设中用于中断使能的寄存器地址偏移量；IER：interrupt enable
register
// XGPIO_IR_CH1_MASK：这是一个掩码，用于指定GPIO外设的第1个IO口所对应的中断信号。在这里，
将这个掩码写入到中断使能寄存器中，表示使能GPIO的第1个IO口产生中断。
// XGPIO_GIE_OFFSET：这是GPIO外设中用于总中断使能的寄存器地址偏移量，通过向这个寄存器写入
某个值，可以使得GPIO外设的所有中断信号都被使能
// XGPIO_GIE_GINTR_ENABLE_MASK：这是一个掩码，用于开启所有GPIO中断信号的总中断使能。将这个
掩码写入到总中断使能寄存器中，表示开启GPIO外设所有中断信号的总中断使能
    Xil_Out32(XPAR_AXI_GPIO_0_BASEADDR + XGPIO_IER_OFFSET, XGPIO_IR_CH1_MASK);
    Xil_Out32(XPAR_AXI_GPIO_0_BASEADDR + XGPIO_GIE_OFFSET,
XGPIO_GIE_GINTR_ENABLE_MASK);
// INTC：interrupt controller
```

```
// XPAR_AXI_GPIO_0_IP2INTC_IRPT_MASK：GPIO中断的掩码。它用于控制哪个GPIO中断源可以激活中断。GPIO可能有多个中断源，这个掩码用于标识哪些中断源是可以激活中断的。
// XIN_MER_OFFSET:Interrupt Controller的Master Enable寄存器的偏移地址。这个寄存器用于启用或禁用整个中断控制器的中断信号，以及硬件中断和软件中断的全局控制。
// XIN_INT_MASTER_ENABLE_MASK 和 XIN_INT_HARDWARE_ENABLE_MASK：两个掩码，MASTER使能中断控制器的主总线，HARDWARE则使能硬件中断信号。
    Xil_Out32(XPAR_AXI_INTC_0_BASEADDR + XIN_IER_OFFSET, XPAR_AXI_GPIO_0_IP2INTC_IRPT_MASK);  // interrupt channel
    Xil_Out32(XPAR_AXI_INTC_0_BASEADDR + XIN_MER_OFFSET, XIN_INT_MASTER_ENABLE_MASK | XIN_INT_HARDWARE_ENABLE_MASK);

    Xil_Out32(XPAR_AXI_GPIO_1_BASEADDR + XGPIO_TRI_OFFSET, 0x0);  // set seg show signal as output
    Xil_Out32(XPAR_AXI_GPIO_1_BASEADDR + XGPIO_TRI2_OFFSET, 0x0);  // set seg pos signal as output
    Xil_Out32(XPAR_AXI_GPIO_2_BASEADDR + XGPIO_TRI_OFFSET, 0x1f);  // btn input
// 同上的GPIO_0部分
    Xil_Out32(XPAR_AXI_GPIO_2_BASEADDR + XGPIO_IER_OFFSET, XGPIO_IR_CH1_MASK);  // allow channel 1 interrupt
    Xil_Out32(XPAR_AXI_GPIO_2_BASEADDR + XGPIO_GIE_OFFSET, XGPIO_GIE_GINTR_ENABLE_MASK);  // allow GPIO interrupt output
// TCSR: Timer Control/Status Register
    // xil_printf("before:0x%x\n",Xil_In32(XPAR_AXI_TIMER_0_BASEADDR + XTC_TCSR_OFFSET));
    Xil_Out32(XPAR_AXI_TIMER_0_BASEADDR + XTC_TCSR_OFFSET,
              Xil_In32(XPAR_AXI_TIMER_0_BASEADDR + XTC_TCSR_OFFSET) & ~XTC_CSR_ENABLE_TMR_MASK);  // write TCSR, stop counter
    // xil_printf("after:0x%x\n",Xil_In32(XPAR_AXI_TIMER_0_BASEADDR + XTC_TCSR_OFFSET));

// 写TLR，预置初始值为RESET_VALUE（100000）
// TLR:Timer Load Register，用于设置定时器的计数器初始值。
    Xil_Out32(XPAR_AXI_TIMER_0_BASEADDR + XTC_TLR_OFFSET, RESET_VALUE);  // write TLR, preset counter value

