

嵌入式案例分析与设计-综合实验设计报告

实验名称：红绿灯管理系统

实验目的：

- 1.进一步掌握本学期以来所做实验用到的各种元器件的使用方法和编程；
- 2.加深 GPIO、中断、定时器、LED、uart、A/D 等的使用；
- 3.加强综合问题解决能力和拓展思维创新能力。

功能需求：

- 1.红绿灯有两种模式：正常模式&行人模式

正常模式的工作方式如下：

行人过（东西方向绿灯，南北方向红灯，保持 2s）

行人和车辆等待交通灯切换状态（两个方向黄灯，保持 1s）

车辆过（南北方向绿灯，东西方向红灯，保持 2s）

行人和车辆等待交通灯切换状态（两个方向黄灯，保持 1s）

行人模式：

行人过（东西方向绿灯，南北方向红灯，保持状态）；

由于板子只有一个 led 灯，设计为只显示东西方向交通灯状态；

- 2.手动选择模式：

由按键实现在两种模式之间切换。

- 3.a/d 采样设定流量：未实现。

- 4.改变红绿等显示时间：

使用 easyarm 通过串口发送更改红绿灯显示时间的信息

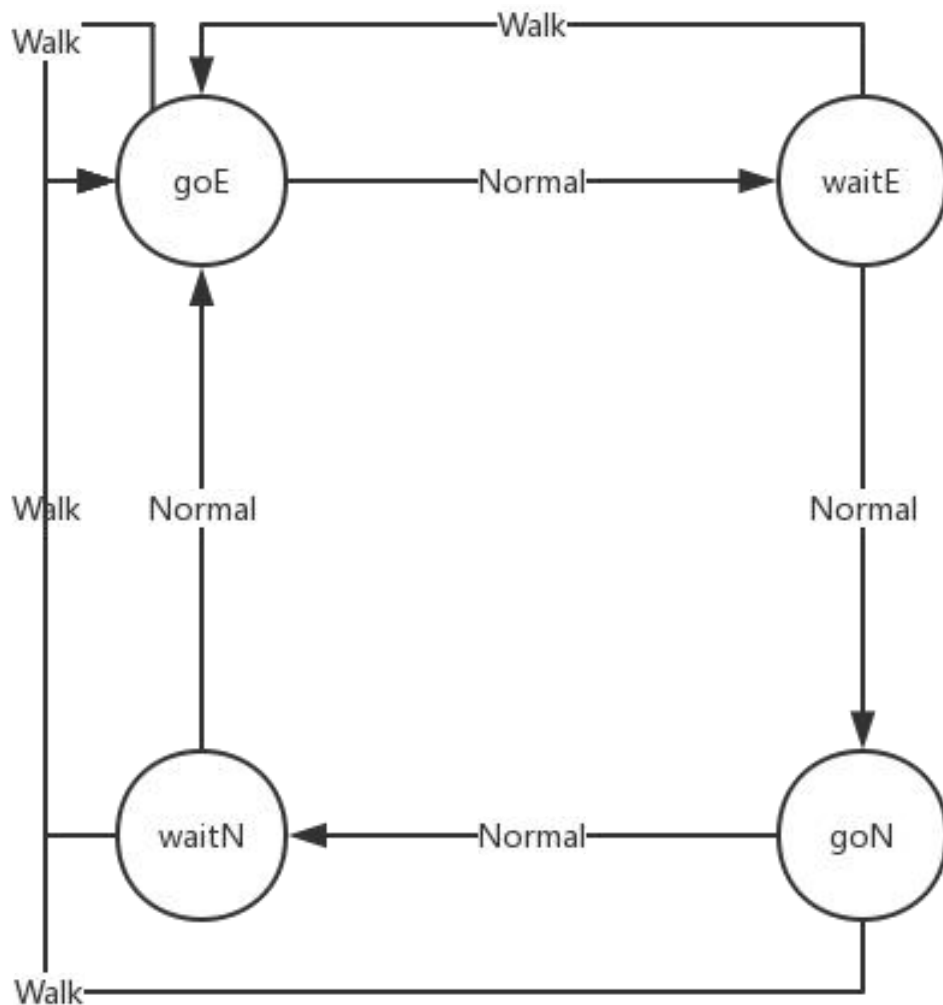
- 5.显示当前模式状态信息：

两种模式切换和交通灯状态切换时，通过串口向 PC 机发送模式状态信息，通过 easyarm 接收并查看。

功能设计：

- 1.红绿灯两种模式：

通过有限状态机实现两种模式，四种状态的切换。



2.按键切换两种模式:

设计通过按键的中断的边缘触发模式配合延时消抖思想,即可实现在按键的中断处理函数里面切换两种模式.

