

**计算机科学与工程学院**

**“硬件课程设计2”报告书**

**题目： 远程操控的电机控制器设计**

**小组长： 刘婷珺**

**小组成员：邱丽丽、刘祥德**

**小组成绩：优**

**日期： 2019 年 1 月 12 日**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 小组成员学号 | | 姓名 | 承担的任务 | 成绩 |
| 915106840304 | | 刘婷珺 | 老师要求的功能以及扩展功能1、2 | 优 |
| 915106840327 | | 刘祥德 | 电机传输协议层以及扩展功能3、4 | 优 |
| 915106840307 | | 邱丽丽 | 老师要求的功能以及扩展功能1、2 | 良 |
| 题目的要求 | 1. 老师的要求 2. 可以通过PC机远程启动和停止电机旋转； 3. 基于S3C2440来设计控制器，选用Timer部件来实现直流电机控制； 4. 可以通过PC机远程对电机进行调速； 5. PC机接收控制器返回的占空比参数； 6. 在控制器的LED显示器上显示控制占空比的参数。 7. PC机按照协议规定的通信格式与控制器进行通信。 8. 扩展的功能 9. 利用键盘A、D键控制电机的转速：按A键表示加速，D键表示减速。 10. 利用时钟信号控制电机启动：当前时间过10秒钟后启动。 11. 回传校验：控制器每次收到PC机发送的指令后，如果校验成功（失败），会回传成功（失败）信号，PC机收到回传信息后，按同样的方式进行校验，并输出校验结果。 12. 测试功能：编写测试脚本，完成系统各功能的测试。 | | | |
| 总体设计 | **一、系统的组成及硬件环境**  构造的一个交叉编译环境，采用宿主机—目标机的的开发架构，如图1.1。宿主机为PC机，其上运行开发工具软件。主机程序采用VB语言编写，并运行在PC机上，用于对控制器发送命令。同时使用ADS1.2来编写嵌入式系统程序，然后将程序通过超级终端由PC机下载到实验箱上运行，实验箱是基于S3C2440的，用来作为控制器的硬件平台，也就是目标机。    图1.1 开发架构  **二、描述系统的软件总体流程**   1. 总体流程   首先由启动引导程序引导main函数执行， main函数中首先初始化了串口通信功能，并向PC机发送了Timer提示信息。随后main函数进入一个while死循环，并在循环的最开始调用check()函数，该函数从串口的接收缓冲区中读取六次数据，将数据存入数组中，并对数据进行校验。如果校验成功，返回true，继续后续的操作；如果校验失败，则返回false，在控制器LED屏上显示“11”，并向pc机发送失败信息，然后进入下一次循环，等待主机程序发来的新的操作指令。  校验成功后，main函数对接收到的数据中的命令码进行判断：  如果命令码是0x31，说明主机发送的是启动电机的指令，故调用TimerInit()函数，对Timer部件进行初始化以实现直流电机的转动。操作成功后，向pc机发送该条指令成功信息。  如果命令码是0x32，说明主机发送的是停止电机转动的指令，仍按照之前的方式，调用TimerInit()函数，并将Timer部件中TCMPB0寄存器的值设置为MOTOR\_MID，从而停止电机的转动。操作成功后，向pc机发送该条指令成功信息。  如果命令码是0x33，说明主机发送的是电机调速指令。主机程序规定：如果要调速，必须先按下调速按钮，发出调速指令后，再通过键盘或者进度条的方式进行调速。因此，在判断完调速指令后，main函数进入到一个新的循环中，再次调用check函数，读取并校验接收缓冲区中的数据。如果此时的命令码等于0x34，则表示主机通过A键对电机进行加速；如果命令码是0x35，则表示通过D键进行减速；如果命令码仍是0x33，则表示通过进度条调速。调速时，通过改变Timer部件比较缓冲寄存器TCMPB0的值，来减小或者增大电压的偏离程度，从而减缓或者加快电机转动。每次操作成功后，均向pc机发送该条指令成功信息。  如果命令码是0x36，说明主机发送的是过10秒启动电机指令，并且发来了用于设置RTC相关时间寄存器的时间值。此时，调用RTC\_init()函数，启动RTC相关寄存器，使用主机发来的时间值设置相关寄存器的初值，并启动中断，设置中断服务程序入口为RTC\_TICK\_ISR函数。在RTC\_TICK\_ISR函数中，返回RTC当前时间给主机，并通过比较RTC当前时间是否等于初始时间加上10秒，来决定是否启动电机。操作成功后，向pc机发送该条指令成功信息。  图1.2即展示了系统流程图。    图1.2 系统流程图   1. 通信协议  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | 起始码 | 命令码 | 参数1 | 参数2 | 校验码  （高位） | 校验码  （低位） |   上图所示的即为PC机和控制器间的通信协议，每一部分均为一个字节，并按照ASCII码方式进行编码。其中，起始码等于0x30，检验码计算方式遵循累加和校验，并保留一个字节。 | | | |
| 详细设计 | 1. **基础功能** 2. **通信驱动** 3. **通信格式**   约定PC机的客户端为发送方，电机的服务端为接收方，双方都需要维护一个通信驱动模块。  