// XTC_CSR_LOAD_MASK：用于启用定时器的加载操作的掩码值
// 启动定时器的加载操作
    Xil_Out32(XPAR_AXI_TIMER_0_BASEADDR + XTC_TCSR_OFFSET,
              Xil_In32(XPAR_AXI_TIMER_0_BASEADDR + XTC_TCSR_OFFSET) | XTC_CSR_LOAD_MASK);  // get counter initial value

// 定时器配置为启用向下计数模式，自动重装载计数器值，同时允许定时器中断
    Xil_Out32(XPAR_AXI_TIMER_0_BASEADDR + XTC_TCSR_OFFSET,
              (Xil_In32(XPAR_AXI_TIMER_0_BASEADDR + XTC_TCSR_OFFSET) & ~XTC_CSR_LOAD_MASK)
              | XTC_CSR_ENABLE_TMR_MASK | XTC_CSR_AUTO_RELOAD_MASK | XTC_CSR_ENABLE_INT_MASK | XTC_CSR_DOWN_COUNT_MASK);

    Xil_Out32(XPAR_AXI_INTC_0_BASEADDR + XIN_IER_OFFSET, XPAR_AXI_GPIO_2_IP2INTC_IRPT_MASK | XPAR_AXI_TIMER_0_INTERRUPT_MASK);  //Enable the interrupt controller
```

```
    Xil_Out32(XPAR_AXI_INTC_0_BASEADDR + XIN_MER_OFFSET, XIN_INT_MASTER_ENABLE_MASK |
XIN_INT_HARDWARE_ENABLE_MASK);
    microblaze_enable_interrupts();  // allow microbalze enable interrupt
    return 0;
}

void My_ISR() {
    int status;
    status = Xil_In32(XPAR_AXI_INTC_0_BASEADDR + XIN_ISR_OFFSET);
    if ((status & XPAR_AXI_GPIO_2_IP2INTC_IRPT_MASK) ==
XPAR_AXI_GPIO_2_IP2INTC_IRPT_MASK) {
        print("btn onclick!\n");
        btn_handler();
    }
    Xil_Out32(XPAR_AXI_INTC_0_BASEADDR + XIN_IAR_OFFSET, status);
    if ((status & XPAR_AXI_TIMER_0_INTERRUPT_MASK) == XPAR_AXI_TIMER_0_INTERRUPT_MASK)
{
        display_seg();
        if (hasClickedLeftBtn == 1) {
            if (j < 195) {
                j++;
                xil_printf("j=%d\n",j);
            }
            else {
                j = 0;
                pos = pos << 1;
                pos += 1;
                i++;
                if (i == 8) {
                    i = 0;
                    pos = 0xfffe;
                }
            }
        }
    }
    Xil_Out32(XPAR_AXI_INTC_0_BASEADDR + XIN_IAR_OFFSET, status);
}

void btn_handler() {
    // 以下代码为防抖动
    currentBtn = Xil_In8(XPAR_AXI_GPIO_2_BASEADDR + XGPIO_DATA_OFFSET) & 0x1f;
        if (currentBtn) {  // btn onClick
            while (currentBtn) {  // btn not spring up
                currentBtn = (Xil_In8(XPAR_AXI_GPIO_2_BASEADDR + XGPIO_DATA_OFFSET) &
0x1f);
                realBtn = (currentBtn ^ lastBtn) & lastBtn;  // get unpressed btn
                lastBtn = currentBtn;
                if (realBtn) {
                    break;
                }
```

```
            }
        }
    switch (realBtn) {
    case 0x02:
        hasClickedLeftBtn = 0;
        pos = 0xfffe;
        j = 0;
        i = 0;
        num = Xil_In16(XPAR_AXI_GPIO_0_BASEADDR+XGPIO_DATA_OFFSET);
        num &= 0x000f;
        xil_printf("led low 4 bits: %x\n",num);
        display_seg();
        break;
    case 0x04:
        hasClickedLeftBtn = 1;
        break;
    default:
        break;
    }
    Xil_Out32(XPAR_AXI_GPIO_2_BASEADDR + XGPIO_ISR_OFFSET,
Xil_In32(XPAR_AXI_GPIO_2_BASEADDR + XGPIO_ISR_OFFSET));
}

