****

《程序设计实训》报告

**学 院 集成电路学院**

**专 业 集成电路设计与集成系统**

**班 级 集成电路设计与集成系统3班**

**学生1姓名学号 赵霖霏WB2324163**

**学生2姓名学号 方俊杰WB2324211**

**学生3姓名学号 张宏羽X12314016**

**课程编号 SJ53309**

**课程学分 2**

**课程学期**

**【硬件部分】**

**无线蓝牙温度监测与散热温控系统**

**一、选题背景**

随着物联网技术的蓬勃发展与智能终端的普及，短距离无线通信技术在现代电子系统中扮演着愈发关键的角色。温度作为物理世界中最基础的重要环境参数之一，在智慧农业的室温恒温控制、工业生产中的设备热力检测、冷链物流的品质保障以及智能家居医疗的体温监护等多个领域均具有核心应用价值，然而，传统的有限温度检测系统往往受限于复杂的布线工程，难以满足现代场景对于设备的便携性的要求。

基于此指导思想，本课题设计并实现了一款基于蓝牙技术的无线温度监测与散热温控系统。鉴于蓝牙技术（Bluetooth）具有低功耗、低成本、强抗干扰能力以及能与智能手机无缝对接的天然优势，本设计选用杰理科技的HC-05蓝牙模块作为通信核心。系统以AT89C52RC为主控芯片，利用美信半导体的DS18B20数字温度传感器进行高精度数据采集，并通过SPP串口协议将温度数据实时传输至移动端，当系统周围环境温度超过设定值时，自动启动降温系统对环境进行降温。本设计旨在通过具体的工程实践，探索嵌入式系统的数据采集与无线传输实现路径，实现更加智能的温控方式。

**二、方案论证(设计理念)**

**2.1设计原理**

本设计使用AT89C52RC单片机作为从机，通过One-Wire总线连接DB18B20温度传感器，并将显示的温度发送给LCD1602屏幕在从机上进行显示。当温度到达25℃以上时，AT89C52RC单片机发送PWM波驱动L293D四倍高电流半桥驱动IC驱动外部DC电机和风扇，实现环境的降温功能。基于HC05蓝牙模块，将采集的温度数据和此时的DC电机转速等级无线传输给上位机（安卓手机）进行显示。

