



**单片机应用课程设计**

题目： 基于51单片机和手机app自定蓝牙通信协议

的滑动无极调光多路LED灯控制设计

|  |  |
| --- | --- |
| **专业班级** 19通信二班 |  |
| **学 号** 190340226 |  |
| **姓 名** 刘良浩 |  |
| **评 阅 人** |  |
|  |  |

**2021年11月**

**中国 马鞍山**

目录

[一 目的： 1](#_Toc89896578)

[二 项目器材： 1](#_Toc89896579)

[三 使用软件： 1](#_Toc89896580)

[五 单片机开发 1](#_Toc89896581)

[1. 程序逻辑 1](#_Toc89896582)

[2. LED的亮度控制： 2](#_Toc89896583)

[3. 设计通信协议： 3](#_Toc89896584)

[4. 对于头尾校验和超时检测 5](#_Toc89896585)

[5. 其它部分 6](#_Toc89896586)

[a. 主函数 6](#_Toc89896587)

[b. 发送模块 6](#_Toc89896588)

[c. 对于变量的定义 6](#_Toc89896589)

[六、 手机app的开发 7](#_Toc89896590)

[1. 开发界面 7](#_Toc89896591)

[2. 逻辑图（下） 7](#_Toc89896592)

[3. 代码解析 7](#_Toc89896593)

[a. 初始化 7](#_Toc89896594)

[b. 蓝牙配置 7](#_Toc89896595)

[c. 实时调光 9](#_Toc89896596)

[d. 定时LED 9](#_Toc89896597)

[e. 其他模块（未开发） 9](#_Toc89896598)

[4. 界面预览 10](#_Toc89896599)

[七、 附件 11](#_Toc89896600)

