上海交通大學
SHANGHAI JIAO TONG UNIVERSITY

School of Electrical Infomation and Electronic Engineering

工科创 IV-D Lab_1 实验报告
人机交互（HMI）综合实验

第四组：林黄轩　毛钶翔　姚逸飞　郜尚　龚欣雨

HMI Experiment for Science and Technology Innovation

Course Code: EI229

2021 年 4 月 2 日

CONTENT

变量解释表

变量名	作用域	变量作用
Var1	全局（任务 1）	作为时钟的递增信号
Var2	全局（任务 2）	作为旋转编码开关控制的数字变量 (QESVar)
beepFlag	全局（蜂鸣器控制）	为避免主循环中使用 delay 导致计数延迟而使用的控制变量，用以判断蜂鸣器是否鸣响
beepCnt	全局（蜂鸣器控制）	用作蜂鸣器鸣响时间的计数变量，在时间脉冲的中断中进行累加
beepGap	全局（蜂鸣器控制）	控制蜂鸣器的鸣响时间（$5ms * beepGap$）
CountMAX	全局（蜂鸣器控制）	用以记录蜂鸣器鸣响的数值，该数值可更改
sw1Flag	全局（主循环）	用以控制主循环中状态变化后的变量初始化，如果无需初始化可注释本部分代码
initVar()	子函数	初始化控制变量与计数变量，切换间隙会在 LED 屏幕上显示 *RESET*，时间长短可变
ResetFlag	全局（任务 1）	用以实现任务 1 功能且消除抖动的布尔值，分别在进入时和 S2 按下时生效
stopFlag		
QESPFlag	全局（任务 2）	用以控制按下 QES 后的重置与防抖
appTick	全局（时间中断）	用以进行时间脉冲的中断计数
appTick2		
INTF	全局（QES 中断）	用以消除 QES 旋转时的抖动

1 实验目标与问题整理

1.1 实验目标

使用 Cyber-Dorm K66 单片机完成以下功能：

1. 定义两个变量 Var1、Var2，Var1 初值 0，Var2 初值 128
2. 通过拨码开关 SW1 切换执行以下两个任务

   a) 程序主循环计数值显示及控制

   - S1 按键（Key1）使 Var1 清零并开始主循环计数，S2 按键停止计数
   - 主循环计数值达到某一个特定数值时蜂鸣器短鸣一声
   - 主循环计数值取 0x0F 的余数显示于数码管

- 合理设置延时时长使显示字符清晰可见

- 测试“反应能力”：听到蜂鸣器鸣叫后尽快按下 S2，看计数值

b) 数码开关 QES 变量显示及控制

- QES 顺时针旋转 Var2 数值增加，Var2 最大值为十进制数 255（0xFF）

- QES 逆时针旋转 Var2 数值减小，Var2 最小值为 0

- 按下 QES 开关使 Var2 恢复初值 128（0x80）

- Var2 数值显示于光柱或 OLED

3. S1、S2 和数码开关的信号可能会出现“抖动”，通过程序处理实现简单的“防抖”功能。

1.2 问题整理

根据小组掌握的知识，将上述目标整理为以下 4 个关键问题：

1. 通过拨码开关切换不同的程序或执行不同的函数
2. 计数逻辑与计数函数，延时参数调试，按键输入与 OLED 显示、数码管输出的应用
3. 数码开关输入与液晶屏和光柱输出的应用
4. 针对不同功能的信号特点设计对应的防抖算法

2 问题分析

2.1 工作程序的切换

通过拨码开关 SW1 的输入，可以进行两个程序的切换。当拨码开关未被按下时，系统执行任务 1；当开关被按下时，系统执行任务 2。在任务状态变更后对原始变量进行重置（也可保留）。