**2.2功能要求**

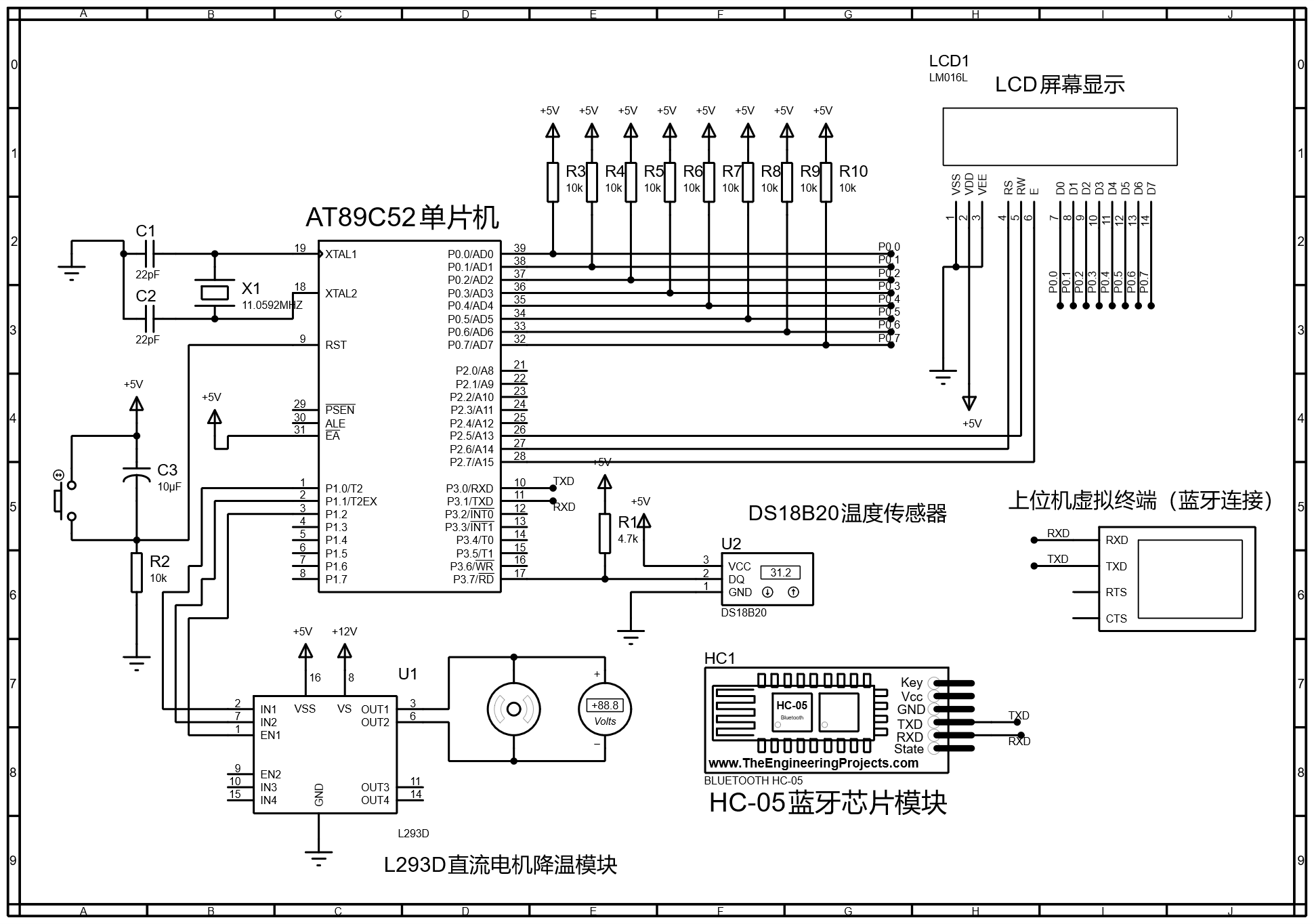
针对物联网应用背景下的远程交互需求，通信模块确立了以HC-05蓝牙模块为核心的无线透传方案。相较于Wi-Fi或ZigBee等其他无线技术，蓝牙技术在短距离通信中具备功耗低、连接建立迅速且无需复杂网络配置的显著优势。HC-05模块通过成熟的SPP串口协议与单片机进行UART通信，能够无缝对接现有的智能手机终端，不仅实现了温度数据的实时无线上传，也为后续扩展远程反向控制功能预留了接口，体现了设计方案的前瞻性与良好的用户体验。同时，本地端保留了LCD1602液晶显示模块，利用其低功耗与并行接口特性，为用户提供最直观的现场人机交互界面，确保在无线连接断开的极端情况下，系统依然具备可读性与可操作性。

在执行控制与系统安全性方面，本方案特别选用了L293D电机驱动芯片来驱动散热风扇，而非直接利用单片机I/O口驱动。L293D芯片内部集成了双H桥电路，能够有效实现逻辑控制电路与大电流驱动电路的电气隔离，防止电机启动或制动时产生的反向电动势击穿单片机端口，从而极大地提升系统的安全性与使用寿命。结合软件层面的PWM脉宽调制算法，系统能够根据当前温度偏差值动态调节电机转速，实现从静止、低速到全速的多级智能温控。这种软硬件结合的控制方案，不仅避免了传统继电器“全开全关”模式造成的能源浪费与温度震荡，更实现了平滑、高效的散热性能，完全符合现代电子系统对能效比与运行稳定性的严苛要求。