3.a/d 采样设定流量: 无

4.改变红绿灯显示时间:

设计采取了按键的中断配合串口通信的查询方式的两种方法的组合。在按键中断触发时,判断是否为特定的按键按下,如果是指定的按键按下则陷入阻塞,查询串口输入。串口接收输入后从阻塞中恢复,根据接收到的值结合有限状态机(结合设计 1)设定红绿灯的显示时间。

5.显示当前状态信息:

设计当状态切换(结合设计 1)时和模式切换(结合设计 2)时,通过串口向 PC 机发送当前的状态和模式信息。

功能实现:

1.有限状态机:

状态机的结构体结构

```
struct State {  
    unsigned long Out;  
    unsigned long Time;  
    unsigned long Next[8];};  
typedef struct State STyp;
```

第一个字段 Out 表示 Led 灯的输出

第二个字段 Time 表示状态持续时间

第三个字段表示在特定模式下的下一状态

```
#define goE 0  
#define waitE 1  
#define goN 2  
#define waitN 3  
#define walkState 1  
#define normalState 0
```

```
STyp FSM[5]={  
    {0x08,200,{waitE,goE}},  
    {0x0A, 100,{goN,goE}},  
    {0x02,200,{waitN,goE}},  
    {0x0A, 100,{goE,goE}}  
};
```

FSM 四个状态分别是：行人走（东西方向绿）

两个方向等待（东西方向黄灯）

车辆走（东西方向红灯）

两个方向等待（东西方向黄灯）

分别持续 200*10ms,100*10ms,200*10ms,100*10ms;

下一个状态分别是：

本次状态	行人模式	正常模式
goE	goE	waitE
waitE	goE	goN
goN	goE	waitN
waitN	goE	goE

此部分主要逻辑代码：

```
while(1){  
    LIGHT = FSM[S].Out; // set lights  
    SysTick_Wait10ms(FSM[S].Time);  
    S = FSM[S].Next[state];  
}
```

2. 按键切换两种模式：

按键中断：配置 PF0 和 PF1 为下降沿边缘触发：

```
SYSCCTL_RCGCGPIO_R |= 0x00000020; // (a) activate clock for port F
```

```
FallingEdges = 0; // (b) initialize counter
```

```
GPIO_PORTF_LOCK_R = 0x4C4F434B;
```

```
GPIO_PORTF_CR_R=0x03;
```

```

        GPIO_PORTF_DIR_R &= ~0x11; // (c) make PF0/4 in (built-in button)
        GPIO_PORTF_AFSEL_R &= ~0x11; // disable alt funct on PF0/4
        GPIO_PORTF_DEN_R |= 0x11; // enable digital I/O on PF0/4
        GPIO_PORTF_PCTL_R &= ~0x000F000F; // configure PF0/4 as GPIO
        GPIO_PORTF_AMSEL_R = 0; // disable analog functionality on PF
        GPIO_PORTF_PUR_R |= 0x11; // enable weak pull-up on PF0/4
        GPIO_PORTF_IS_R &= ~0x11; // (d) PF0/4 is edge-sensitive
        GPIO_PORTF_IBE_R &= ~0x11; // PF0/4 is not both edges
        GPIO_PORTF_IEV_R &= ~0x11; // PF0/4 falling edge event
        GPIO_PORTF_ICR_R = 0x11; // (e) clear flag0/4
        GPIO_PORTF_IM_R |= 0x11; // (f) arm interrupt on PF0/4 *** No IME bit as
mentioned in Book ***

```

```

        NVIC_PRI7_R = (NVIC_PRI7_R & 0xFF00FFFF) | 0x00A00000; // (g) priority 5
        NVIC_EN0_R = 0x40000000; // (h) enable interrupt 30 in NVIC
        EnableInterrupts(); // (i) Clears the I bit

```

按照上述代码初始化 PF0/4 的下降沿触发中断，即可在两者按下时触发中断，我们在中断处理函数里面处理逻辑

```

void GPIOPortF_Handler(void){
    GPIO_PORTF_ICR_R = 0x11; // acknowledge flag0/4
    if((GPIO_PORTF_DATA_R & 0x01) == 0x01){
        state = 1 - state; // switch state
    }
    if((GPIO_PORTF_DATA_R & 0x10) == 0x10){
        // todo
    }
}

```

这样已经实现了按键切换状态，但是由于按键触发存在抖动现象，不得不引入消抖程序，即加入时延。

消抖的思想是，对于在每次成功的中断后的一段时间内再次触发的中断不再处理。因此引入了定时器，并通过定义一个时间戳的变量记录当前时间。

```

#define F16HZ (50000000/16)
int timestamp = 0;
void UserTask(void){
    timestamp += 1;
}
Timer0A_Init(&UserTask, F16HZ); // initialize timer0A (16 Hz);

```

在主函数中调用此函数，设定 timer0A 每 1/16s 触发一次时钟中断，调用 UserTask() 函数。我们将 1/16s 作为一个时间单位，timestamp 每过 1/16s 自动+1，起到了记录时间的作用。

然后在按键中断中添加时延的判断：

```

int last = 0;
void GPIOPortF_Handler(void){
    int now = timestamp;
    GPIO_PORTF_ICR_R = 0x11; // acknowledge flag0/4
    if(now - timestamp < 10){
        last = timestamp;
        return
    }
};