2.2 程序主循环数值显示及控制

2.2.1 功能实现

在不同条件下通过按动按键来实现不同功能的任务容易使用分支语句配合辅助的布尔型变量来实现。计数功能通过单片机的计时器配合合理的延时函数来实

现。

本试验采用八段共阳极数码管，用来显示一位数字 8 的七个笔画与 1 位小数点，所有发光二极管的阳极接到一起形成公共阳极的数码管，其在应用时应将公共极接到 VCC 上，当某一字段发光二极管的阴极为低电平时，相应字段就点亮，反之为高电平时，相应字段就不亮。显示功能通过 **CDK66_OLED** 头文件中的函数以及 **ShowNumHEX()** 函数来实现。利用 **BOARD_I2C_GPIO()** 函数还可以实现光柱的数值显示控制。

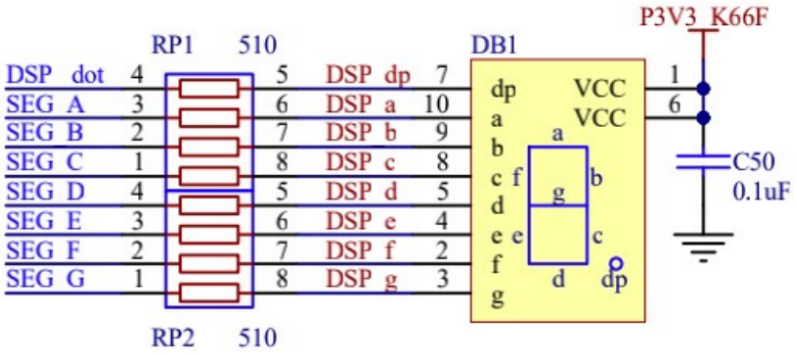


图 1: 八段共阳极数码管原理图

2.2.2 防抖方法

按键按动过程中，由于机械上的振动或电位上的不稳定，产生的信号会带有一定的跳动，而单片机的识别周期极短，可以检测到这些跳动从而产生误判，防抖的基本原理就是让通过程序滤除这些跳动。在本实验中，我们通过标志位的状态变化来实现软件防抖操作，降低程序对于延时的需求。在案件按下后，标志位拉起，直到按钮弹起后再置零。同时使用全局变量控制不会干扰其他工作的正常执行，使得按下按钮时可同时进行其他操作。

2.3 数码开关 QES 变量显示及控制

2.3.1 功能实现

该任务要求的功能较为简单，通过例程中定义好的 **QESB()**、**QESA()** 和 **QESP()** 函数，结合正交信号的原理容易设计函数实现拨码盘控制变量 Var2 增减的功能。

本实验采用的旋转编码开关，采用正交编码的方式，通过判断高低电平可以读取旋转方向。若采取 A 信号上升沿时调用程序，则通过在中断服务程序中读取的 B 信号的状态，可知 B 为高电平，说明为正转。反之，当 B 为低电平时，说明为反转。旋转编码开关的原理如下图所示：

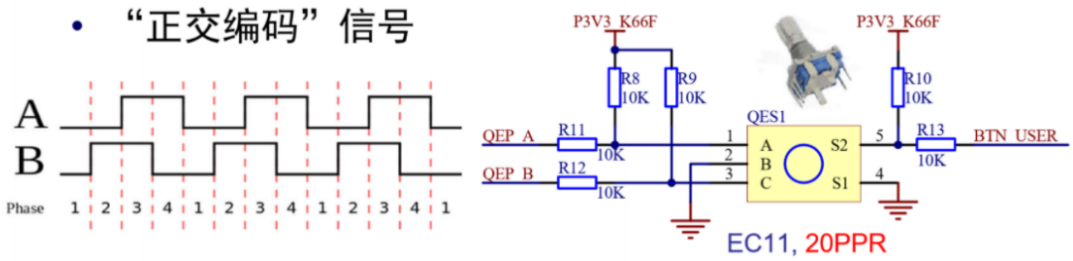


图 2: 旋转编码开关原理图

2.3.2 防抖方法

由于编码盘使用正交编码，可发现在 A 相的上升沿与下降沿之间会发生 B 相的转换，任何一相在正常情况下不会发生连续的变换。因此需要确认读取 B 相状态时 A 相应处于上升沿触发后、下降沿触发前。最为合理的防抖方法是配置 A 相与 B 项的双中断，仅当读取到两个不同的中断后再进行状态变化。但程序使用的中断配置有限，无需为此多添加一个中断串口，所以我们通过对 A 的上升沿与下降沿两侧的中断监控进行防抖处理。