**2.3 Proteus仿真**

Proteus是一款功能强大的电子设计自动化软件，主要用于电子电路的设计，仿真和调试，它由实验中心电子（LabCenterElectronics）公司开发，广泛应用于电子工程，自动化，机科学等相关领域的教学和研究中。软件以其直观的界面，丰富的元件库和强大的仿真功能而受到用户的青昧。Proteus提供了大量的电子元件，包括电阻，电容，二极管，晶体管，单片机，传感器等，涵盖了从基础元件到复杂集成电路的各种类型.并且，它能够实时仿真电路的运行状态，用户可以观察电路的电压，电流，波形等参数变化.该软件支持多种主流单片机，如51系列，AVR系列，PIC系列，STM32系列等。Proteus支持从原理图到印刷电路板的自动转换和布线，支持印刷电路板的3D视图显示，方便用户直观地查看PCB的布局和结构。

在仿真电路绘制过程中，选取AT89C52单片机作为核心控制单元，分别连接了DS18B20温度传感器模块、LCD1602液晶显示电路、L293D直流电机驱动模块以及模拟蓝牙通信的虚拟串口终端。通过按照设计原理连接各元器件的电气线路，并加载编译后的HEX程序文件，对系统在不同温度阈值下的逻辑判断、数据显示及风扇电机PWM调速功能进行了全面的模拟与调试。具体的Proteus原理图如图2.1所示。