**基于51单片机和手机app自定蓝牙通信协议**

**的滑动无极调光多路LED灯控制设计**

刘良浩190340226

# 一 目的：

设计通信协议，使用蓝牙模块连接单片机，使手机app对蓝牙模块发送控制命令实现对多个LED的亮度控制，控制完成后单片机反馈控制信息于手机显示。

# 二 项目器材：

STC89C52RC开发板、HC-04蓝牙模块、蓝牙串口转换模块、杜邦线

# 三 使用软件：

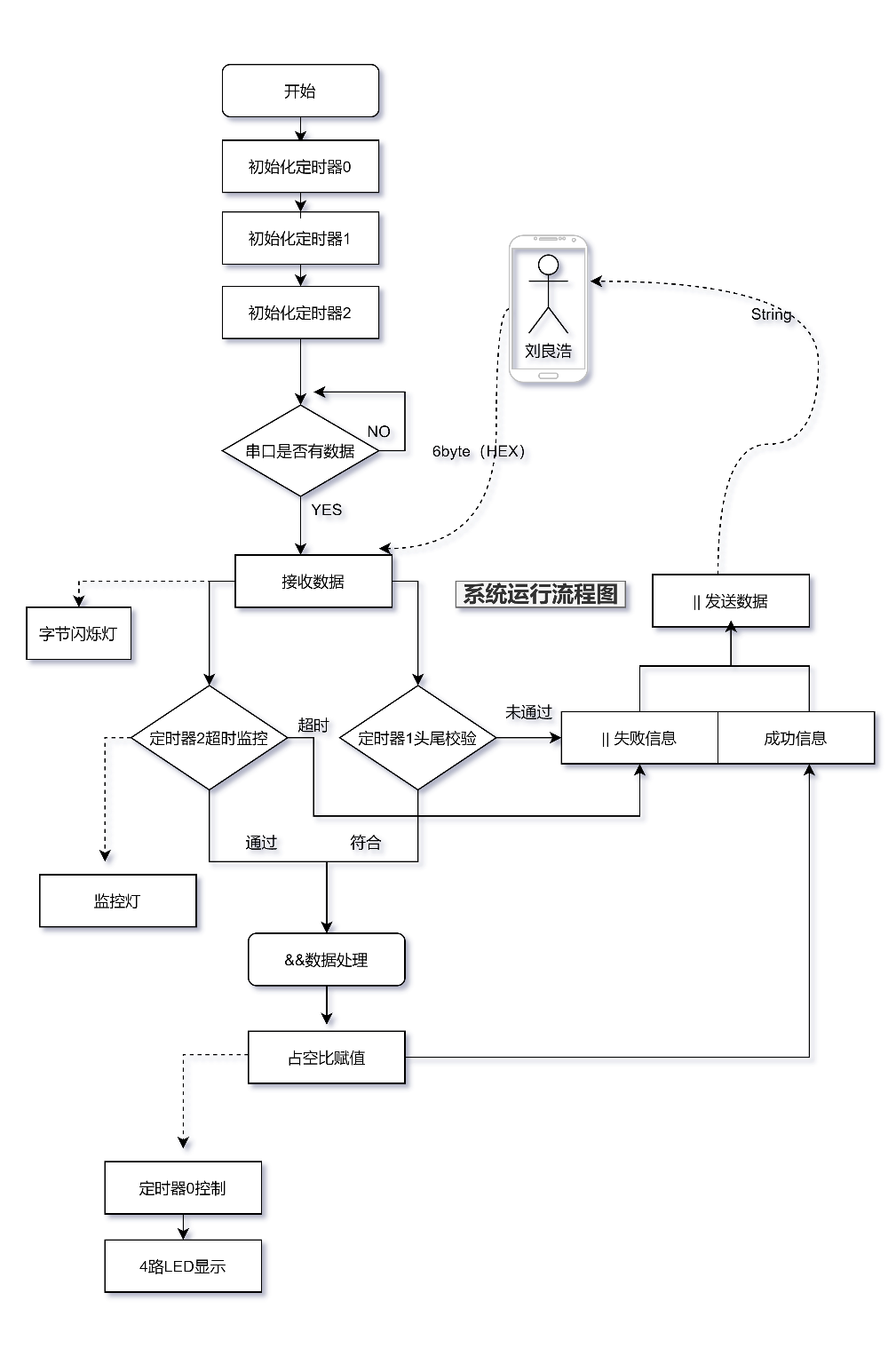
VSCode（源代码编写）、keil u5（编译）、stc-isp（烧录调试）、diagrams（逻辑图）

App Inventor（手机App开发）wxbit调试助手（用于手机app调试）

photoshop2022（图形界面制作）

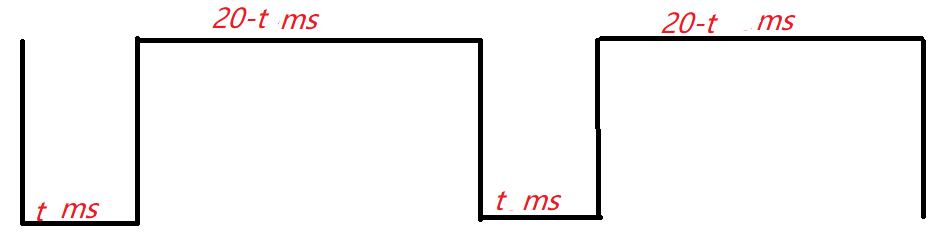
# 五 单片机开发

## 程序逻辑



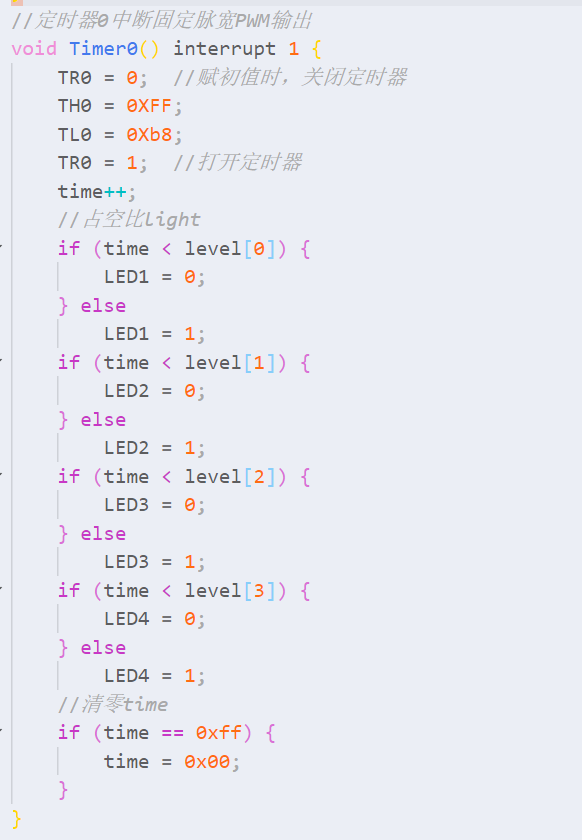
## LED的亮度控制：

LED的亮度控制由单片机产生的PWM信号实现，PWM信号是由定时器定时实现。信号的周期20ms，定时一种是t毫秒，第二种是20-t毫秒。通过调节t的时间可以实现LED的亮度调节控制。



