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

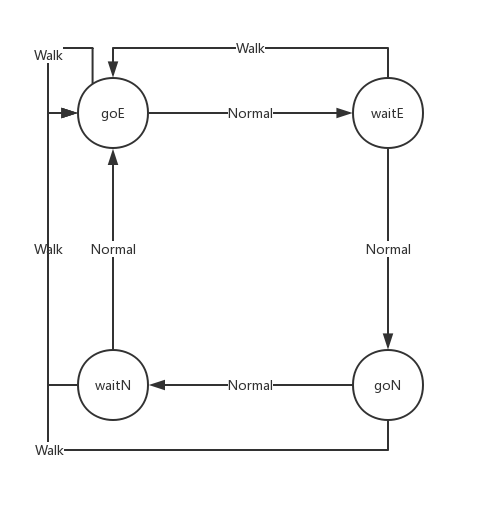
5.显示当前模式状态信息：

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eaysarm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];

}

1. 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：

SYSCTL\_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。

1. a/d采样设定流量：未实现。
2. 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&0xFFFFFF00)+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。

1. 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：