PC机的客户端发包格式如下：   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | 起始符 | 命令码 | 参数1 | 参数2 | 校验码  （高位） | 校验码  （低位） |   **PC机发包代码为：**  Public Function sendPkt(b0 As Byte, b1 As Byte, b2 As Byte, b3 As Byte, b4 As Byte, b5 As Byte)  Dim b(0) As Byte  b(0) = b0  Form1.MSComm1.Output = b  b(0) = b1  Form1.MSComm1.Output = b  b(0) = b2  Form1.MSComm1.Output = b  b(0) = b3  Form1.MSComm1.Output = b  b(0) = b4  Form1.MSComm1.Output = b  b(0) = b5  Form1.MSComm1.Output = b  'Text2 = Hex(b0) & " " & Hex(b1) & " " & Hex(b2) & " " & Hex(b3) & " " & Hex(b4) & " " & Hex(b5)  End Function  **电机接收代码为：**  for(j=0;j<6;j++){  err=Uart\_Getchn(c1,0); //从串口接收命令  delay(10,10);  recarr[j]=c1[0];  if(j<=3){  checksum+=recarr[j];  }  }  电机的服务端发包格式如下：   |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | 起始符 | 命令码 | 参数1 | 参数2 | 校验码  （高位） | 校验码  （低位） | 终止符 |   电机方的包多出一个终止符，包长为7。  **电机的发包代码为：**  //通过串口0向PC机发包  int sendPkt(U8 pkt[]){  int i;  for(i=0;i<PKT\_LEN;i++){  Uart\_SendByten(0,pkt[i]); //每次发送一个U8字符(按一个字节的格式)  //delay(10,10);  }  return 1;  }  **PC机的接收代码为：**  //当且仅当收到7个字符时，该事件会被触发  Private Sub MSComm1\_OnComm()  Dim RevByte As Variant  Dim cnt As Integer  cnt = Form1.MSComm1.InBufferCount  Dim dt As Date  dt = Now  Dim strDate As String  strDate = Format(dt, "yyyy年MM月dd日mm分ss秒")  Text3 = ""  If cnt > 0 Then  RevByte = Form1.MSComm1.Input //一次性收取7个字符数  If RevByte(0) = &H30 And RevByte(6) = &H60 Then '判断首位和末位是否为协议要求的符号'  Dim sum As Integer  Dim pktSum As Integer  sum = 0  sum = (RevByte(0) Mod 256) + (RevByte(1) Mod 256) + (RevByte(2) Mod 256) + (RevByte(3) Mod 256)  pktSum = (RevByte(4) - &H30) \* 10 + (RevByte(5) - &H30)  If sum = pktSum Then '检验该ACK，确保电机消息的可信度'  Text3 = "收到的信息正确"  If RevByte(1) = &H1 Then  If RevByte(2) = &H11 Then  Text2.Text = Text2.Text & "电机启动成功"  End If  If RevByte(2) = &H21 Then  Text2.Text = Text2.Text & "电机停止成功"  End If  If RevByte(2) = &H31 Then  Text2 = "电机加速成功" & "当前占空比为:" & RevByte(3)  Text1 = RevByte(3)  End If  If RevByte(2) = &H41 Then  Text2 = Text2.Text & "电机减速成功" & "当前占空比为:" & RevByte(3)  Text1 = RevByte(3)  End If  If RevByte(2) = &H51 Then  Text2 = Text2.Text & "电机进入调速模式" & "当前占空比为:" & RevByte(3)  Text1 = RevByte(3)  End If  If RevByte(2) = &H61 Then  Text2 = Text2.Text & "电机进入定时启动模式成功"  End If  'Text2 = "电机方校验成功"  ' If RevByte(2) = &O1 Then '显示占空比'  ' Text1 = RevByte(3)  ' End If  End If  If RevByte(1) = &H2 Then  If RevByte(2) = &H12 Then  Text2 = Text2.Text & "电机启动失败"  End If  If RevByte(2) = &H22 Then  Text2 = Text2.Text & "电机停止失败"  End If  If RevByte(2) = &H32 Then  Text2 = Text2.Text & "电机加速失败"  End If  If RevByte(2) = &H42 Then  Text2 = Text2.Text & "电机减速失败"  End If  If RevByte(2) = &H52 Then  Text2 = Text2.Text & "电机进入调速模式失败"  End If  If RevByte(2) = &H62 Then  Text2 = Text2.Text & "电机进入定时启动模式失败"  End If  End If  If RevByte(1) = &H3 Then  Text2 = Text2.Text & "主机发送了未知的指令"  End If  If RevByte(1) = &H4 Then  'Text2 = "收到RTC回馈"  Text4 = RevByte(3)  Text5 = RevByte(2)  End If    Else  Text3 = "收到的信息损坏"  End If  Else '如果起始符不是双方要求的起始符,则认为该帧不是合法的'  