**图2.1 Proteus原理图**

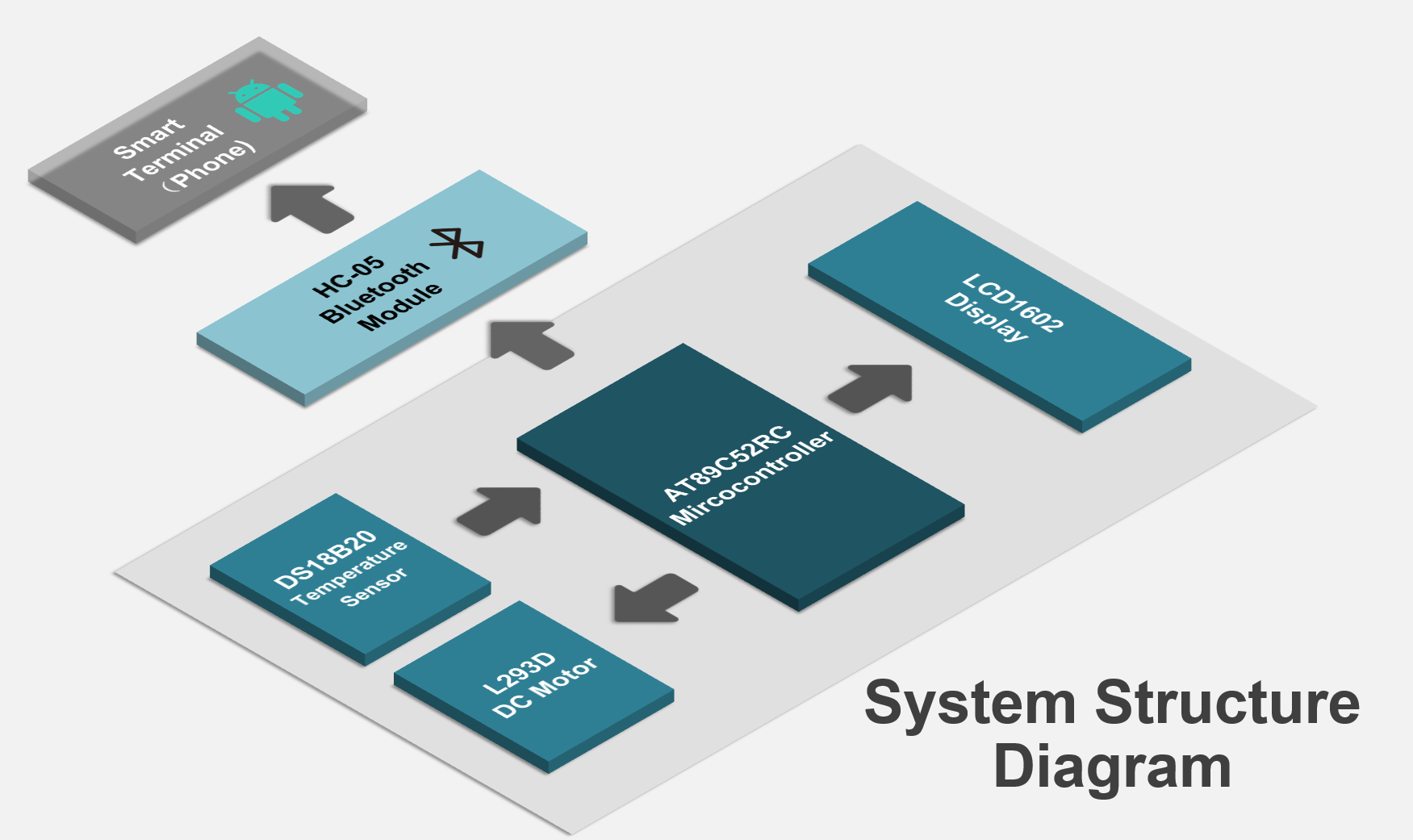
**Fig2.1 Proteus Schematic**

**三、过程论述**

**3.1 硬件系统整体设计**

本设计整体设计由DS18B20温度传感器模块、HC-05蓝牙模块、LCD1602液晶显示模块、L293D电机驱动模块及直流风扇组成，能够实时检测环境温度，并给予直观反馈。AT89C52单片机作为主控芯片，该8位单片机具有强大的性能和广泛的应用范围，非常适合小规模嵌入式系统，它具有功耗低，外围设备接口丰富，具备较大的闪存和RAM容量，易于开发编程等优点。

为了实现温度监测与智能散热的控制逻辑，我们将根据DS18B20采集的温度数据设计合适的控制策略。当传感器检测到的环境温度超过预设阈值时，系统会自动通过L293D驱动芯片启动直流风扇进行散热；同时，LCD1602显示屏会实时刷新当前检测到的温度数值，供用户直接读取。除此之外，在本设计中还可以通过HC-05蓝牙模块与上位机建立无线连接，将实时温度数据透传至远程终端，实现远程监控与数据记录，各模块如图3.2所示。