计算方法：由于使用256级亮度，本实验采用单定时器控制中断时间达到输出pwm波，控制低电平的占空比实现led灯亮度的控制。

晶振频率11.0952MHZ，机器周期为12倍时钟周期。所以定时器计数一次需要12/f。使用定时器0方式1（16位计数）计数，最大计数值为65536。Pwm信号周期为20ms，分成256次，所以每次进入中断需要78微秒，T0=65536-78\*11.0952/12≈0XFFB8。



## 设计通信协议：

控制指令由以下格式构成：

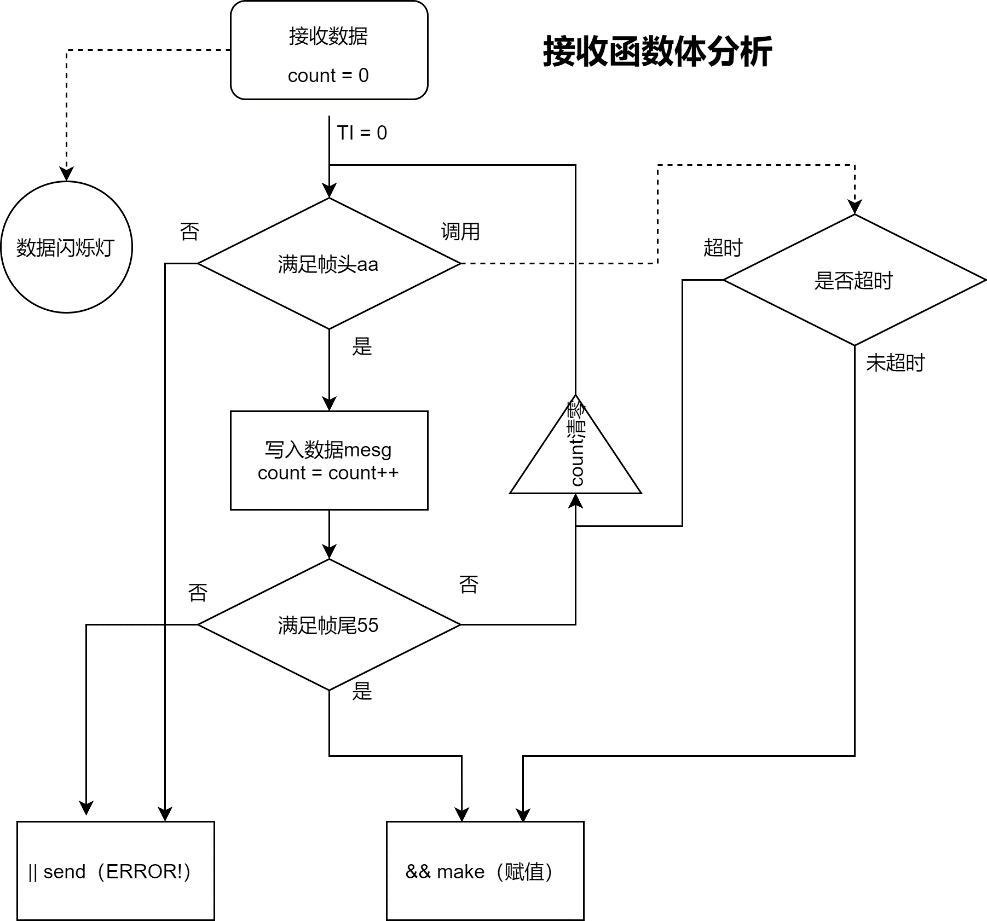
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 头 | LED1亮度 | LED2亮度 | LED3亮度 | LED4亮度 | 尾 |
| 0xaa（固定） | 256级亮度设定:0x00-0xff | 256级亮度设定:0x00-0xff | 256级亮度设定:0x00-0xff | 256级亮度设定:0x00-0xff | 0x55（固定） |