Text2 = "起始符不正确"  Text3 = "未知的帧"  Text2 = RevByte(0) & " " & RevByte(1) & " " & RevByte(2) & " " & RevByte(3) & " " & RevByte(4) & " " & RevByte(5) & " " & RevByte(6)  End If  Text2.Text = Text2.Text & " " & strDate & vbCrLf  End If  End Sub   1. **包构造**   //Packet构造函数,Packet的格式从左往右分别是：  起始符、命令码、参数1、参数2、校验码高位、检验码低位、终止符  int makePacket(U8 pkt[],U8 cmd, U8 param1, U8 param2){  pkt[0] = START\_FLAG; //写入起始符号  pkt[1] = cmd; //写入命令码  pkt[2] = param1; //写入参数1  pkt[3] = param2; //  return checkSum(pkt,cmd,param1,param2);  }  //校验和写入函数,就检验码写入到Packet当中  int checkSum(U8 pkt[],U8 cmd ,U8 param1,U8 param2){  //避免加法溢出，先对U8分别取模  U8 sum = MOD8(START\_FLAG) + MOD8(cmd) + MOD8(param1) + MOD8(param2);  //十进制校验和的高位ASCII码  U8 highCheckSum = Asc(sum / 10);  //十进制检验和的低位ASCII码  U8 lowCheckSum = Asc(sum % 10);  pkt[4] = highCheckSum; //写入校验码高位  pkt[5] = lowCheckSum; //写入校验码低位  pkt[6] = END\_FLAG; //写入终止符  return sum;  }    图2.1电机传输Packet构造流程  上图展示了电机端通信包的构造过程，椭圆里面的内容表示构造函数的几个输入，长方形表示通信包构造的子过程。  循环等待主机程序命令：  while(1)  {  int flag=check();  if(flag==1){  key\_flag=recarr[1];  //key\_flag=0x01;  switch (key\_flag){}  else{  //h=(checksum&0xff00)/256;  lednum1con=num[1];  // h=(checksum&0x00ff);  lednum2con=num[1];  sendAck(FAILED\_CODE,0);  }  }   1. **通过PC机远程启动和停止电机旋转**   case INIT\_CMD: //启动电机  TimerInit(MOTOR\_MID - MOTOR\_CONT/10); //执行启动命令,顺时针转  lednum1con=num[1]; //显示高位数  lednum2con=num[0]; //显示低位数  key\_flag=0x00;  sendAck(SUCCESS\_CODE,0);  break;  case STOP\_CMD: //停止电机  TimerInit(MOTOR\_MID); //执行停止命令  lednum1con=num[18]; //显示高位数  lednum2con=num[18]; //显示低位数  key\_flag=0x00;  sendAck(SUCCESS\_CODE,0);  break;   1. **通过PC机远程对电机进行调速**   case SPEED\_CMD: // 调速模式  lednum1con=num[2];  lednum2con=num[2];  delay(10,10);  sendAck(SUCCESS\_CODE,0);  while(1)  {  flag=check();//从串口接收调节占空比的参数  if(flag==1){  if (recarr[1]==STOP\_CMD)  {  TimerInit(MOTOR\_MID);  lednum1con=num[18]; //显示高位数  lednum2con=num[18]; //显示低位数  sendAck(SUCCESS\_CODE,0);  break;  }  // 电机加速转动  if(recarr[1]==ADD\_CMD){  if(data2>4){  data2=data2-2;  //显示占空比  k=data2/10;  lednum1con=num[k];  k=data2%10;  lednum2con=num[k];  TimerInit(MOTOR\_MID-MOTOR\_CONT/data2);  sendAck(SUCCESS\_CODE,data2);  }  else{  k=data2/10;  lednum1con=num[k];  k=data2%10;  lednum2con=num[k];  //发送加速失败的响应  TimerInit(MOTOR\_MID-MOTOR\_CONT/data2);  //发送加速失败的响应  sendAck(FAILED\_CODE,data2);  }  }  // 电机减速转动  else if(recarr[1]==DOWN\_CMD){  if(data2<30){  data2=data2+2;  //显示占空比  k=data2/10;  lednum1con=num[k];  k=data2%10;  lednum2con=num[k];  TimerInit(MOTOR\_MID-MOTOR\_CONT/data2);  sendAck(SUCCESS\_CODE,data2);  }  else{//低于30不再减速  k=data2/10;  lednum1con=num[k];  k=data2%10;  lednum2con=num[k];  