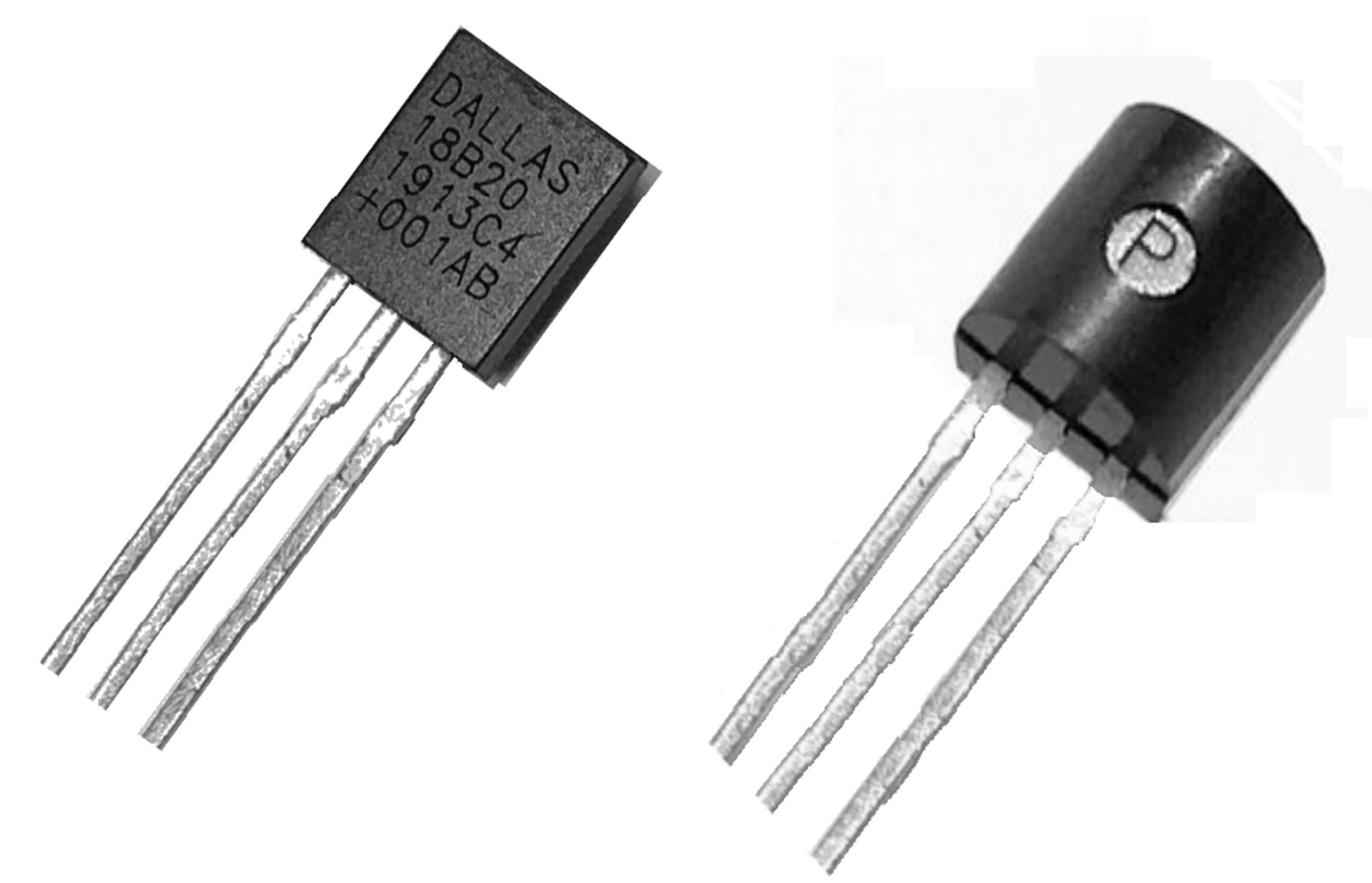
**图3.2 无线蓝牙温度监测与散热温控系统结构框图**

**Fig3.2 System structure diagram of the wireless bluetooth temperature monitoring**

**and heat dissipation temperature control system**

**3.2 DS18B20温度传感器型号及原理**

温度传感器采用DS18B20数字温度传感器。DS18B20数字温度计提供9到12位可配置的温度读数，用于指示设备的温度。信号通过One-Wire总线在DS18B20之间发送，因此从中央微处理器到DS18B20只需要连接一根线。用于读取、写入和执行温度转换的电源可以直接从数据线上获取，无需外部电源。由于每个DS18B20都包含唯一的硅序列号，因此多个DS18B20可以存在于同一个One-Wire总线上。这允许在许多不同地点放置温度传感器。应用此功能的场景包括暖通空调环境控制、建筑物内部温度传感、设备或机械的温度监测，以及过程监控和控制。DS18B20外观如图2所示。



