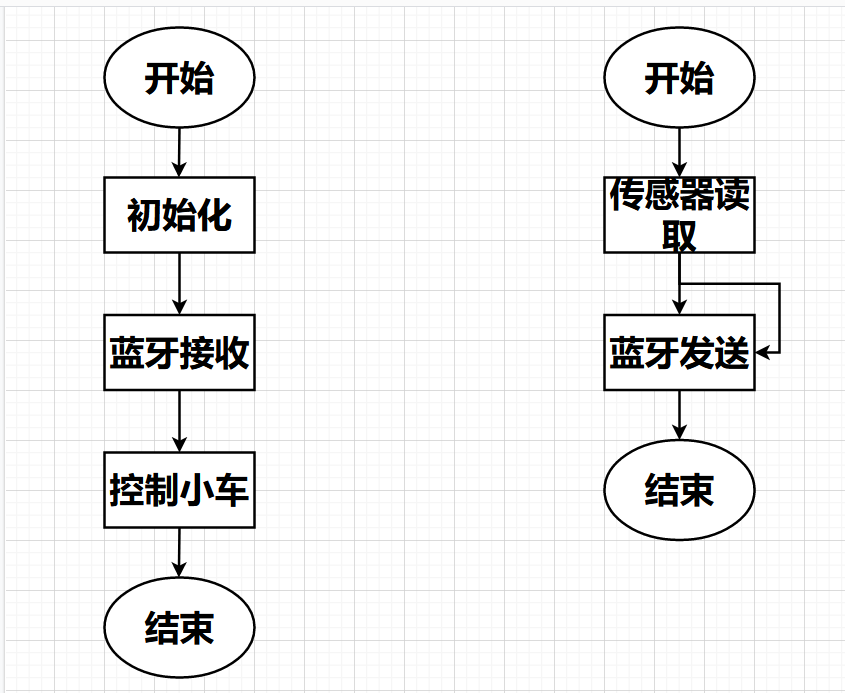
**3.2软件设计**

系统软件说明：

在进行微机控制系统设计时,除了系统硬件设计外,大量的工作就是如何根据每个生产对象的实际需要设计应用程序。因此,软件设计在微机控制系统设计中占重要地位。对于本系统,软件更为重要。  
 在单片机控制系统中,大体上可分为数据处理、过程控制两个基本类型。数据处理包括:声波识别、蓝牙数据传输等。过程控制程序主要是使单片机按一定的方法进行计算,然后再输出,以便控制生产。  
 为了完成上述任务,在进行软件设计时,通常把蹩个过程分成若千个部分,每一部分叫做一个模块。所谓“模块”,实质上就是所完成一定功能,相对独立的程序段,这种程序设计方法叫模块程序设计法。  
模块程序设计法的主要优点是;  
 单个模块比起一个完整的程序易编写及调试;模块可以共存,一个模块可以被多个任务在不同条件下调用:模块程序允许设计者分割任务和利用已有程序,为设计者提供方便。模块程序简单性为观察者带来方便。