TimerInit(MOTOR\_MID-MOTOR\_CONT/data2);  //发送减速失败的响应  sendAck(FAILED\_CODE,data2);  }    }  else if(recarr[1]==SPEED\_CMD){//自定义速度  data1=recarr[2];  //参数显示在2位LED显示器上  i=(data1 & 0xf0)/16;  lednum1con=num[i]; //显示高位数  i=(data1 & 0x0f);  lednum2con=num[i]; //显示低位数  data1=((data1 & 0xf0)/16)\*10 + (data1 & 0x0f);  data2=data1;  TimerInit(MOTOR\_MID - MOTOR\_CONT/data1);  sendAck(SUCCESS\_CODE,data2);  }  }  else{  lednum1con=num[3];  lednum2con=num[3];  }  }  key\_flag=0x00;  break;   1. **接收控制器返回占空比参数**   // 电机加速转动  if(recarr[1]==ADD\_CMD){  if(data2>4){  data2=data2-2;  //显示占空比  k=data2/10;  lednum1con=num[k];  k=data2%10;  lednum2con=num[k];  TimerInit(MOTOR\_MID - MOTOR\_CONT/data2);  sendAck(SUCCESS\_CODE,data2);//将占空比作为额外参数嵌入到ACK中，发送到PC  }  else{  k=data2/10;  lednum1con=num[k];  k=data2%10;  lednum2con=num[k];  TimerInit(MOTOR\_MID - MOTOR\_CONT/data2);//发送加速失败  sendAck(FAILED\_CODE,data2);//发送加速失败的响应  }  }  当电机收到了调速指令时，电机会改变比较缓冲寄存器TCMPB0的值，来增加增大/减小电压的偏离程度，从而改变自身的转动状态。随机发送一个嵌入了占空比的ACK到前台客户端，表明已经完成调速，前台可以根据该ACK读取到后台回传的占空比。   1. **LED实时显示占空比**   num[19]={0xc0,0xf9,0xa4,0xb0,0x99,0x92,0x82,0xf8,0x80,0x90,0x88,0x83,0xc6,0xa1,0x86,0x8e,0xbf,0x7f,0xff};  //显示占空比  k=data2/10; //Data2记录着电机当前的占空比参数  lednum1con=num[k]; // lednum1con是八段LED的寄存器  k=data2%10;  lednum2con=num[k];  TimerInit(MOTOR\_MID - MOTOR\_CONT/data2);  sendAck(SUCCESS\_CODE,data2);  因为num数组记录了数字与LED编码的关系，计算占空比的高位和低位，取到num值分别赋给LED寄存器，LED就会显示当前的占空比。  二**、 扩展功能**   1. **利用键盘A、D键控制电机的转速：按A键表示加速，D键表示减速。**   首先要进入调速模式，才能利用A、D键控速；利用鼠标拖动调速条也能调速。每次调速基于当前的速度增减。PC机在调速模式下，发送带占空比参数的指令给系统，系统先检查校验和，然后提取占空比对电机调速。  case SPEED\_CMD: // 调速模式  lednum1con=num[2];  lednum2con=num[2];  delay(10,10);  sendAck(SUCCESS\_CODE,0);  while(1)  {  flag=check();//从串口接收调节占空比的参数  if(flag==1){  if (recarr[1]==STOP\_CMD)  {  TimerInit(MOTOR\_MID);  lednum1con=num[18]; //显示高位数 lednum2con=num[18]; //显示低位数  sendAck(SUCCESS\_CODE,0);  break;  }  // 电机加速转动  if(recarr[1]==ADD\_CMD){  if(data2>4){  data2=data2-2;  //显示占空比  k=data2/10;  lednum1con=num[k];  k=data2%10;  lednum2con=num[k];  TimerInit(MOTOR\_MID-MOTOR\_CONT/data2);  sendAck(SUCCESS\_CODE,data2);  }  else{  k=data2/10;  lednum1con=num[k];  k=data2%10;  lednum2con=num[k];  //发送加速失败的响应  TimerInit(MOTOR\_MID - MOTOR\_CONT/data2);  //发送加速失败的响应 sendAck(FAILED\_CODE,data2);  }  }  // 电机减速转动  else if(recarr[1]==DOWN\_CMD){  if(data2<30){  data2=data2+2;  //显示占空比  k=data2/10;  lednum1con=num[k];  k=data2%10;  lednum2con=num[k];  TimerInit(MOTOR\_MID - MOTOR\_CONT/data2);  sendAck(SUCCESS\_CODE,data2);  }  else{//低于30不再减速  k=data2/10;  lednum1con=num[k];  k=data2%10;  lednum2con=num[k];  TimerInit(MOTOR\_MID - MOTOR\_CONT/data2);  