**图3.3 DS18B20外观图**

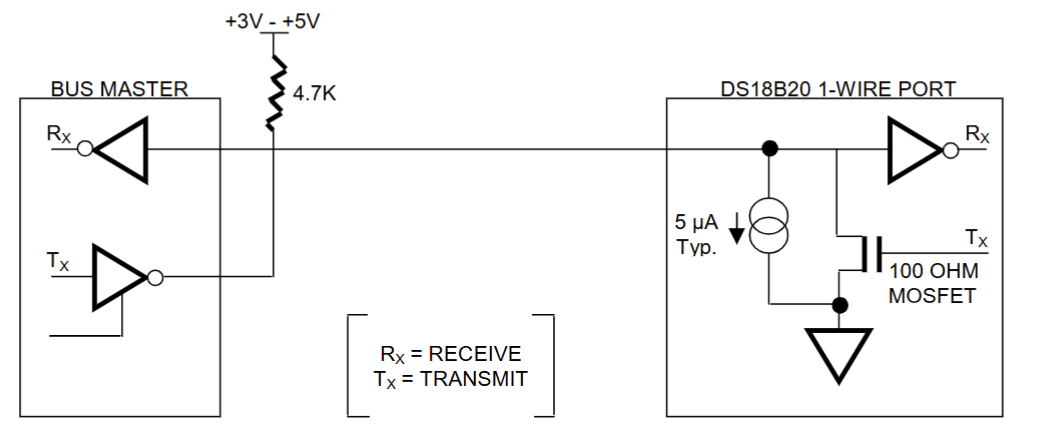
**Fig3.3 Apperence of DS18B20**

**3.3 One-Wire总线时序分析**

One-Wire总线是一种具有单个总线主设备和一个或多个从设备的系统。DS18B20行为如同一个从设备。对这种总线系统的讨论分为三个主题：硬件配置、事务序列和总线设计。

**3.3.1 One-Wire总线的硬件配置**

One-Wire总线按定义只有一条线路；重要的是每个设备在适当的时候能够驱动这条线路。为了便于实现这一点，连接到One-Wire总线的每个设备都必须具有开漏或三态输出。DS18B20的One-Wire端口（DQ引脚）是开漏输出，其内部电路等效于图3所示的电路。多设备总线是由多个从设备连接到One-Wire总线构成的。One-Wire总线需要一个大约5kΩ的上拉电阻。



**图3.4 硬件配置**

**Fig3.4 Hardware configuation**

**3.3.2** **One-Wire总线的事务序列**

通过One-Wire端口访问DS18B20的协议如下：

* 初始化
* ROM功能指令
* 存储器命令功能
* 发送/数据

**（****1）初始化**

One-Wire总线上的所有交易都以初始化序列开始。初始化序列包括由总线主设备发出的复位脉冲，随后是设备发出的存在脉冲。

**（2）ROM功能指令**

一旦总线主设备检测到存在脉冲，就可以发出五个ROM功能命令中的一个。所有ROM功能命令都是8位长。以下列出这些命令,参见图5中的流程图：

①读ROM [33h]

此命令允许总线主机读取DS18B20的8位家族码、唯一的48位序列号和8位CRC。此命令只能在总线上仅有一个DS18B20时使用。如果总线上有多个从设备，当所有从设备同时发送数据时会发生数据冲突（开漏输出将产生线与逻辑结果）。

②匹配ROM [55h]

匹配ROM命令，后跟64位ROM序列，允许总线主机在多点总线上寻址特定的DS18B20。只有与64位ROM序列完全匹配的DS18B20才会响应后续的存储器功能命令。所有不匹配64位ROM序列的从设备将等待复位脉冲。此命令可用于总线上单个或多个设备。

③跳过ROM [CCh]

