ARM 实验报告 (讨论题)

张浩宇 522031910129

1 实验一：时钟选择与 GPIO 实验

1. 在硬件不改变的情况下，在程序中将内部时钟从 16M 改为 8M，或将外部时钟从 25M 改为 30M，会有什么结果？

   由于时钟频率的改变，使系统频率变化，会导致延时函数的延时时间的改变，从而使闪烁频率改变。

2. 能否将启用 PLL 后的系统时钟频率调整到 PIOSC(16M) 或 MOSC(25M) 以下？

   可以。将 PLL 频率配置为 300MHz 或 400，可输出此 PPL 频率的任意分频作为系统频率，最低可配置为 6Mhz，因此可以将系统时钟调整到 PIOSC(16M) 或 MOSC(25M) 以下。

3. 将 PLL 后的时钟调整为最大值 120M，LED 闪烁会有什么变化？为什么？

   会使 LED 闪烁频率很快，在快闪时由于闪烁变化超过人眼的分辨能力，会表现为常亮。

4. 理解`GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_0, GPIO_PIN_0)`函数中每个函数参数的意义。第三个函数项为`GPIO_PIN_0`，如果改为 1 或改为 2，或其他值，分别有什么现象？

   第一个参数表示操作的 GPIO 端口基地址，这里表示操作端口 F；第二个参数表示操作的引脚，这里仅操作第一个引脚 PF0；第三个参数表示操作的引脚的电平值，这里仅将 PF0 设置为高电平，PF0 的 LED 点亮。

   `GPIO_PIN_0`的值得为 0x01。如果改为 1，PF0 引脚的状态仍为高电平，现象不变。如果改为 2，即 0x02，PF0 引脚处为低电平，LED 将熄灭。如果修改为其他值，仅当最低位即 PF0 对应位置为 1 时，PF0 的 LED 点亮，否则将熄灭

5. 结合硬件说明`GPIOPinConfigure`行的作用。如果此行注释，在 WATCH 窗口中观察 `key_value`值会有什么变化？

   该行的作用是 GPIO 引脚配置，将 PJ0 和 PJ1 输入配置为驱动强度 2mA、推挽上拉模式。如果此行注释，输入端口未配置为上拉。在 WATCH 窗口中观察，

注释前未按下按键key_value 值为 0x00000001，注释后未按下按键key_value值为 0x00000000。

6. 为什么在慢闪时，按住 USR_SW1 时，没有马上进入快闪模式？

在实验一的程序中，闪烁是通过阻塞延时来实现的，其在延时时，MCU 被完全阻塞，不执行其他任务。若在阻塞延时中按下开关，则必须等当前延时结束后才会处理响应按键按下，因此不能马上进入快闪模式。

2 实验二：I2C GPIO 扩展及 SYSTICK 中断实验

1. 在使用 PIOSC 及 MOSC 时，能否生成频率不等于晶振频率的系统时钟？如内部晶振为 16M，能否生成 10M 的系统时钟频率。

可以。在使用 PIOSC 及 MOSC 时，可以通过 PLL 来生成频率不等于晶振频率的系统时钟。例如，要生成 10M 的系统时钟频率，只需在初始化时调用下列配置函数：

```
SysCtlClockFreqSet((SYSCTL_OSC_INT|SYSCTL_USE_PLL|SYSCTL_CFG_VCO_480
    ),10000000);
```

2. 在使用 PLL 时，系统频率最小值及最大值分别为多少？

在使用 PLL 时，系统频率最小值为 6Mhz，最大值为 120Mhz。

3. 如果跑马灯要求为 2 位跑马，例：当显示为 1 时，跑马灯点亮 LED8，LED1，当显示为 2 时，跑马灯点亮 LED1，LED2，如此循环，如何实现？

定义一个uint8_t类型的变量bit_choose作为数码管的位选，初始值为 0x81，当 SysTick 计数达到上限，输出标志，要改变数码管状态时，按照如下方式改变bit_choose的值：

```
    if (value & 0x80) { // 最高位的1移到最低位
        value = (value << 1) | 0x01;
    } else {
        value <<= 1;
    }
```