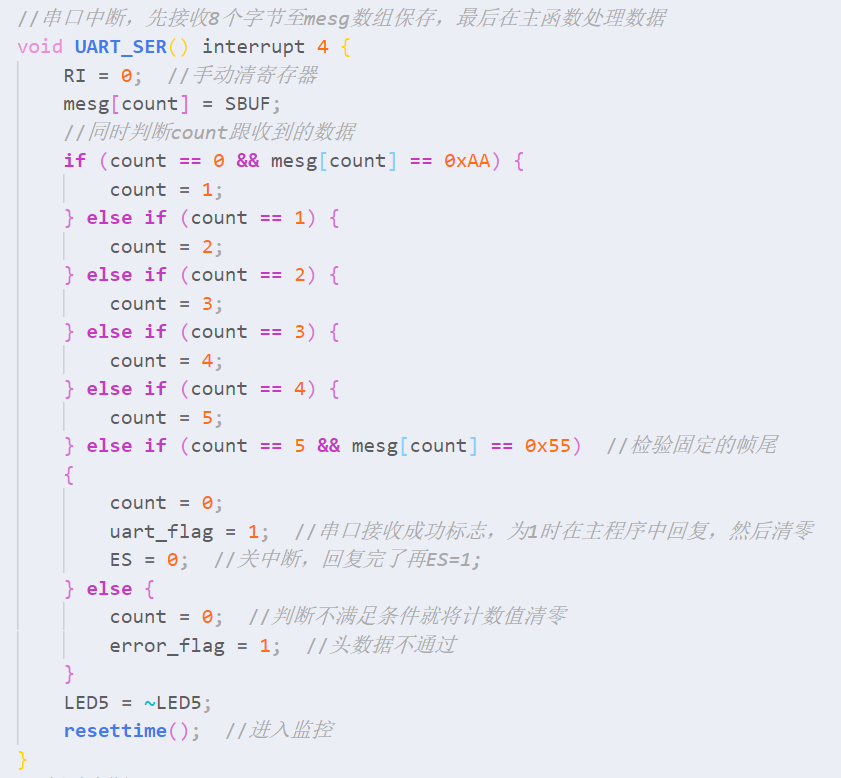
先依次接收8个字节至数组保存，使用级联的 if-else-if 多分支结构。最后再分析处理数据，在测试的时候发现了bug，如果数据帧发送一半，然后突然停止，再来重新发，就会丢失一帧的数据。比如先接受到aa 然后断了，再进来aa 01 01，就不受控制了。

一开始想过使用数组移位存数据，但是需要用到循环，可能会在通信过快时，占用太多时间。

对于数据突然中断的bug，一开始我把if(count==0&&mesg[count]==0xaa)改成了if(mesg[count]==0xaa)其他都没变，这样每次输入AA头可当成从头开始，但是因为数据段存在调节led亮度0XAA，于是想到使用时间监控整个帧发送的时间，第一个方案使用另一个定时器，由于已使用调光，且51单片机无法实现多线程任务，只能实现单线程不能再用，后改成延时函数来进行监控，解决此bug，但是使用延时函数缺点会导致发送以及数据处理时间耽搁。后想到C52单片机有定时器2，于是使用定时器2对串口进行数据传输进行超时检测。

逻辑图：





## 对于头尾校验和超时检测

当头数据正确时，开始进入定时器2，进行检测数据帧发送是否超时，如果超时将把count清零且发送错误信息与控制端。使用定时器2还需要注意配置sfr T2MOD = 0xC9;定时器2寄存器地址，并且手动清零标志位。Run数据类型采用unsigned short弥补unsigned char仅有256不足。