sendAck(FAILED\_CODE,data2);//发送减速失败的响应  }  }  else if(recarr[1]==SPEED\_CMD){//自定义速度  data1=recarr[2];  //参数显示在2位LED显示器上  i=(data1 & 0xf0)/16;  lednum1con=num[i]; //显示高位数  i=(data1 & 0x0f);  lednum2con=num[i]; //显示低位数  data1=((data1 & 0xf0)/16)\*10 + (data1 & 0x0f);  data2=data1;  TimerInit(MOTOR\_MID - MOTOR\_CONT/data1);  sendAck(SUCCESS\_CODE,data2);  }  }  else{  lednum1con=num[3];  lednum2con=num[3];  }  }  key\_flag=0x00;  break;   1. **利用时钟信号控制电机启动：当前时间过10秒后启动。**   PC机点击10秒启动按钮，将当前时间的分、秒作为参数传送指令。前台系统接收后，首先检查校验和，将flag10置为1，然后启动RTC部件使之计数，将flag10置为0，保证电机不会每隔10秒启动一次。RTC的分、秒也实时传送给PC机并显示。若RTC的时间等于PC机传送的时间加上10秒，则电机启动。  前台系统接受指令：  case TIMER\_CMD:  lednum1con=num[3];  lednum2con=num[6];  flag10=1;  RTC\_init();  key\_flag=0x00;  sendAck(SUCCESS\_CODE,0);  break;  RTC部件启动并及时：  void RTC\_init(void)  {  U8 Y,MO,D,W,H,MI,S;  year=19;  month=1;  day=10;  wkd=4;  hour=15;  //minute=4;  //second=50;  //初始化RTC相关寄存器  rRTCCON = (rRTCCON|0x01);  rRTCALM = 0x00;  rRTCRST = 0x00;  rTICINT = 0xff;  rRTCCON = (rRTCCON&0xfe);    Y=(U8)(year/10\*16+year%10);  MO=(U8)(month/10\*16+month%10);  D=(U8)(day/10\*16+day%10);  W=(U8)(wkd);  H=(U8)(hour/10\*16+hour%10);  //MI=(U8)(minute/10\*16+minute%10);  //S=(U8)(second/10\*16+second%10);    rRTCCON = (rRTCCON|0x01);    rBCDYEAR=Y;  rBCDMON=MO;  rBCDDAY=W;  rBCDDATE=D;  rBCDHOUR=H;  rBCDMIN=recarr[2];  rBCDSEC=recarr[3];  rRTCCON = (rRTCCON&0xfe);    minu=(recarr[2]&0x70)/16\*10+(recarr[2]&0x0f);  sec=(recarr[3]&0x70)/16\*10+(recarr[3]&0x0f);    INTS\_OFF();  SetISR\_Interrupt(IRQ\_NUM, RTC\_TICK\_ISR, NULL);  //设置中断服务程序入口  INTS\_ON(); //初始化CPSR寄存器的中断开放位  }  static void RTC\_TICK\_ISR(int vector, void\* data)  {  U8 Y,S,MO,D,W,H,MI,sen,mm;  int fen,miao,kk;    rRTCCON = (U8)(rRTCCON |0x01);  S = rBCDSEC;  MI=rBCDMIN;  rRTCCON = (U8)(rRTCCON &0xfe);    sen=((S & 0xf0)/16)\*10 + (S & 0x0f);  mm=((MI & 0xf0)/16)\*10 + (MI & 0x0f);    sendTimer(mm,sen);    //judge part  fen=(MI&0x70)/16\*10+(MI&0x0f);  miao=(S&0x70)/16\*10+(S&0x0f);    if(flag10==1){  if(sec<50){  if(miao==sec+10){  TimerInit(MOTOR\_MID - MOTOR\_CONT/10);  lednum1con=num[1];  lednum2con=num[0];  flag10=0;  return;  }  }  else{  if((fen==minu+1)&&(miao==sec+10-60)){  TimerInit(MOTOR\_MID - MOTOR\_CONT/10);  lednum1con=num[1];  lednum2con=num[0];  flag10=0;  return;  }  }  }  }   1. **状态回传**   /////////////\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 命令码 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\* //////////////  #define INIT\_CMD 0x31  #define STOP\_CMD 0x32  #define SPEED\_CMD 0x33  #define ADD\_CMD 0x34  #define DOWN\_CMD 0x35  #define TIMER\_CMD 0x36  /////////////\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 