A 相出现漂移时可能会导致短时间多次触发上升沿中断，这也是抖动的主要原因。我们设置 A 相上升沿中断标志位 **INTF**，在下降沿触发时再将其置零，防止连续的状态变化。这样可以避免在中断中使用长时间的 delay 导致中断持续被占用，提高程序的运行效率。

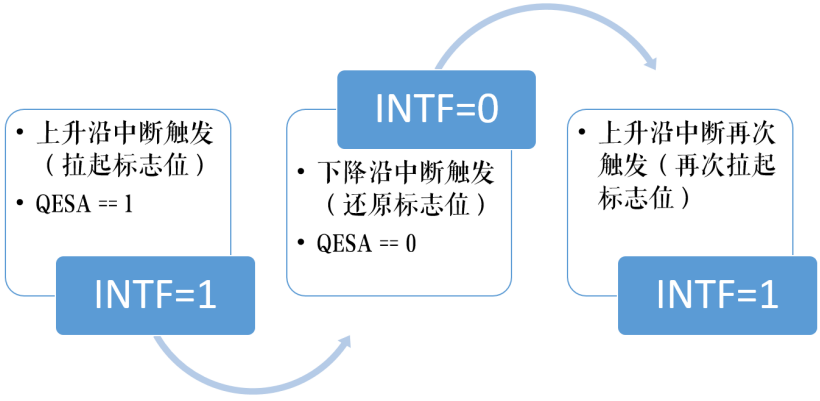


图 3: 防抖实现原理示意图

3 程序开发

3.1 任务 1 代码实现

3.1.1 程序主体

程序主体部分代码如下所示

Listing 1: **void Task1**

```
1  void Task1(void)
2  {
3          OLED_P8x16Str(0,0,"          ");
4          BOARD_I2C_GPIO(0);
5          if (KEY1())
6          {
7                  ResetFlag = 0;
8                  delay();//avoid shifting accidents
9          }
10         else
11         {
12                 if(!ResetFlag)
13                 {
14                         Var1 = -1;//inital value -1 can display 0 for 1 sec
15                         appTick2 = 0;
16                         ResetFlag = 1;
17                         beepCnt = beepFlag = stopFlag = 0;
18                         BEEP_OFF();
19                 }
20         }
21         if (!KEY2())
22         {
23                 stopFlag = 1;
24                 BEEP_OFF();
25         }
26         if((appTick2%200) == 0)
27         {
28                 if(!stopFlag) Var1++;
29                 ShowNumHEX(Var1);
30                 if(Var1 == CountMAX && !beepFlag) {
31                         BEEP_ON();
32                         beepFlag = 1;
33                 }
34                 while ((appTick2%200) == 0) {}
35         }
36 }
```

代码禁用了 OLED 屏幕与光柱，其中 5~16 行使用 **ResetFlag** 标识符进行 S1 的抖动消除，而 17~25 行则实现了 S2 的抖动消除。二者均在弹起后进行重置，并

通过全局变量保证了标识符在不同循环中的传输。26~34 行实现了 **Var1** 的按秒递增，并通过 **while 循环检测 appTick2 的变化**以实现准确的秒控制。程序在主体中执行蜂鸣器的鸣响并挂起鸣响标志，在鸣响的瞬间按下 S2 可终止蜂鸣器鸣响，用以模拟闹钟功能。如不进行重置，则蜂鸣器会在鸣响一定时间后停止。