这种方法的确本人想的，别人可能会做的差不多，但绝不是抄袭或模仿，但是缺点是多占用一个定时器，并且中断函数内容变多了，消耗了一部分时间

## 其它部分

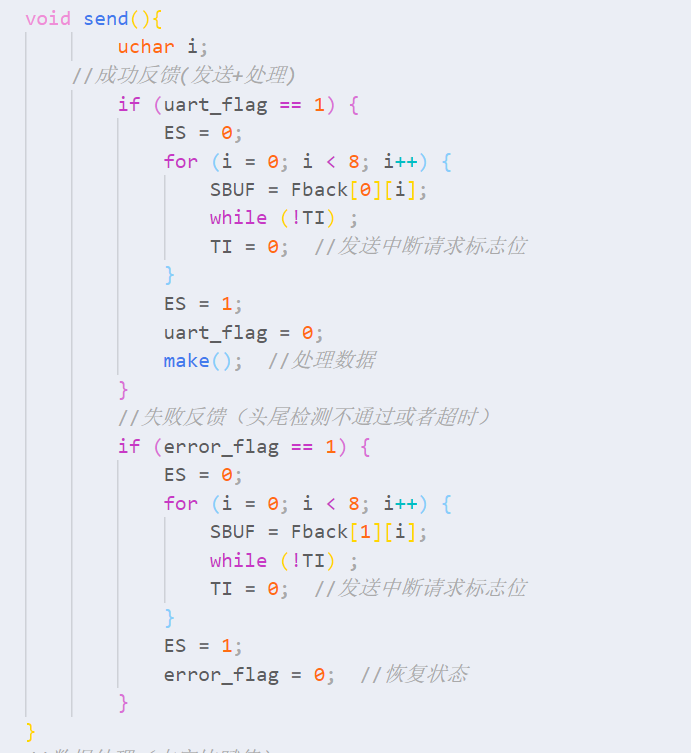
### 主函数

循环判断



### 发送模块

写了两个判断，分别对应通信的两个状态（成功和失败）。I使用局部变量，用完释放内存。



### 对于变量的定义

对于uart\_flag, error\_flag 仅表示状态使用静态全局位变量static bit节省空间。Mesg[]，level[]两个数组分别用于接收的数据存储，以及PWM的占空比。一个code Feback[][]二维数组由于值不需改变，为节省RAM存储在ROM中用于反馈成功或失败的字符串信息

# 六、 手机app的开发

虽然通过蓝牙调试助手可实现控制，但是人机交互性太差，每次需要依次发送6个字节数据包，一方面效率低，另一方面，很容易在输入过程中，发送一个错误字节，达不到需要的效果。

由于预算在课设时间不足，于是我使用Google App Inventor进行5小时的简单开发，在线开发的Android编程环境，使用积木式的堆叠法来完成Android程序。