命令码 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\* //////////////  /////////////\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 状态码 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\* //////////////  #define SUCCESS\_CODE 0x01 //指令执行成功  #define FAILED\_CODE 0x02 //指令执行失败  #define UNKNOW\_CODE 0X03  #define TIMER\_CODE 0x04  #define INIT\_SUCCESS 0x11  #define STOP\_SUCCESS 0x21  #define ADD\_SUCCESS 0x31  #define DOWN\_SUCCESS 0x41  #define SPEED\_SECCESS 0x51  #define TIMER\_SUCCESS 0x61  #define INIT\_FAILED 0x12  #define STOP\_FAILED 0x22  #define ADD\_FAILED 0x32  #define DOWN\_FAILED 0x42  #define SPEED\_FAILED 0x52  #define TIMER\_FAILED 0x62  /////////////\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 状态码 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\* //////////////  //维护状态反馈的发包逻辑,用以报告电机在收到各类指令之后的执行状态  void sendAck(U8 cmd,U8 param1){  U8 pkt[PKT\_LEN];  int i = 0;  for(;i<PKT\_LEN;i++){  pkt[i] = 0x00;  }  if(cmd == SUCCESS\_CODE){  switch(recarr[1]){  case INIT\_CMD:  //电机启动成功的ACK  makePacket(pkt,SUCCESS\_CODE,INIT\_SUCCESS,ZERO\_FLAG);  break;  //电机停止成功的ACK  case STOP\_CMD:  makePacket(pkt,SUCCESS\_CODE,STOP\_SUCCESS,ZERO\_FLAG);  break;  //电机调速成功的ACK  case SPEED\_CMD:  makePacket(pkt,SUCCESS\_CODE,SPEED\_SECCESS,param1);  break;  //电机加速成功的ACK, 回传占空比,0代表未知  case ADD\_CMD:  makePacket(pkt,SUCCESS\_CODE,ADD\_SUCCESS,param1);  break;  //电机减速成功的ACK  case DOWN\_CMD:  makePacket(pkt,SUCCESS\_CODE,DOWN\_SUCCESS,param1);  break;  //电机定时启动成功的ACK  case TIMER\_CMD:  makePacket(pkt,SUCCESS\_CODE,TIMER\_SUCCESS,param1);  break;  default:  //传入了未知的命令码  makePacket(pkt,UNKNOW\_CODE,ZERO\_FLAG,ZERO\_FLAG);  break;  }  }  else if (cmd == FAILED\_CODE){  //失败的ACK  switch(recarr[1]){  case INIT\_CMD:  makePacket(pkt,FAILED\_CODE,INIT\_FAILED,ZERO\_FLAG);  break;  case STOP\_CMD:  makePacket(pkt,FAILED\_CODE,STOP\_FAILED,ZERO\_FLAG);  break;  case SPEED\_CMD:  makePacket(pkt,FAILED\_CODE,SPEED\_FAILED,ZERO\_FLAG);  break;  case ADD\_CMD:  makePacket(pkt,FAILED\_CODE,ADD\_FAILED,ZERO\_FLAG);  break;  case DOWN\_CMD:  makePacket(pkt,FAILED\_CODE,DOWN\_FAILED,ZERO\_FLAG);  break;  case TIMER\_CMD:  makePacket(pkt,FAILED\_CODE,TIMER\_FAILED,ZERO\_FLAG);  break;  default:  makePacket(pkt,UNKNOW\_CODE,ZERO\_FLAG,ZERO\_FLAG);  break;  }  }  sendPkt(pkt);  }  建立通信协议的目的，是为了建立前后台交互的契约，加入检验和减少传输差错。实际上，当前台发出一个指令包到后台后，前台对后台的指令执行状态并不清楚。  因此，需要增设“状态回传”功能。状态回传是指：在通信协议的基础上维护一个状态反馈机制，在电机执行前台指令后，根据电机当前状态回传对应的状态码。状态码包的构造基于之前建立的makePacket函数，即ACK的长度为7。  