这样每次位选两个数码管并且移动，实现 2 位跑马。

4. 在 3 基础上，数码管显示也改为 2 位显示。例如

第一步显示 1，2，跑马灯显示 LED1,2

第二步显示 2，3，跑马灯显示 LED2，3

……

第八步显示 8，1，跑马灯显示 LED8，1

第九步回到第一步

在 3 的基础上，要实现数码管不同位显示不同的数字，需要对数码管快速的扫描，快速切换位选和断码，以表现出显示不同的数字。定义一个uint8_t类型的变量bit_choose作为数码管的位选，指向每次点亮的两个数码管的较前一个，初始值 0x01，定义一个uint8_t类型的变量seg_display表示当前显示的数字段码索引，初始值为 0。

在 SysTick 中断服务函数中，设定 100ms 和 1ms 的两个计数标志，分别作为数码管扫描和数码管滚动显示的标志。

```
void SysTick_Handler(void)
{
    if (systick_100ms_couter != 0)
           systick_100ms_couter--;
    else
    {
           systick_100ms_couter     = SYSTICK_FREQUENCY/10;
           systick_100ms_status     = 1;
    }
    if (systick_1ms_couter   != 0)
        systick_1ms_couter--;
    else
    {
        systick_1ms_couter    = SYSTICK_FREQUENCY/1000;
        systick_1ms_status    = 1;
    }
}
```

每当systick_100ms_status为 1 时，切换位选和段码，以显示不同的数字，每当systick_1ms_status为 1 时，进行扫描显示。

```
void main(){
    ......
    while(1) {
        ......
```

```
        if (systick_100ms_status == 1) {
            systick_100ms_status = 0;
        bit_choose = (bit_choose << 1) | (bit_choose >> 7) // 位选移
            动
        seg_display = (seg_display + 1) % 8; // 数码管显示索引移动
        }

        if (systick_1ms_status == 1) {
            systick_1ms_status = 0;
            Display( bit_choose, seg_display);
        }
    }
}
```

Display(uint8_t bit_choose, uint8_t seg_display)函数实现一次显示扫描。

```
void Display(uint8_t bit_choose, uint8_t seg_display) {
    static uint8_t extra_bit=0; // 标志当前显示第一位还是第二位
    // 段选
    result = I2C0_WriteByte(TCA6424_I2CADDR,TCA6424_OUTPUT_PORT1,
        seg7[(seg_display+extra_bit)%8]);
    // 位选
    result = I2C0_WriteByte(TCA6424_I2CADDR,TCA6424_OUTPUT_PORT2,(
        bit_choose << extra_bit) | (bit_choose >> 7*extra_bit ));
    extra_bit!=extra_bit;
}
```

5. 用 USR_SW1 控制跑马灯的频率，

按第 1 下，间隔为 1S；

按第 2 下，间隔为 2S；

按第 3 下，间隔为 0.2S；

按第 4 下，回到上电初始状态，间隔 0.5S。

以 4 为模，循环往复。

在 SysTick 中断服务函数中，检测按键的按下，根据按键的按下次数，改变跑马灯计数上限标志的值。

```c
void SysTick_Handler(void)
{
    static uint8_t key_value, key_value_last=1,key_press_count=0;
    if (systick_couter != 0)
           systick_couter--;
    else
    {   // 达到上限SYSTICK_MAX时触发标志
           systick_couter    = SYSTICK_MAX;
           systick_status    = 1;
    }

    key_value = GPIOPinRead(GPIO_PORTJ_BASE, GPIO_PIN_0); // 读取按键
    if (key_value == 0 && key_value_last != 0) { // 按键按下时
        key_press_count++;
        if (key_press_count == 1) {
            SYSTICK_MAX=SYSTICK_FREQUENCY; // 1s
        } else if (key_press_count == 2) {
            SYSTICK_MAX=2*SYSTICK_FREQUENCY; // 2s
        } else if (key_press_count == 3) {
            SYSTICK_MAX=SYSTICK_FREQUENCY/5; // 0.2s
        } else if (key_press_count == 4) {
            SYSTICK_MAX=SYSTICK_FREQUENCY/2; // 0.5s
            key_press_count = 0; // 回到初始状态
        }
    }

    key_value_las=key_value;
}
```