现在要用手机发送一串指令“AA 01 02 12 45 55”，每次指令都要手打很烦，有必要整一个整合的工具，能一键发送指令。

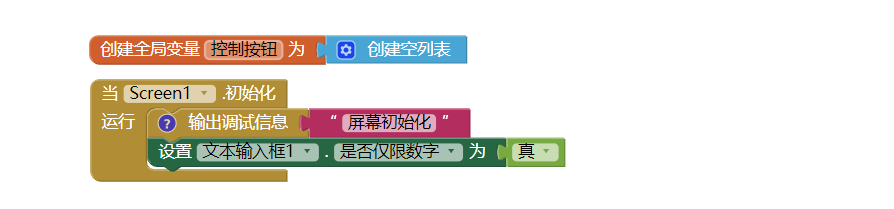
## 开发界面



## 逻辑图（下）

## 代码解析

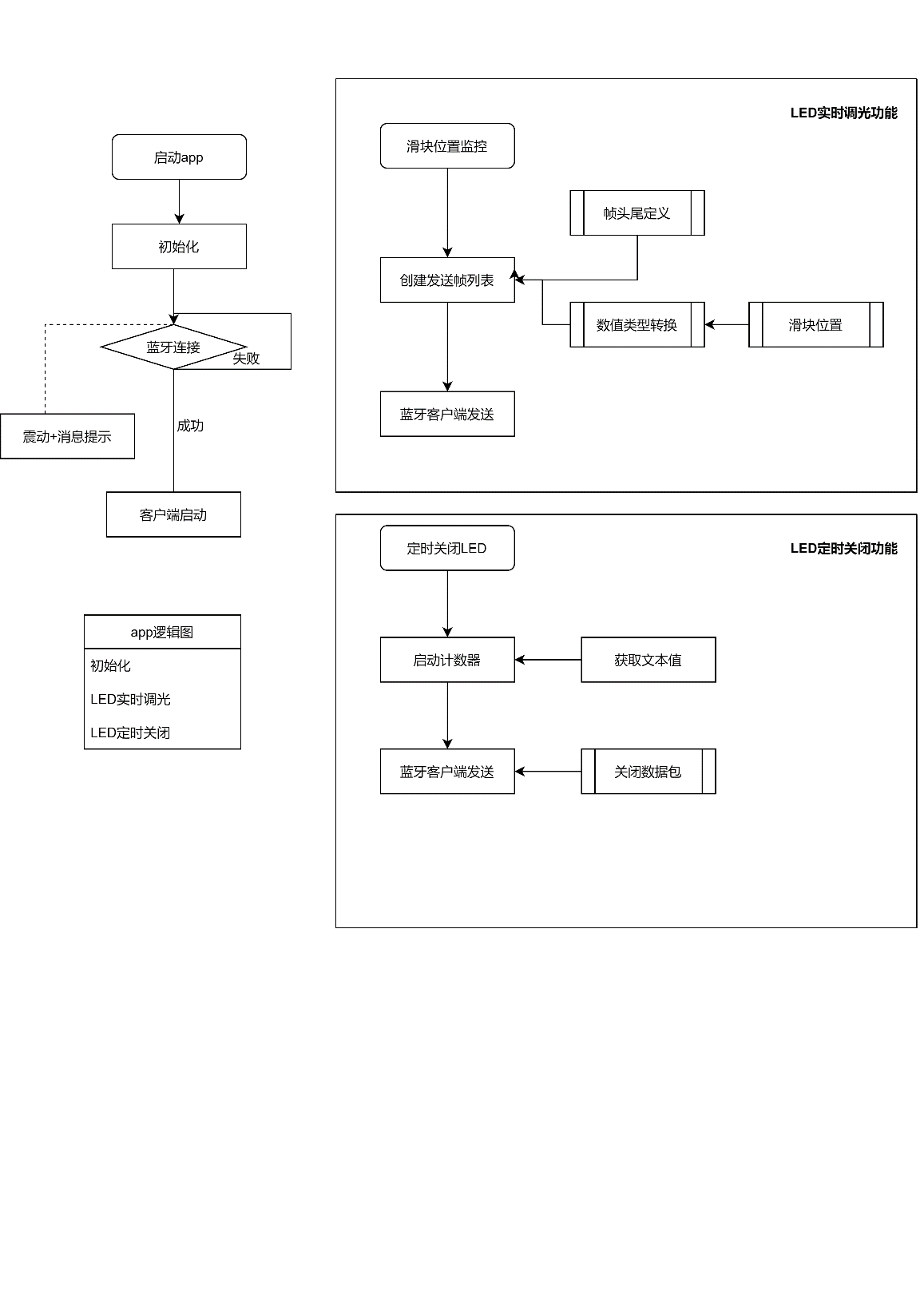
### 初始化



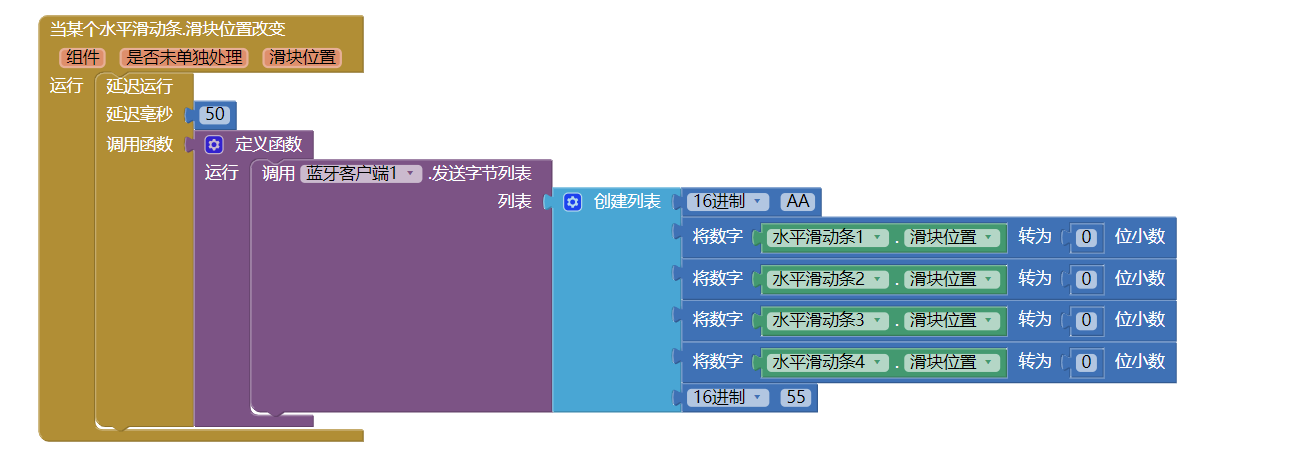
### 蓝牙配置



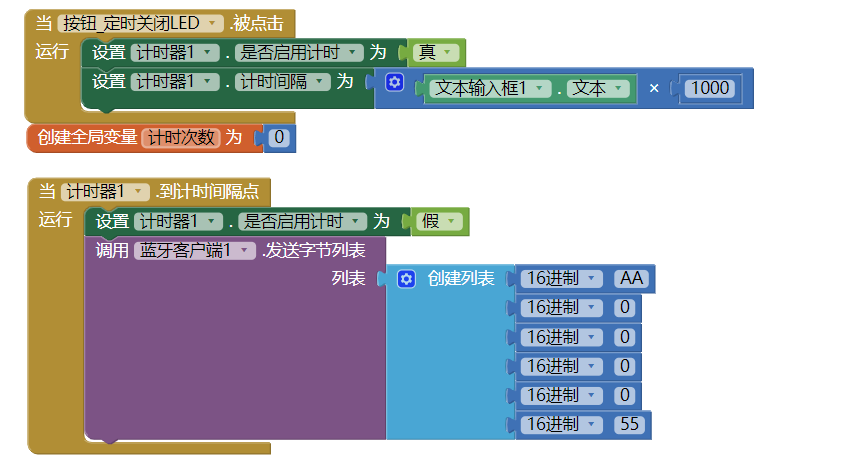




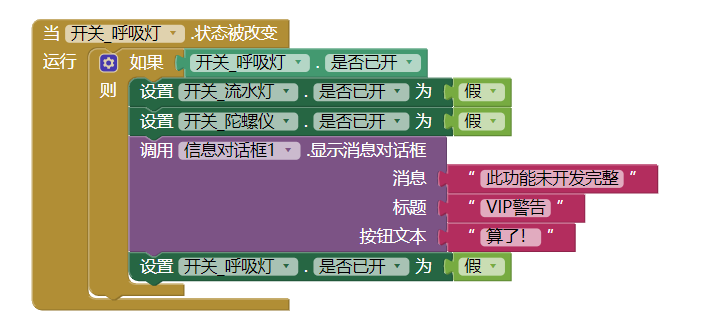
### 实时调光



### 定时LED



### 其他模块（未开发）



## 界面预览





# 七、 附件

#include <reg52.h>

#define uchar unsigned char

void Timer0\_Init(); // PWM波定时器

void Timer1\_Init(); //串口

void Timer2\_Init(); //超时检测定时

void make(); //数据处理

void resettime(); //超时检测

void send();//响应信息

sfr T2MOD = 0xC9; //定时器2寄存器地址

sbit LED1 = P0 ^ 0;

sbit LED2 = P0 ^ 1;

sbit LED3 = P0 ^ 2;

sbit LED4 = P0 ^ 3;

sbit LED5 = P0 ^ 6; //数据灯

sbit LED6 = P0 ^ 7; //计时灯

uchar level[4] = {0X00, 0X00, 0X00, 0X00}, mesg[6]; //占空比level+协议变量mesg

uchar code Fback[2][8] = {"SUCCED!","ERROR!"};

uchar time, a;

static bit uart\_flag, error\_flag; //使用位变量节省空间

static uchar count; //串口接收计数的静态变量初始值为0

//主函数

void main() {

Timer0\_Init();

Timer1\_Init();

Timer2\_Init();

while (1) {

send();