**3.2.1总体软件框图**



**3.2.2发送模块主程序**

**//这是发送端的程序，请下载到板1上**

**#include <BLEDevice.h>**

**#include <BLE2902.h>**

**#include <time.h>**

**#include <String.h>**

**//传感器引脚可以再设置**

**#define minitor\_1 13**

**#define minitor\_2 34**

**#define minitor\_3 35**

**#define SERVICE\_UUID "4fafc201-1fb5-459e-8fcc-c5c9c331914b"**

**#define CHARACTERISTIC\_UUID "beb5483e-36e1-4688-b7f5-ea07361b26a8"**

**int C\_recv,A\_recv,B\_recv; //三个接收模块**

**float recv1,recv2,recv3;**

**char data\_sent[6];**

**float t;**

**int flag=0,clo=0,shifou=0;**

**int RetVal =0;**

**float tt[5];**

**int s=0;**

**float t1=30;**

**clock\_t start = 0 ;**

**clock\_t ending = 0 ;**

**clock\_t time1;**

**unsigned long previousMillis = 0;**

**class MyServerCallbacks: public BLEServerCallbacks {**

**void onConnect(BLEServer\* pServer) {};**

**void onDisconnect(BLEServer\* pServer) {**

**pServer->startAdvertising(); // 如果客户端断开连接了就重新开启Advertising广播，使客户端可以再次搜索到自己**

**}**

**};**

**BLECharacteristic\* pCharacteristic = NULL;**

**void time\_open()**

**{**

**start = clock();**

**}**

**void time\_close()**

**{**

**ending = clock();**

**}**

**void judge()**

**{**

**if(shifou==0)**

**{**

**tt[s]=t;**

**s++;**

**if(s==4)**

**{**

**s=0;**

**if(tt[0]<=t1 && tt[1]<=t1 && tt[2]<=t1 && tt[3]<=t1)**

**{**

**flag=1;**

**shifou=1;**

**clo=0;**

**}**

**for(int i=0;i<5;i++)**

**{**

**tt[i]=0;**

**}**

**}**

**}**

**if(shifou==1)**

**{**

**tt[s]=t;**

**s++;**

**if(s==4)**

**{**

**flag==0;**

**}**

**if(s==5)**

**{**

**s=0;**

**if(tt[0]<=t1 && tt[1]<=t1 && tt[2]<=t1 && tt[3]<=t1)**

**{**

**shifou=2;**

**}**

**for(int i=0;i<5;i++)**

**{**

**tt[i]=0;**

**}**

**}**

**}**

**RetVal+=shifou;**

**RetVal+=RetVal\*10+flag;**

**RetVal+=RetVal\*10+clo;**

**}**

**int t\_work(char flag)**

**{**

**read\_sensor();**

**//可能存在问题，上次喇叭的声音或杂音有干扰，需要在ABC三个接收模块都清零后进行下一步**

**if(flag == 0)//只记录BC的信号时间差**

**{**

**if( A\_recv == 0 && B\_recv == 1)**

**{**

**time\_open();**

**q: read\_sensor();**

**if( A\_recv == 1 && B\_recv == 1)**

**{**

**time\_close();**

**t = ending - start;**

**clo = 1; //接近B端**

**return t;**

**}**

**delay(2);**

**goto q;**

**}**

**if( A\_recv == 1 && B\_recv == 0)**

**{**

**time\_open();**

**w: read\_sensor();**

**if( A\_recv == 1 && B\_recv == 1)**

**{**

**time\_close();**

**t = ending - start;**

**clo = 0; //接近A端**

**return t;**

**}**

**delay(2);**

**goto w;**

**}**

**}**

**else//只记录AB的信号时间差**

**{**

**if( C\_recv == 0 && A\_recv == 1)**

**{**

**time\_open();**

**e: read\_sensor();**

**if( A\_recv == 1 && C\_recv == 1)**

**{**

**time\_close();**

**t = ending - start;**

**clo = 0; //接近A端**

**return t;**

**}**

**delay(2);**

**goto e;**

**}**

**if( C\_recv == 1 && A\_recv == 0)**

**{**

**time\_open();**

**r: read\_sensor();**

**if( A\_recv == 1 && C\_recv == 1)**

**{**

**time\_close();**

**t = ending - start;**

**clo = 1; //接近C端**

**return t;**

**}**

**delay(2);**

**goto r;**

**}**

**}**

**}**

**void read\_sensor()**

**{**

**//读取传感器数值**

**C\_recv = digitalRead(minitor\_1);**

**A\_recv = digitalRead(minitor\_2);**

**B\_recv = digitalRead(minitor\_3);**

**}**

**void setup() {**

**Serial.begin(115200);**

**Serial.println();**

**pinMode(minitor\_1,INPUT);**

**pinMode(minitor\_2,INPUT);**

**pinMode(minitor\_3,INPUT);**

**BLEDevice::init("ESP32-BLE");**

**BLEServer \*pServer = BLEDevice::createServer(); // 创建服务器**

**pServer->setCallbacks(new MyServerCallbacks());**

**BLEService \*pService = pServer->createService(SERVICE\_UUID); // 创建服务**

**pCharacteristic = pService->createCharacteristic(**

**CHARACTERISTIC\_UUID,**

**BLECharacteristic::PROPERTY\_READ |**

**BLECharacteristic::PROPERTY\_WRITE |**

**BLECharacteristic::PROPERTY\_NOTIFY |**

**BLECharacteristic::PROPERTY\_INDICATE**

**);**

**pCharacteristic->setValue("Hello World! ");**

**pCharacteristic->addDescriptor(new BLE2902());**

**pService->start();**

**BLEAdvertising \*pAdvertising = BLEDevice::getAdvertising();**

**pAdvertising->addServiceUUID(SERVICE\_UUID); // 广播服务的UUID**

**BLEDevice::startAdvertising();**

**}**

**void loop()**

**{**

**judge();**

**t\_work(flag);**

**pCharacteristic->setValue(RetVal);**

**Serial.println(RetVal);**

**pCharacteristic->notify(); // 每隔interval时间主动推送一次数据**

**RetVal=0;**

**delay(20);**

**}**

**3.2.3 接收端主程序**

**#include<FreeRTOS.h> //多线程库**

**/\* 创建任务一和任务二的句柄，并初始化 \*/**

**TaskHandle\_t TASK\_HandleOne = NULL;**

**#include <time.h>**

**#define sound 5 //喇叭**

**#define LED 2 //LED**

**#define left\_motor\_1 33 //IN1**

**#define left\_motor\_2 25 //IN2**

**#define right\_motor\_1 26 //IN3**

**#define right\_motor\_2 27 //IN4**

**#define ENA 32 //ENA**

**#define ENB 14 //ENB**

**#include <BLEDevice.h>**

**#define SERVICE\_UUID "4fafc201-1fb5-459e-8fcc-c5c9c331914b"**

**#define CHARACTERISTIC\_UUID "beb5483e-36e1-4688-b7f5-ea07361b26a8"**

**boolean doSacn = true;**

**boolean doConnect = false;**

**boolean connected = false;**

**BLEAdvertisedDevice\* pServer;**

**BLERemoteCharacteristic\* pRemoteCharacteristic;**

**int buf[3];**

**int bbb[3];**

**// 搜索到设备时回调功能**

**class MyAdvertisedDeviceCallbacks: public BLEAdvertisedDeviceCallbacks {**

**void onResult(BLEAdvertisedDevice advertisedDevice) {**

**// if (advertisedDevice.haveServiceUUID() && advertisedDevice.isAdvertisingService(BLEUUID("4fafc201-1fb5-459e-8fcc-c5c9c331914b"))) {**

**if (advertisedDevice.haveName() && (advertisedDevice.getName()=="ESP32-BLE")) {**

**advertisedDevice.getScan()->stop(); // 停止当前扫描**

**pServer = new BLEAdvertisedDevice(advertisedDevice); // 暂存设备**

**doSacn = false;**

**doConnect = true;**

**Serial.println("发现想要连接的设备");**

**}**

**}**

**};**

**// 客户端与服务器连接与断开回调功能**

**class MyClientCallback : public BLEClientCallbacks {**

**void onConnect(BLEClient\* pclient) {}**

**void onDisconnect(BLEClient\* pclient) {**

**doSacn = true;**

**connected = false;**

**Serial.println("失去与设备的连接");**

**}**

**};**

**// 收到服务推送的数据时的回调函数**

**void NotifyCallback(BLERemoteCharacteristic\* pBLERemoteCharacteristic, uint8\_t\* pData, size\_t length, bool isNotify) {**