6. 请编程在数码管上实现时钟功能，在数码管上最左端显示分钟 + 秒数，其中分钟及秒数均为 2 位数字。如 12:00，共 5 位。

   每隔一秒，自动加 1，当秒数到 60 时，自动分钟加 1，秒数回到 00，分钟及秒数显示范围 00 59。

   当按下 USR_SW1 时，秒数自动加 1；

   当按下 USR_SW2 时，分钟自动加 1；

当按下以上一个或两个按键不松开时，对应的显示跳变数每隔 200mS 自动加 1。即如下

按下 USR_SW1 1S，则显示跳变秒数加 5。

定义全局变量min和sec存储当前时间，Systick 的 1s 计时标志有效时，增加对应时间值，并通过快速扫描实现时间在数码管上显示。

```
void main(){
    ......
    while(1) {
        ......
        if (systick_1s_status == 1) { // 时间增加和进位
            systick_1s_status = 0;
            sec++;
            if (sec == 60) {
                sec = 0;
                min++;
                if (min == 60) {
                    min = 0;
                }
            }
        if (systick_1ms_status == 1) { // 扫描显示时钟
            systick_1ms_status = 0;
            Display();
        }

    }
}
```

其中Display()函数实现数码管的快速扫描显示。

```
void Display() {
    static uint8_t i=0; // 指示本次扫描的位置
    // 构造当前显示的段码
        uint8_t seg_array[8];
        seg_array[0]=(seg7[(min)/10];
        seg_array[1]=(seg7[(min)%10];
        seg_array[2]=0x40;
```

```
        seg_array[3]=(seg7[(sec)/10];
        seg_array[4]=(seg7[(sec)%10];

    // 位选和段选显示本次扫描位的数字
        result = I2C0_WriteByte(TCA6424_I2CADDR,TCA6424_OUTPUT_PORT1,
            seg_array[i]);

        result = I2C0_WriteByte(TCA6424_I2CADDR,TCA6424_OUTPUT_PORT2
            ,(uint8_t)(0x01 << i));
        i++;
        if (i==5) i=0; // 仅用前5位显示
}
```

在 SysTick 中断服务函数中，根据按键按下情况，改变当前计时变量值和 1s 计数上限

```
void SysTick_Handler(void)
{
    static uint8_t key_value1, key_value1_last=1,key_value2,
        key_value2_last=1;
    // 读取按键状态
    key_value1 = GPIOPinRead(GPIO_PORTJ_BASE, GPIO_PIN_0);
    key_value2 = GPIOPinRead(GPIO_PORTJ_BASE, GPIO_PIN_1);
    if (key_value == 0 && key_value_last != 0) { // 按键1按下时
        SYSTICK_1s_MAX=SYSTICK_FREQUENCY; // 1s
        sec++;
        if (sec == 60) {
            sec = 0;
            min++;
            if (min == 60) {
                min = 0;
            }
        }
    }
    else if(key_value == 0 && key_value_last == 0) { // 按键1长按时
        SYSTICK_1s_MAX=SYSTICK_FREQUENCY/5; // 0.2s // 改变计数上限
    }
```

```
    else {
        SYSTICK_1s_MAX=SYSTICK_FREQUENCY; // 1s
    }
    if (key_value2 == 0 && key_value2_last != 0) { // 按键2按下时
        min++;
        if (min == 60) {
            min = 0;
        }
    }

    if (systick_1s_couter    != 0) // 计时增加计数
            systick_1s_couter--;
    else
    {
        // 达到上限SYSTICK_MAX时触发标志
            systick_1s_couter = SYSTICK_1s_MAX;
            systick_1s_status  = 1;
    }
    if (systick_1ms_status == 1) { // 扫描显示计数
        systick_1ms_status = 0;
        Display( bit_choose, seg_display);
    }



    key_value_las=key_value;
}
```

3 实验三：UART 串行通讯口及中断优先级实验

1. 实验 32，`if (UARTCharsAvail(UART0_BASE))`此行程序的作用。如果没有此行，会导致什么问题？

   此行程序的作用是判断 UART0 接收 FIFO 中是否有数据，如果有数据时接收并按原样发送，没有数据则跳过。

   如果没有此行，每次主程序循环串口会一直进行阻塞接收和发送，程序会阻塞在此，不能执行跑马灯等其他任务。

2. 实验 33，void UART0_Handler(void)为什么没有在主函数声明？

   void UART0_Handler(void)是一个中断服务函数,其函数名是固定的,在startup_TM4C129.件中已默认注册，在主函数中重写即可覆盖该函数，无需另声明。