状态反馈机制有两个作用：   * PC客户端可以在电机执行指令失败时，重传指令； * 方便后续白盒测试，覆盖电机程序的所有分支，并得到准确的测试结果；   每个状态ACK的定义如下：   |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | | 起始符 | 命令码 | 参数1 | 参数2 | 校验码  （高位） | 校验码  （低位） | 终止符 | 含义 | | 30H | 01H | 11H | XX |  |  | 60H | 启动成功 | | 30H | 01H | 21H | XX |  |  | 60H | 停止成功 | | 30H | 01H | 31H | 占空比 |  |  | 60H | 调速成功 | | 30H | 01H | 41H | 占空比 |  |  | 60H | 加速成功 | | 30H | 01H | 51H | 占空比 |  |  | 60H | 减速成功 | | 30H | 01H | 61H | XX |  |  | 60H | 定时成功 | | 30H | 02H | 12H | XX |  |  | 60H | 启动失败 | | 30H | 02H | 22H | XX |  |  | 60H | 停止失败 | | 30H | 02H | 32H | XX |  |  | 60H | 调速失败 | | 30H | 02H | 42H | XX |  |  | 60H | 加速失败 | | 30H | 02H | 52H | XX |  |  | 60H | 减速失败 | | 30H | 02H | 62H | XX |  |  | 60H | 定时失败 | | 30H | 03H | XX | XX |  |  | 60H | 未知指令 | | 30H | 04H | 分 | 秒 |  |  | 60H | RTC传输 |   每个ACK会在MSComm1\_OnComm事件中被解析，前台进行状态可视化。  以下是在电机在执行指令失败后，底层系统的发包流程：    图2.2指令执行失败时的发包过程   1. **流水线测试**   流水测试是指利用白盒测试代码构建多个测试样例，以使PC客户端与电机的分支条件的所有可能值都至少取到一次，即每个分支的代码都会被覆盖一次以上。具体的实现如下：  'Suite1 : 测试电机启动/停止  Private Sub Command5\_Click()  b = sendPkt(&H30, &H31, &H0, &H0, &H0, &H61) '发送启动电机指令,使得校验和对,期待结果为电机启动'  Sleep (5000)    b = sendPkt(&H30, &H32, &H0, &H0, &H0, &H62) '发送停止电机指令,使得校验和对,期待结果为电机停止'  Sleep (5000)    b = sendPkt(&H30, &H31, &H0, &H0, &H0, &H42) '发送启动电机指令,使得校验和错,期待结果为电机反馈启动失败信息'  Sleep (5000)    b = sendPkt(&H30, &H31, &H0, &H0, &H0, &H61) '发送启动电机指令,使得校验和对,期待结果为电机启动'  Sleep (5000)    b = sendPkt(&H30, &H32, &H0, &H0, &H0, &H61) '发送停止电机指令,使得校验和错,期待结果为电机反馈停止失败信息'  Sleep (5000)    b = sendPkt(&H30, &H32, &H0, &H0, &H0, &H62) '发送停止电机指令,使得校验和对,期待结果为电机停止'  End Sub  'Suite2: 测试电机的调速模式  Private Sub Command6\_Click()  Dim i As Integer  Dim ration As Integer  Dim lowSum As Byte  Dim rb As Byte  ration = 10  i = 0  Sleep (2000)  b = sendPkt(&H30, &H33, &H0, &H0, &H0, &H63) '发送调速指令,使得校验和对,期待结果为电机进入调速模式并反馈'  Sleep (2000)  '占空比从10增加到30,期待结果为:电机从快到慢转动  For i = 0 To 10  rb = (ration \ 10) \* 16 + (ration Mod 10)  lowSum = calSum(&H30, &H33, rb, &H0)  b = sendPkt(&H30, &H33, rb, &H0, &H0, lowSum)  Sleep (2000) '每隔2s减速一次  ration = ration + 2  Next    '占空比从30递减到10,期待结果为:电机从慢到快转动  ration = 30  For i = 0 To 10  rb = (ration \ 10) \* 16 + (ration Mod 10)  lowSum = calSum(&H30, &H33, rb, &H0)  b = sendPkt(&H30, &H33, rb, &H0, &H0, lowSum)  Sleep (2000) '每隔2s加速一次  ration = ration - 2  Next    b = sendPkt(&H30, &H32, &H0, &H0, &H0, &H62) '退出调试模式'  rb = (ration \ 10) \* 16 + (ration Mod 10)  lowSum = calSum(&H30, &H33, rb, &H0)    Sleep (2000)  b = sendPkt(&H30, &H33, &H0, &H0, &H0, &H65) '发送调速指令,使得校验和错,,期待结果为电机未进入调速模式并反馈'    '未成功进入调速模式,发送了加速命令  Sleep (2000)  b = sendPkt(&H30, &H0, rb, &H0, &H0, lowSum) '发送不带调速FLAG的加速指令,期待结果为电机调速错误，当前不处于调速模式'  Sleep (2000)  b = sendPkt(&H30, &H33, rb, &H0, &H0, lowSum) '发送带调速FLAG的加速指令,期待结果为电机调进入速模式调速成功'  Sleep (2000)  '占空比从30递减到10,期待结果为:电机从慢到快转动  ration = 30  For i = 0 To 10  rb = (ration \ 10) \* 16 + (ration Mod 10)  lowSum = calSum(&H30, &H33, rb, &H0)  b = sendPkt(&H30, &H33, rb, &H0, &H0, lowSum)  Sleep (2000) '每隔2s加速一次  ration = ration – 2  Next  End Sub | | | |