**//char buf[length + 1];**

**for (size\_t i = 0; i < length; i++) {**

**bbb[i] = pData[i];**

**}**

**buf[length] = 0;**

**buf[0]=bbb[0]%10;**

**buf[1]=(bbb[0]%100-buf[0])/10;**

**buf[2]=bbb[0]/100;**

**}**

**// 用来连接设备获取其中的服务与特征**

**bool ConnectToServer(void) {**

**BLEClient\* pClient = BLEDevice::createClient(); // 创建客户端**

**pClient->setClientCallbacks(new MyClientCallback()); // 添加客户端与服务器连接与断开回调功能**

**if (!pClient->connect(pServer)) { // 尝试连接设备**

**return false;**

**}**

**Serial.println("连接设备成功");**

**BLERemoteService\* pRemoteService = pClient->getService(SERVICE\_UUID); // 尝试获取设备中的服务**

**if (pRemoteService == nullptr) {**

**Serial.println("获取服务失败");**

**pClient->disconnect();**

**return false;**

**}**

**Serial.println("获取服务成功");**

**pRemoteCharacteristic = pRemoteService->getCharacteristic(CHARACTERISTIC\_UUID); // 尝试获取服务中的特征**

**if (pRemoteCharacteristic == nullptr) {**

**Serial.println("获取特性失败");**

**pClient->disconnect();**

**return false;**

**}**

**Serial.println("获取特征成功");**

**if(pRemoteCharacteristic->canRead()) { // 如果特征值可以读取则读取数据**

**Serial.printf("该特征值可以读取并且当前值为: %s\r\n", pRemoteCharacteristic->readValue().c\_str());**

**}**

**if(pRemoteCharacteristic->canNotify()) { // 如果特征值启用了推送则添加推送接收处理**

**pRemoteCharacteristic->registerForNotify(NotifyCallback);**

**}**

**}**

**void function(char cho)**

**{**

**switch(cho)**

**{**

**case 'S':**

**ledcWrite(0,0);**

**ledcWrite(1,0);**

**for(int i=0;i<=15;i++)**

**{**

**digitalWrite(LED,HIGH); //停车且伴有明显的声光指示**

**ledcWriteTone(6,1800);**

**delay(500);**

**digitalWrite(LED,LOW);**

**delay(500);**

**}**

**ledcWriteTone(6,0);**

**break;**

**case 'L': //左转**

**digitalWrite(left\_motor\_1,HIGH);**

**digitalWrite(left\_motor\_2,LOW);**

**ledcWrite(0,1750);**

**ledcWrite(1,1750);**

**delay(550);**

**digitalWrite(left\_motor\_1,LOW);**

**digitalWrite(left\_motor\_2,HIGH);**

**break;**

**case 'A':**

**digitalWrite(left\_motor\_1,HIGH);**

**digitalWrite(left\_motor\_2,LOW);**

**ledcWrite(0,1750);**

**ledcWrite(1,1750);**

**delay(880);**

**digitalWrite(left\_motor\_1,LOW);**

**digitalWrite(left\_motor\_2,HIGH);**

**break;**

**/\***

**case 'B':**

**digitalWrite(left\_motor\_1,HIGH);**

**digitalWrite(left\_motor\_2,LOW);**

**digitalWrite(right\_motor\_1,HIGH);**

**digitalWrite(right\_motor\_2,LOW);**

**ledcWrite(0,1700);**

**ledcWrite(1,1700);**

**delay(80);**

**digitalWrite(left\_motor\_1,LOW);**

**digitalWrite(left\_motor\_2,HIGH);**

**digitalWrite(right\_motor\_1,LOW);**

**digitalWrite(right\_motor\_2,HIGH);**

**ledcWrite(0,0);**

**ledcWrite(1,0);**

**break;**

**\*/**

**}**

**}**

**void judge() // buf[0]==clo; buf[1]==flag; buf[2]==shifou**

**{**

**/\*if(buf[2]==0)**

**{**

**if(buf[1]==0)**

**{**

**do**

**{**

**function('B');**

**}while(buf[0]==1)**

**}**

**}\*/**

**if(buf[2]==1)**

**{**

**function('S');**