void display_seg() {
    Xil_Out8(XPAR_AXI_GPIO_1_BASEADDR + XGPIO_DATA2_OFFSET,segCode[num]);
    Xil_Out8(XPAR_AXI_GPIO_1_BASEADDR + XGPIO_DATA_OFFSET ,pos);
    Xil_Out32(XPAR_AXI_TIMER_0_BASEADDR + XTC_TCSR_OFFSET,
Xil_In32(XPAR_AXI_TIMER_0_BASEADDR+XTC_TCSR_OFFSET));
}
```

不考虑库的引入和宏的定义，将上述代码分成几个部分：

1. `main()`函数
2. `My_ISR()`函数
3. 自定义变量，函数及其实现

其实就四个部分，而真正需要我们编写的，只有1和2，可能有时也会需要改动一下4。

- 首先`main()`函数里，是对中断服务的各种初始化操作，我们**不需要**在`main()`函数中做任何的改动，所以之前代码中能正常运行的`main()`函数，大胆的C过来吧，不需要改动
- 其次`My_ISR()`函数，My_ISR的全称是My Interrupt Service Routine，在这里进行各种中断事务的处理，比如按下某个button，比如你需要哪个数码管扫略显示，比如可能还需要更新一下某个LED的亮灭状态等，这些也可以直接从过去的代码中C过来，可能会有一些些改动
- 然后是自定义全局变量和自定义的函数&实现，这一部分才是我们要编写的核心代码，在这一部分的代码中，我们需要对不同的按键`Onclick`操作进行处理，需要对数码管显示进行处理，以下将继续以周三电信班的考题为例，结合我的核心代码进行讲解

核心代码部分

按下`U`键，读入开关状态

这一部分的核心当然是`btn_handler()`函数啦：

```
int status;
status = Xil_In32(XPAR_AXI_INTC_0_BASEADDR + XIN_ISR_OFFSET);
if ((status & XPAR_AXI_GPIO_2_IP2INTC_IRPT_MASK) == XPAR_AXI_GPIO_2_IP2INTC_IRPT_MASK)
{
    print("btn onclick!\n");
    btn_handler();
}
```

以上一部分代码来自`My_ISR()`，可以直接从以前运行正常对的代码中C过来

```
void btn_handler() {
    // 以下代码为防抖动
    currentBtn = Xil_In8(XPAR_AXI_GPIO_2_BASEADDR + XGPIO_DATA_OFFSET) & 0x1f;
        if (currentBtn) {  // btn onClick
            while (currentBtn) {  // btn not spring up
                currentBtn = (Xil_In8(XPAR_AXI_GPIO_2_BASEADDR + XGPIO_DATA_OFFSET) &
0x1f);
                realBtn = (currentBtn ^ lastBtn) & lastBtn;  // get unpressed btn
                lastBtn = currentBtn;
                if (realBtn) {
                    break;
                }
            }
        }
    switch (realBtn) {
    case 0x02:
        hasClickedLeftBtn = 0;
        pos = 0xfffe;
        j = 0;
        i = 0;
        num = Xil_In16(XPAR_AXI_GPIO_0_BASEADDR+XGPIO_DATA_OFFSET);
        num &= 0x000f;
        xil_printf("led low 4 bits: %x\n",num);
        display_seg();
        break;
    case 0x04:
        hasClickedLeftBtn = 1;
        break;
    default:
        break;
    }
    Xil_Out32(XPAR_AXI_GPIO_2_BASEADDR +
XGPIO_ISR_OFFSET,Xil_In32(XPAR_AXI_GPIO_2_BASEADDR + XGPIO_ISR_OFFSET));
}
```

以上代码关键点：

1. 防抖动
2. 进入分支判断、处理

数码管译码显示

```
//segCode为0-f的七段数码管译码显示码，以8421BCD码为下标，如segCode[3]=0xb0，
segCode[e]=0x86;

unsigned short segCode[] =
{0xc0,0xf9,0xa4,0xb0,0x99,0x92,0x82,0xf8,0x80,0x90,0x88,0x83,0xa7,0xa1,0x86,0x8e};
```

数码管译码显示的`0-F`的数值也是可以提前准备好的，直接C过去

```
if ((status & XPAR_AXI_TIMER_0_INTERRUPT_MASK) == XPAR_AXI_TIMER_0_INTERRUPT_MASK) {
        display_seg();
        if (hasClickedLeftBtn == 1) {
            if (j < 195) {
                j++;
                xil_printf("j=%d\n",j);
            }
            else {
                j = 0;
                pos = pos << 1;
                pos += 1;
                i++;
                if (i == 8) {
                    i = 0;
                    pos = 0xfffe;
                }
            }
        }
    }
    Xil_Out32(XPAR_AXI_INTC_0_BASEADDR + XIN_IAR_OFFSET, status);
```

以上这段代码在`My_ISR()`函数中，倘若没有“按下`L`键左移”的需求，那么`if`语句中将只有一个`display_seg()`函数