```

```
//your logic
last=timestamp;
```

```
}
```

在每次按键中断触发后，如果和上次按键中断时间间隔太小，则将此次中断视为抖动现象，并更新 last 为当前时间，每次中断触发处理后，最后都将 last 更新为当前时间。

由此完成了按键中断的抖动处理和模式切换。

3. a/d 采样设定流量：未实现。

4. 改变红绿灯显示时间。

(1) 配置串口

```
void UART_Init(void){
    SYSCTL_RCGC1_R |= SYSCTL_RCGC1_UART0; // activate UART0
    SYSCTL_RCGC2_R |= SYSCTL_RCGC2_GPIOA; // activate port A
    UART0_CTL_R &= ~UART_CTL_UARTEN; // disable UART
    UART0_IBRD_R = 27; // IBRD = int(50,000,000 / (16 * 115,200)) =
int(27.1267)
    UART0_FBRD_R = 8; // FBRD = int(0.1267 * 64 + 0.5) = 8
// 8 bit word length (no parity bits, one stop bit, FIFOs)
    UART0_LCRH_R = (UART_LCRH_WLEN_8|UART_LCRH_FEN);
    UART0_CTL_R |= UART_CTL_UARTEN; // enable UART
    GPIO_PORTA_AFSEL_R |= 0x03; // enable alt funct on PA1-0
    GPIO_PORTA_DEN_R |= 0x03; // enable digital I/O on PA1-0
// configure PA1-0 as UART
    GPIO_PORTA_PCTL_R =
(GPIO_PORTA_PCTL_R&0xFFFFF00)+0x00000011;
    GPIO_PORTA_AMSEL_R &= ~0x03; // disable analog functionality on
PA
}
```

(2) 使用串口：

本次程序中仅使用了

```
unsigned long UART_InUDec(void);
```

```
void UART_OutString(char *pt);
```

两个方法。

在按键中断中插入语句：

```
void GPIOPortF_Handler(void){
    GPIO_PORTF_ICR_R = 0x11; // acknowledge flag0

    if((GPIO_PORTF_DATA_R&0x10)==0x10){
        unsigned long s;
        UART_OutString("input time to goE/goN:(seconds)");
        OutCRLF();
        s=UART_InUDec();
        FSM[0].Time=s*100;
        FSM[2].Time=S*100;
        UART_OutString("goE/goN states now last for ");
        UART_OutUDec(s);
        UART_OutString(" seconds");
```

```

        OutCRLF();
    }
    last = timestamp;
}

```

消抖方法见功能实现 2

这样在按下 PF4 时，通过 UART_InUDec（）获取设定的时间，通过对 FSM[0].Time 和 FSM[2].Time 的设定来改变两个状态的持续时间。

5. 输出状态模式信息

（1）在功能实现 1 的 main 函数的死循环里插入语句：

```

while(1){
    UART_OutString("traffic light state:\t");
    if(S==goN){
        UART_OutString("goN");
    }
    else if(S==goE){
        UART_OutString("goE");
    }
    else if(S==waitN){
        UART_OutString("waitN");
    }
    else if(S==waitE){
        UART_OutString("waitE");
    }
    else{
        UART_OutString("INVALID STATES");
    }
    OutCRLF();
    LIGHT = FSM[S].Out; // set lights
    SysTick_Wait10ms(FSM[S].Time);
    Input = SENSOR;    // read sensors
    S = FSM[S].Next[state];
}

```

即可在状态切换时向串口发送状态信息；

（2）在功能实现 2 的按键中断处理函数里插入语句：

```

void GPIOPortF_Handler(void){
    GPIO_PORTF_ICR_R = 0x11;    // acknowledge flag0
    if((GPIO_PORTF_DATA_R&0x01)==0x01){

        state=1-state;
        if(state==walkState){
            UART_OutString("Changed to walker mode");
        }
        else{
            UART_OutString("Changed to normal mode");
        }
        OutCRLF();
        FallingEdges = FallingEdges + 1;
    }
}

```

```
}  
}
```

即可实现在模式切换时向串口发送模式信息。

实验结果展示：


板子上的 led 灯现象不具有说明性，只截取 easyarm 的输出作为实验结果展示：

```
Traffic Light Management Program started!  
normal mode  
state goE/goN last for 2 seconds  
state waitE/waitN last for 1 seconds  
traffic light state:    goN  
traffic light state:    waitN  
traffic light state:    goE  
traffic light state:    waitE
```

发送数据

4

☐ 以十六进制格式发送 ☒ 按键即发送

高级  发送 清空 取消

接收窗口 ☐ 十六进制方式显示 ☒ 停止显示

```
traffic light state:    goN  
Changed to walker mode  
traffic light state:    goE  
Changed to normal mode  
traffic light state:    waitE  
input time to goE/goN:(seconds)  
4goE/goN states now last for 4 seconds  
traffic light state:    goN
```