**delay(1000);**

**function('L');**

**if(buf[0]==0 && buf[1]==1)**

**{**

**ledcWrite(0,1720);**

**ledcWrite(1,1700);**

**}**

**/\***

**if(buf[1]==1)**

**{**

**do**

**{**

**function('B');**

**}while(buf[0]==1);**

**}**

**\*/**

**if(buf[1]==0)**

**{**

**if(buf[0]==0)**

**{**

**ledcWrite(1,1760);**

**delay(60);**

**ledcWrite(1,1700);**

**}**

**else if(buf[0]==1)**

**{**

**ledcWrite(0,1770);**

**delay(60);**

**ledcWrite(0,1700);**

**}**

**}**

**if(buf[2]==2)**

**{**

**function('S');**

**function('S');**

**function('S');**

**function('S');**

**function('S');**

**function('S');**

**}**

**}**

**}**

**void setup() {**

**Serial.begin(115200);**

**Serial.println();**

**xTaskCreate(**

**TASK\_ONE, // 任务函数**

**"TaskOne", // 任务名**

**8\*1024, // 任务栈大小，根据需要自行设置**

**NULL, // 参数，入参为空**

**1, // 优先级**

**&TASK\_HandleOne); // 任务句柄**

**// 初始化串口**

**ledcSetup(0, 5000, 10); //通道0， 5KHz，10位解析度**

**ledcSetup(1, 5000, 10); //通道1， 5KHz，10位解析度**

**pinMode(sound,OUTPUT);**

**ledcAttachPin(sound,6);**

**ledcSetup(6,1200,8); //喇叭定义**

**pinMode(LED,OUTPUT); //LED定义**

**pinMode(left\_motor\_1,OUTPUT);**

**pinMode(right\_motor\_1,OUTPUT);**

**pinMode(left\_motor\_2,OUTPUT);**

**pinMode(right\_motor\_2,OUTPUT);**

**pinMode(ENA,OUTPUT);**

**pinMode(ENB,OUTPUT);**

**digitalWrite(left\_motor\_1,LOW);**

**digitalWrite(left\_motor\_2,HIGH);**

**digitalWrite(right\_motor\_1,LOW);**

**digitalWrite(right\_motor\_2,HIGH);**

**ledcAttachPin(ENA, 0);**

**ledcAttachPin(ENB, 1);**

**BLEDevice::init("");**

**BLEScan\* pBLEScan = BLEDevice::getScan();**

**pBLEScan->setAdvertisedDeviceCallbacks(new MyAdvertisedDeviceCallbacks());**

**pBLEScan->setActiveScan(true);**

**pBLEScan->setInterval(100);**

**pBLEScan->setWindow(80);**

**}**

**void loop() {**

**/\*// 如果需要扫描则进行扫描**

**if (doSacn) {**

**Serial.println("开始搜索设备");**

**BLEDevice::getScan()->clearResults();**

**BLEDevice::getScan()->start(0); // 持续搜索设备**

**}**

**// 如果找到设备就尝试连接设备**

**if (doConnect) {**

**if (ConnectToServer()) {**

**connected = true;**

**}**

**else {**

**doSacn = true;**

**}**

**doConnect = false;**

**}\*/**

**delay(400);**

**ledcWrite(0,1800);**

**ledcWrite(1,1800);**

**delay(1200);**

**function('S');**

**function('L');**

**ledcWrite(0,1800);**

**ledcWrite(1,1800);**

**delay(950);**

**function('S');**

**function('S');**

**//judge();**

**}**

**/\* 任务一的函数体,由于入参即为NULL，因此函数体的入参需为void \*参数，否则报错 \*/**

**void TASK\_ONE(void \*param )**

**{**

**// 打印来自任务1的信息**

**for( int i = 0;i<10;i++ )**

**{**

**Serial.println("Hello from TaskOne~");**

**delay(10);**

**}**

**// 打印10次后通过任务一的句柄退出任务，并释放资源**

**Serial.println("Ending TaskOne!");**

**for(int i=0;i<=10000;i++)**

**{**

**ledcWriteTone(6,500); //喇叭间断响**

**delay(50);**

**ledcWriteTone(6,0);**

**delay(50);**

**}**

**vTaskDelete( TASK\_HandleOne ); //退出任务**

**}**