3. 为什么 33 的中断中需要读取中断标志并清除，而 SYSTICK 不需要？

   因为 SysTick 中断会被硬件自动清除。

4. 请根据上位机发出的命令字符串，如“MAY+01”，其中：

   MAY 为月份，（JAN,FEB,…DEC）均为三位。

   “+”表示加运算符,“-”表示减运算符，均为 1 位。

   01 表示增加或减少量，均为 2 位。范围 00-11。

   以上均为 ASCII 码，MAY+01 应该回之以 JUNE，MAY-06 应该回之以 NOV。

   在 UART0 中断服务函数中，对接收的字符串判断，并根据不同情况进行处理，发送相应的字符串即可。

   定义一个表示月份的全局数组以便操作，并定义两个全局变量分别表示接收到的月份和要增加的月份。

```
char *month[] = {"JAN", "FEB", "MAR", "APR", "MAY", "JUN", "JUL", "
    AUG", "SEP", "OCT", "NOV", "DEC"};
int8\_t mouth=0,add=0; // 接收到的月份和要增加的月份
```

   在 UART0 中断服务函数中，以非阻塞方式接收字符，并进行判断，以确定对应的月份和月份增加数。

```
void UART0_Handler() {
    .....
    UARTStringGetNonBlocking(msg); // 接收字符串
    for (int i=0;i<12;i++) {
        if (strncmp(msg, month[i], 3) == 0) {
            mouth = i;
            break;
        }
        if (i==11) {
            UARTStringPutNonBlocking("Invalid month!\n");
            return;
        }
    }
```

```
14
15      // 确定增加的月份
16      if (msg[3] == '+') {
17          add = (msg[4]-'0')*10 + (msg[5]-'0');
18          mouth += add;
19          mouth %= 12; // 防止超界
20      } else if (msg[3] == '-') {
21          add = (msg[4]-'0')*10 + (msg[5]-'0');
22          mouth -= add;
23          mouth = (mouth+12)%12; // 防止超界
24      } else {
25          UARTStringPutNonBlocking("Invalid command!\n");
26          return;
27      }
28
29      UARTStringPutNonBlocking(month[mouth]); // 输出处理好的月份
30
31
32  }
```

5. 请根据上位机的命令，如“14:12+05:06”，字符串中：

   14:12 为分钟与秒，共 5 位，包括一个“:”。

   “+”表示加运算符，“-”表示减运算符，均为 1 位。

   05:06 为分钟与秒的变化量，共 5 位。包括一个“:”，范围 00:00 23:59。

   以上均为 ASCII 码，14:12+05:06 回之以 19:18。

   在 UART0 中断服务函数中，对接收的字符串判断，并根据不同情况进行处理。根据输入字符串得到分钟和秒，然后根据加减号和变化量进行处理，最后输出处理好的分钟和秒。

```
1   void UART0_Handler() {
2       .....
3       UARTStringGetNonBlocking(msg); // 接收字符串
4       uint8_t min = (msg[0]-'0')*10 + (msg[1]-'0');
5       uint8_t sec = (msg[3]-'0')*10 + (msg[4]-'0');
6       uint8_t add_min = (msg[6]-'0')*10 + (msg[7]-'0');
7       uint8_t add_sec = (msg[9]-'0')*10 + (msg[10]-'0');
```

```
8
9       if (msg[5] == '+') {
10          min += add_min;
11          sec += add_sec;
12          if (sec>=60) {  // 处理进位
13              min += sec/60;
14              sec %= 60;
15          }
16          if (min>=60) {  // 处理进位
17              min %= 60;
18          }
19      } else if (msg[5] == '-') {
20          min -= add_min;
21          sec -= add_sec;
22          if (sec<0) {  // 处理借位
23              min -= 1;
24              sec += 60;
25          }
26          if (min<0) {
27              min = 0;
28              UARTStringPutNonBlocking("Invalid input!\n");
29          }
30      } else {
31          UARTStringPutNonBlocking("Invalid command!\n");
32          return;
33      }
34      char time_str[6]; // 创建一个字符数组来存储时间字符串
35      sprintf(time_str, "%02d:%02d", min, sec); // 格式化时间
36      UARTStringPutNonBlocking(time_str); // 输出时间
37  }
```