代码第 29 行实现了数字在八位数码管上的显示，通过调用 **ShowNumHex** 函数显示 16 进制的最低位数字。单个数值显示时间为 1 秒。

3.1.2 中断处理

中断处理部分代码如下所示

Listing 2: **Task 1 中断处理**

```
1  void APPTICK_PIT_HANDLER(void)
2  {
3      /* Clear interrupt flag.*/
4      PIT_ClearStatusFlags(PIT, kPIT_Chnl_0, kPIT_TimerFlag);
5      appTick++;
6      appTick2++;
7      if(beepFlag && (beepCnt < beepGap))
8      {
9          beepCnt++;
10     }
11     else BEEP_OFF();
12
13 #if defined __CORTEX_M && (__CORTEX_M == 4U)
14     __DSB();
15 #endif
16 }
```

在时钟信号的基础中断中，我们加入了鸣响标识符被挂起的判断。鸣响标志 **BeepFlag** 在Listing 1中被挂起并启动中断中的蜂鸣器计时。在蜂鸣器鸣响达到指定时间后终止计时、置零标识符并关闭蜂鸣器。

3.2 任务 2 代码实现

3.2.1 程序主体

任务 2 的程序主体部分如下所示

Listing 3: **Task2**

```
1  void Task2(void)
2  {
3          //OLED_Init();
4          ShowNumOFF();
```

```
        if(QESP())
        {
                QESPFlag = 1;
                delay();//in order to avoid accidents
        }
        else if(!QESP() && QESPFlag){
                Var2 = 0x80;
                QESPFlag = 0;
        }
        OLED_Print_Num(0,0,Var2);
        BOARD_I2C_GPIO(Var2);
}
```

程序主体主要实现了旋转编码开关按下后对 Var2 的初始化，通过 QESPFlag 实现对这一进程的控制与防抖。同时通过 **OLED_Print_Num 函数和 BOARD_I2C_GPIO 函数**使得 Var2 变量可以输出到 OLED 显示屏和光柱。

3.2.2 中断处理

任务 2 中断处理部分如下所示

Listing 4: **Task2 中断处理**

```
void BOARD_RESa_IRQ_HANDLER(void)
{
    /* Clear external interrupt flag. */
        GPIO_PortClearInterruptFlags(BOARD_INITPINS_QESa_GPIO,
            BOARD_INITPINS_QESa_GPIO_PIN_MASK);
    if(QESA() && !INTF)
    {
        if (!QESB()) Var2++;
        else if (Var2 > 0) Var2--;
        INTF = 1;
    }
    else if(!QESA())
    {
        INTF = 0;
    }
    Var2 %= 256;
    delay();//Delete this line if unnecessary.
#if defined __CORTEX_M && (__CORTEX_M == 4U)
    __DSB();
#endif
}
```

这一中断使用 A 相的上升沿与下降沿触发，因此需要在 pin-> 路由详情中配置 99 号对应的 QESa 触发方式为 *interrupt on either edge*。

代码分别判断上升沿中断与下降沿中断的情况。当上升沿触发后，QESA() 读

取到高电平，此时判断上升沿的操作是否被访问过，如果被访问过则直接跳过以消除抖动，如果没有被访问过则进入条件并依据 B 相的状况改变 Var2 的数值。当下降沿触发后，QESA() 读取到低电平，此时访问上升沿的次数被清零，可以执行下一次变化。具体执行流程可以参考图3。

由于 Var2 的最大值为 255，程序中对其执行模运算并保留模 256 的余数。

3.3 任务切换与重置

任务切换的判断在主程序的 while 循环中执行，其代码如下：

Listing 5: 主函数 **while** 循环

```
while(1) {
    if(SW1()){
        if(!sw1Flag){
                sw1Flag = 1;
                initVar();
        }
        Task1();
    }
    else{
        if(sw1Flag){
                sw1Flag = 0;
        }
        Task2();
        }
}
```