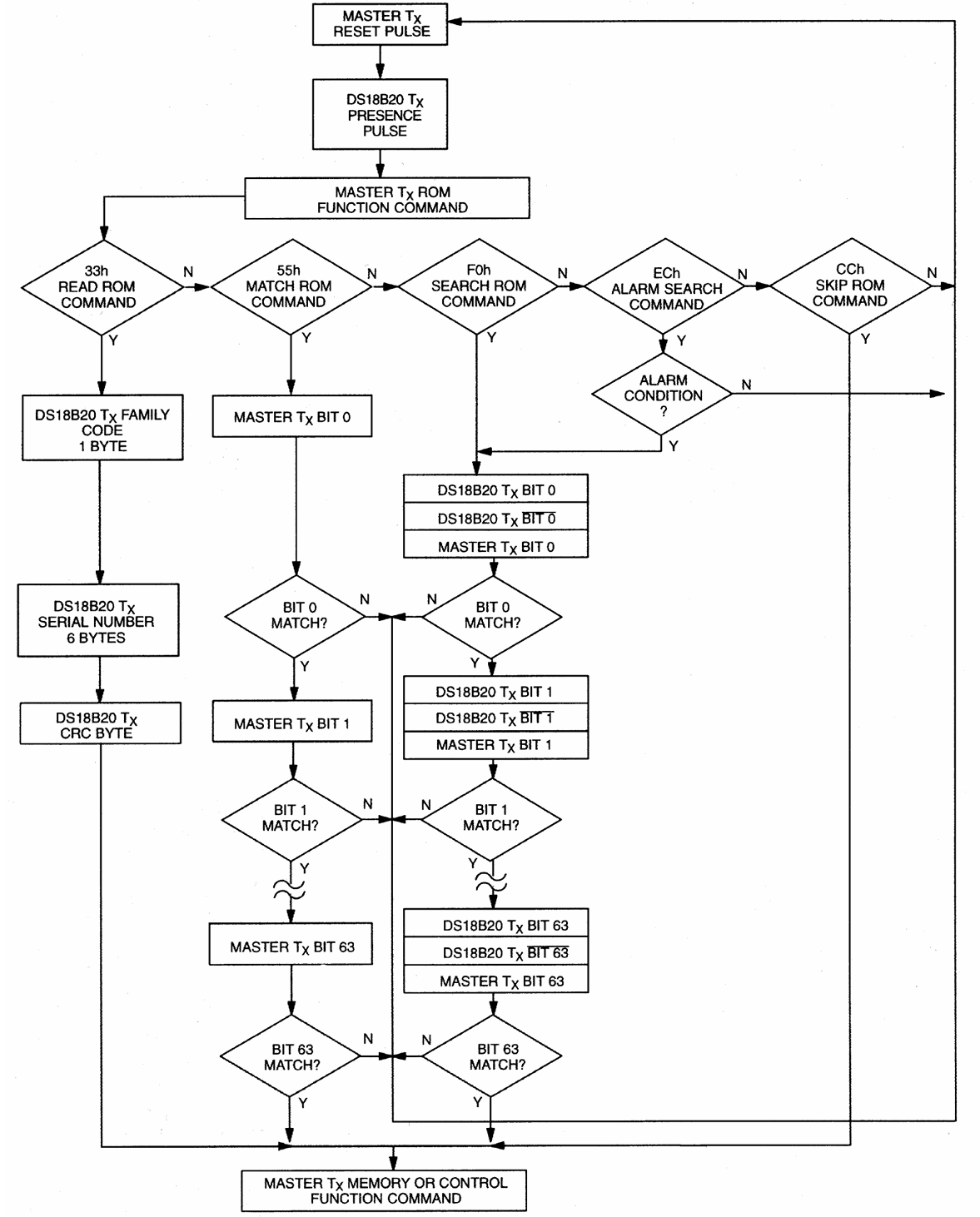
此命令可以在单总线系统中节省时间，允许总线主设备在不提供64位ROM码的情况下访问内存功能。如果总线上存在多个从设备，并在跳过ROM命令之后发出读取命令，总线上将发生数据冲突，因为多个从设备同时传输数据（开漏下拉将产生线与逻辑结果）。

④搜索ROM [F0h]

当系统首次启动时，总线主设备可能不知道One-Wire总线上设备的数量及其64位ROM码。搜索ROM命令允许总线主设备通过排除法来识别总线上所有从设备的64位ROM码。

⑤报警搜索 [ECh]

该命令的流程图与搜索ROM命令相同。然而，DS18B20仅在上次温度测量中遇到报警条件时才会响应此命令。报警条件定义为温度高于TH或低于TL。只要DS18B20通电，报警条件就会保持设置，直到另一次温度测量显示非报警值为止。在报警触发时，会考虑存储在EEPROM中的触发值。如果存在报警条件并且更改了TH或TL设置，应进行另一次温度转换以验证任何报警条件。