```
void display_seg() {
    Xil_Out8(XPAR_AXI_GPIO_1_BASEADDR + XGPIO_DATA2_OFFSET,segCode[num]);
    Xil_Out8(XPAR_AXI_GPIO_1_BASEADDR + XGPIO_DATA_OFFSET ,pos);
    Xil_Out32(XPAR_AXI_TIMER_0_BASEADDR + XTC_TCSR_OFFSET,
Xil_In32(XPAR_AXI_TIMER_0_BASEADDR+XTC_TCSR_OFFSET));

}
```

以上代码是七段数码管译码显示的核心部分，前两句分别将数码管的片选、位选信号输出到数码管硬件设备上，

第三句将重启AXI_TIMER寄存器，为下一次触发中断做准备

若是要在两位上显示数据，该如何操作?

周一电信班的考题中要求在数码管的右**两位**显示十六进制数字，这时我们就需要对`pos`进行额外的处理，让他每次译码显示（每次执行`display_seg()`）的时候对应的值都不一样，以在在不同的位置显示我们想让他显示的值

我的具体操作为，定义一个全局变量`i`用来标记此时的位置，每次执行`display_seg()`函数时，让`i`增长一次，同时`pos`也做出相应改变，这样再下一次执行`display_seg()`函数时，就是不一样的`pos`在发挥作用了，如果想在不同位置显示不同的值，也可以用`i`来选值

按下`L`键，位置以1Hz进行左移

这里我们利用`My_ISR()`函数循环执行的特性，再定义一个全局变量`j`,通过`j`的增长来控制以1Hz频率执行某些操作，比如在周三晚电信题目中就是将`pos`做一些改动

```
if (hasClickedLeftBtn == 1) {
    if (j < 195) {
        j++;
        xil_printf("j=%d\n",j);
    } else {
        j = 0;
        pos = pos << 1;
        pos += 1;
        i++;
        if (i == 8) {
            i = 0;
            pos = 0xfffe;
        }
    }
}
```

这里的`xil_printf()`函数我本来是想查看`j`的状态的，奈何怎么删掉这句之后就出问题了，所以没有删掉它。猜测是因为输出到terminal是耗时操作，删除之后执行太快了就出问题了

细节处理

这里还要求重新按下`U`键后再次读入数据，且保持在右端不动以等待下一次`L`键的按下，我们只需要吧`i`，`pos`，`j`等值归零，恢复到刚烧录进去的状态即可

总结

需要我们自己写的代码只有自定义的变量和函数，且模板可现套用

可用模板：

0~F译码显示

```
//segCode为0-f的七段数码管译码显示码，以8421BCD码为下标，如segCode[3]=0xb0，
segCode[e]=0x86;
unsigned short segCode[] =
{0xc0,0xf9,0xa4,0xb0,0x99,0x92,0x82,0xf8,0x80,0x90,0x88,0x83,0xa7,0xa1,0x86,0x8e};
```

按键防抖动

```
unsigned short currentBtn, lastBtn, realBtn;


currentBtn = Xil_In8(XPAR_AXI_GPIO_2_BASEADDR + XGPIO_DATA_OFFSET) & 0x1f;
    if (currentBtn) {  // btn onClick
        while (currentBtn) {  // btn not spring up
            currentBtn = (Xil_In8(XPAR_AXI_GPIO_2_BASEADDR + XGPIO_DATA_OFFSET) &
0x1f);
            realBtn = (currentBtn ^ lastBtn) & lastBtn;  // get unpressed btn
            lastBtn = currentBtn;
            if (realBtn) {
                break;
            }
        }
    }
switch(realBtn) {

}
```

1Hz频率

```
if (hasClickedLeftBtn == 1) {
    if (j < 195) {
        j++;
        xil_printf("j=%d\n",j);
    } else {
        j = 0;
        /* do something with 1Hz frequency*/
    }
}
```

剩下的代码，基本都是重复的啦，直接化身CV战士就可以了

考题预测

- 按下`U`键，读取开关低四位状态，以16进制显示在数码管上；

- 按下`L`键，数字以1Hz频率加一加到15后下一位归零，同时向左移，左移可循环；
- 再次按下`U`键，读取新的低四位开关状态，固定在最右侧不动，等待下一次`L`键按下