程序通过标识符实现状态的切换与重置。切换时只需监督判断 SW1 的状态，如果其读数为 1 则执行 Task1，反之则执行 Task2。通过 SW1Flag 可以判断状态是否发生变化。如果发生变化则对变量进行初始化。由于两个任务的变量不相关，所以在两次切换中只需执行一次初始化。

对于初始化代码，我们添加了一些特殊功能，具体代码如下：

Listing 6: 初始化

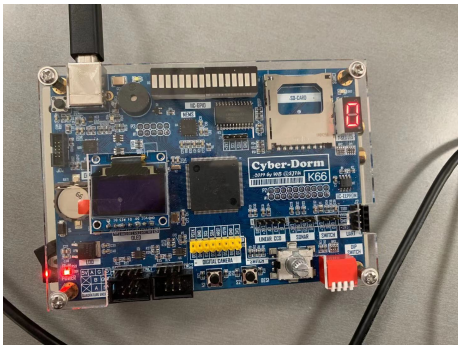
```
void initVar(void)
{
        beepFlag        =       0;
        beepCnt         =       0;
        ResetFlag       =       0;
        stopFlag        =       0;
        appTick         =       0;
        appTick2        =       0;
        Var1            =       0;
        Var2            =       0;
        QESPFlag        =       1;
```

```
12          INTF              =       0;
13          ShowNumOFF();
14          OLED_P8x16Str(0,0," RESET NOW  ");
15          delay_long();
16          OLED_P8x16Str(0,0,"            ");
17          BOARD_I2C_GPIO(0);
18  }
```

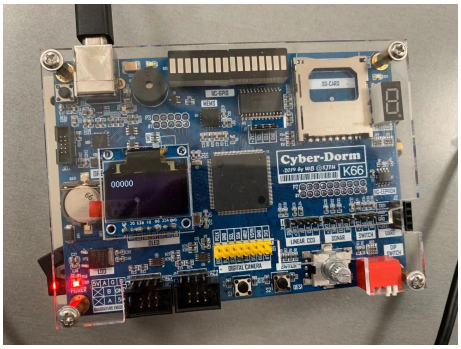
在变量初始化的过程中，我们让 OLED 屏幕输入一段时间的“RESET”，表示其正在执行代码的初始化[1]。

4 实验结果

4.1 使用拨码开关 SW1 切换任务

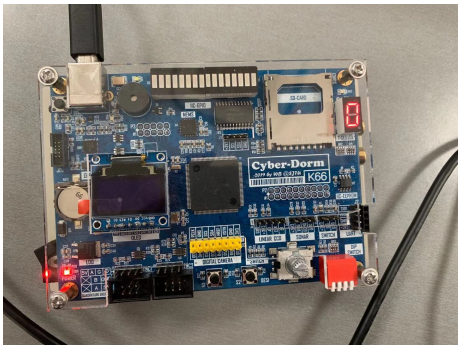


(a) 按下开关前

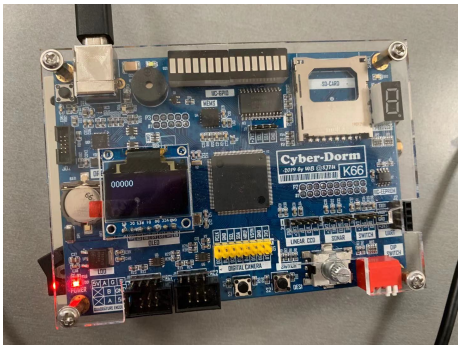


(b) 按下开关后

4.2 任务 1 的执行



(c) 显示余数为 0



(d) 显示余数为 E

[1]其实代码的执行非常短暂，但是我们想让这一过程看上去更有仪式感

在合理设置延时时长后，可以顺利完成计数、暂停、启动、置零和特定数值提示并计数的功能，并可以清楚地观察到 Var1 变量的每次变化。按下 S2 后，计时停止；按下 S1 后，计时清零。以上内容连续进行 20 次操作未出现因抖动问题而产生的异常情况。