| 体会 | **刘婷珺：**  本次硬件课程设计中，我主要负责完成老师要求的基础功能以及两个扩展功能：使用键盘上的A、D键调节电机转动的速度，以及利用RTC部件实现电机的过十秒启动。我们在老师提供的模板基础上进行开发，由于是第一次系统地将软硬件结合起来开发，我们不仅要了解底层硬件的工作原理，还要掌握上位机与控制器之间的通信方式。刚接触老师要求的通信协议时，由于未学过VB编程语言，并且对ASCII码传输方式了解不深入，我们遇到了很多困难，就连数据存储方式是16进制还是10进制都困扰了我们很久。但我们没有放弃，通过增加调试信息来检验存储的数据以及计算的结果是否正确，并不断咨询老师，以对通信协议有一个更加全面的认知。在编写下位机代码时，我们通过查阅书本以及参考以往的实验资料，对各个硬件模块有了全新的了解，尤其是Timer部件和RTC部件，掌握了其中主要寄存器的功能以及编写代码时，应如何设置相关寄存器的值。  但我们的系统仍存在不足，在实现定时启动功能时，我们本来是想通过在下位机端设置一个固定的时间来实现定时启动。但由于这样操作，会增加我们的测试开销：每次测试时都需要重新设置该时间，并将代码下载到实验箱，我们将该功能改为过十秒启动。经过老师指点，我们发现，可以通过传送两条指令来实现定时启动功能，即：第一条指令用于传送PC机当前时间以设置RTC的初值，第二条指令则将每次需要启动的时间传送过去，以完成定时启动。这样既方便用户的使用，也增加了系统灵活性。  **刘祥德：**  本次硬件课程设计中，我主要负责实现1个基础功能：电机传输协议层；2个扩展功能：状态反馈机制的实现、基于PC机前台程序的流水测试。在这次设计中，我学习到很多东西：  第一，我掌握了上位机与控制器之间的通信原理。在实现前后台通讯的过程中，我遇到了PC接收Buffer的越界问题，通过向老师请教才知道即使是在实验室环境，还是会存在很多电磁干扰，使得PC误认为接收到了字符，触发了接收事件。在这个基础上，我学会使用VB的串口通信包设置接收阈值，规定每次接收到字符的数目，避免了单字符传送引起的歧义问题，很好地拓展了PC客户端通信协议栈的表征能力。  第二，我学习到了通信协议的编写方式。通俗来讲，协议就是通信双方的契约，双方需要听得懂对方在说什么，就需要制定一系列的规则。我们在构造检验码的时候，无法理解为何需要将检验和的高位和低位转化为ASCII码。通过老师指导，我们才知道，这是一种国际标准，如果实现的通信协议是私有的，那么我们可以用自己的通信标准；但是，一旦用在国际上，我们的通信协议就无法兼容国际，写出来的通信程序兼容性很差，这才是我们要制定协议的原因。另外，因为嵌入式系统不存在所谓的操作系统，因此计算机网络中的OSI 7层协议栈就需要我们手动在底层实现，类似TCP/IP的协议是不存在，而现在我在控制器中实现的协议，类似是一种简化版的不可靠的UDP协议，不具备所谓的错误重传功能。  第三，我借鉴了HTTP通信的状态码机制，应用在当前的系统上。在通信协议的基础上，我成功维护一个状态反馈机制，在电机执行前台每一个指令后，根据电机当前状态回传对应的状态码。通过和符老师的交流，我学会使用状态码反馈，书写了一个全自动的测试脚本，设计了多个测试样例，完成了对电机底层代码分支的白盒测试，得到我的期望结果。而且通过测试，我也的确发现了几个致命的系统Bug，如：调速模式状态位丢失导致的电机停转、发包过快导致的电机异常、分支Break缺失带来的指令执行错误；  第四，我重新了认识了“计算机系统”的概念。作为软工的学生，我之前更多的学习的重心是放在软件系统设计与实现上。这次课设，我更多时间是在底层钻研，我搞明白了串口读、写的触发条件、电机转速与电压加持之间的关系，RTC计数的方式，寄存器与八段LED、RTC……这些对我而言不再是很干瘪的代码，需要思考很多硬件底层的逻辑。更重要的是，我还在底层应用上了通信知识，这让我明白，一个计算机系统，必然是软件、硬件、计算机网络（通讯）上的统一，它们之间不是孤立的，而是相互依存的，这是我最大的收获。  最后，我认为自己做的系统还存在可以改进的地方，在通信过程中如果出现校验错误、包损坏等情况，作为发包方的PC的客户端应该具备指令重传功能。这样，需要我们在双方实现类似TCP的机制，对我们的挑战是巨大的。总而言之，设计的路上离不开队友的团结合作，老师的辛勤指导，知要与行合一，这是我需要在未来继续贯彻的。  **邱丽丽：**  我与刘婷珺同学共同完成了PC机向系统的通信格式的更改，实现了用A、D键进行调速，以及“10秒后”启动的功能。  我们在四天的实验时间里，遇到了许多的困难。由于一开始对底层系统以及语言的不熟悉，我们经常代码编写错误，需要查找资料，一句句的从头将代码看懂。在熟悉了以后，我们的进度就加快了。每行代码我们都能说出他的作用，在修改的时候就不会混乱。从一开始以实现老师的要求为目标到后来我们按照自己的想法，借鉴了嵌入式课程实验的RTC实验代码，加入了10秒定时启动等功能，我们都是一步一个脚印走下去的。虽然过程很漫长，但是我认为我们是可以自我骄傲的。  这次课程设计大部分编程都是相互依赖的，嵌入式系统和主机程序做不到完全的独立编写，在这个过程中，我学到了一些VB的知识，比如一些控件的使用和简单的VB语法。对于串口的通讯我也加深了了解，如何利用一定的通信格式发送命令。同时，RTC部件的使用我也更加熟练。我们共同合作，共同努力，再联合调试，我也学到了很多团队合作的经验，收获颇丰。  但是在一开始的时候我们没有努力地去读懂老师的模板而是立刻去添加自己的功能，造成了很多的事物，如果我们一开始就尝试去读懂，后来的修改会更加的简单。 | | | |