}

}

//定时器0中断固定脉宽PWM输出

void Timer0() interrupt 1 {

TR0 = 0; //赋初值时，关闭定时器

TH0 = 0XFF;

TL0 = 0Xb8;

TR0 = 1; //打开定时器

time++;

//占空比light

if (time < level[0]) {

LED1 = 0;

} else

LED1 = 1;

if (time < level[1]) {

LED2 = 0;

} else

LED2 = 1;

if (time < level[2]) {

LED3 = 0;

} else

LED3 = 1;

if (time < level[3]) {

LED4 = 0;

} else

LED4 = 1;

//清零time

if (time == 0xff) {

time = 0x00;

}

}

//串口中断，先接收8个字节至mesg数组保存，最后在主函数处理数据

void UART\_SER() interrupt 4 {

RI = 0; //手动清寄存器（数据接收完成自动置1）

mesg[count] = SBUF;

//同时判断count跟收到的数据

if (count == 0 && mesg[count] == 0xAA) {

count = 1;

} else if (count == 1) {

count = 2;

} else if (count == 2) {

count = 3;

} else if (count == 3) {

count = 4;

} else if (count == 4) {

count = 5;

} else if (count == 5 && mesg[count] == 0x55) //检验固定的帧尾

{

count = 0;

uart\_flag = 1; //串口接收成功标志，为1时在主程序中回复，然后清零

ES = 0; //关中断，回复完了再ES=1;

} else {

count = 0; //判断不满足条件就将计数值清零

error\_flag = 1; //头数据不通过

}

LED5 = ~LED5;

resettime(); //进入监控

}

//超时中断50ms\*400==20s

void T2\_Time() interrupt 5 {

static unsigned short run;

TF2 = 0; //手动标志位清零

LED6 = 0;

run++;

if (run == 400) {

if (uart\_flag != 1) {

error\_flag = 1; //时间到未完整传输，发送失败提示

}

count = 0; //清零

LED6 = 1; //关闭状态指示灯

LED5 = 1; //关闭数据传输灯

run = 0;

TR2 = 0; //只要计时到关闭监控，等待resettime()下一次开启

}

}

//定时器0初始化

void Timer0\_Init() {

TMOD |= 0x01; //定时器0工作方式1

TH0 = 0XFF;

TL0 = 0Xb8;

ET0 = 1; //开定时器0中断

TR0 = 1; //启动定时器0

EA = 1; //开总中断

}

//定时器1初始化

void Timer1\_Init() {

SCON = 0x50; // sm0sm1=01，串口工作方式1，10位=8为数据+1位开始+1位停止

TMOD |= 0x20;

TH1 = 0xfd; //波特率9600

TL1 = 0xfd;

TR1 = 1;

ES = 1; //串口使能

PS = 1; //串口中断优先级

EA = 1;

}

//定时器2初始化50ms

void Timer2\_Init() {

T2MOD = 0; //初始化模式寄存器

T2CON = 0; //初始化控制寄存器

TH2 = 0x4C; //设置定时初始值50ms

TL2 = 0x00;

RCAP2H = 0x4C; //设置定时重载值

RCAP2L = 0x00; //设置定时重载值

TR2 = 0; //定时器2默认关闭

ET2 = 1; //开定时器2中断

EA = 1;

}

void send(){

uchar i;

//成功反馈(发送+处理)

if (uart\_flag == 1) {

ES = 0;

for (i = 0; i < 8; i++) {

SBUF = Fback[0][i];

while (!TI) ;

TI = 0; //发送中断请求标志位

}

ES = 1;

uart\_flag = 0;

make(); //处理数据

}

//失败反馈（头尾检测不通过或者超时）

if (error\_flag == 1) {

ES = 0;

for (i = 0; i < 8; i++) {

SBUF = Fback[1][i];

while (!TI) ;

TI = 0; //发送中断请求标志位

}

ES = 1;

error\_flag = 0; //恢复状态

}

}

//数据处理（占空比赋值）

void make() {

for (a = 0; a < 4; a++) {

ET0 = 0; //关定时器0

level[a] = mesg[a + 1];

ET0 = 1; //开定时器0

}

}

//超时处理开启

void resettime() {

TR2 = 0; //关闭定时器2

if (count != 0) {

TR2 = 1; //打开定时器2

}

}