**图3.5 ROM功能指令流程图**

**Fig.5 ROM functions flow chart**

**（3）存储器命令功能**

①写入寄存器 [4Eh]

此命令用于向DS18B20的暂存器写入数据，从TH寄存器开始。接下来的3个字节将被保存在暂存器地址2至4的位置。在发出复位命令之前，必须全部写入这3个字节。

②读取暂存器 [BEh]

此命令读取暂存器的内容。读取将从字节0开始，持续进行直到读取到第九个（字节8，CRC）字节。如果不需要读取所有位置，主设备可以在任何时候发出复位命令以终止读取。

③复制暂存器 [48h]

此命令将暂存器的内容复制到DS18B20的E存储器中，将温度触发字节存储在非易失性存储器中。如果总线主设备在此命令之后发出读取时间槽，只要DS18B20正在将暂存器复制到E存储器中，它将在总线上输出0；复制过程完成后，将输出1。如果使用寄生电源供电，则总线主设备在发出此命令后必须立即启用强上拉至少10ms。DS18B20的EEPROM在T=+55°C时，最低可写入50,000次，数据保留时间为10年。

④转换温度 [44h]

此命令开始温度转换。无需进一步的数据。温度转换完成后，DS18B20将保持空闲状态。如果总线主设备在该命令之后发出读取时间槽，只要DS18B20正在执行温度转换，它将在总线上输出0；当温度转换完成后，将输出1。如果使用寄生电源供电，则总线主设备必须在发出此命令后，启用强上拉一段时间，该时间必须大于tconv。

⑤回读 E2[B8h]

此命令将E2存储器中存储的温度触发值和配置寄存器内容回读到暂存器中。此回读操作在DS18B20上电时也会自动发生，因此只要设备接通电源，暂存器中就会立即有有效的数据。在该命令发送后，每次读取数据时间槽时，设备将输出其温度转换器的忙标志：0=忙，1=就绪。

⑥读取电源 [B4h]

在向DS18B20发送此命令后，每个读取数据的时间槽，设备将指示其供电模式：0=寄生电源，1=外部电源供电。

**3.3.3 DS18B20传感器总线设计**

DS18B20主要包含4部分：ROM、温度传感器、TH和TL、配置寄存器。ROM是一个64位的存储器，它是DS18B20的ID，也是DS18B20的地址序列码，出厂之前就光刻好，用来识别单总线上挂接的多个DS18B20身份：温度传感器，完成对温度的测量与转换：TH和TL，用来完成低温高温触发报警；配置寄存器，用来设置温度精度。

DS18B20单总线通信方式是通过指令的读写来实现，主要指令包括ROM指令与功能指令两种。本文采用指令有[CCH]、[44H]、[BEH]，其中[CCH]属于ROM指令，允许总线控制器无须提供DS18B20的64位ROM码就可以对DS18B20进行读写操作；[44H]属于功能指令，完成DS18B20一次的温度转换；[BEH]属于功能指令，用来读取DS18B20高速暂存器中的数据。

首先进行初始化，首先拉低总线至少480μs，维持时间不得超过960μs，即此时总线信号OneWire = 0，持续的低电平信号作为全局复位指令，能够强制总线上挂载的DS18B20复位其内部逻辑，做好接收后续指令的准备；在释放总线后，单片机进入监听状态。依据DS18B20的时序特性，传感器在检测到总线上的上升沿信号后，会等待15~60μs。随后主动将总线再次拉低60~240μs，以此发出脉冲信号，表明DS18B20在线且工作正常。根据上述分析，可得DS18B20初始化代码如下：

usigned char OneWire\_Init()

{

unsigned char i;

unsigned char AckBit;

OneWire =1;

OneWire =0;

i = 247;

while (--i); *// 总线拉低至少480μs*

OneWire =1; *// 释放总线*

i = 32;

while (--i); *// 在60~240μs内从机会拉低总线*

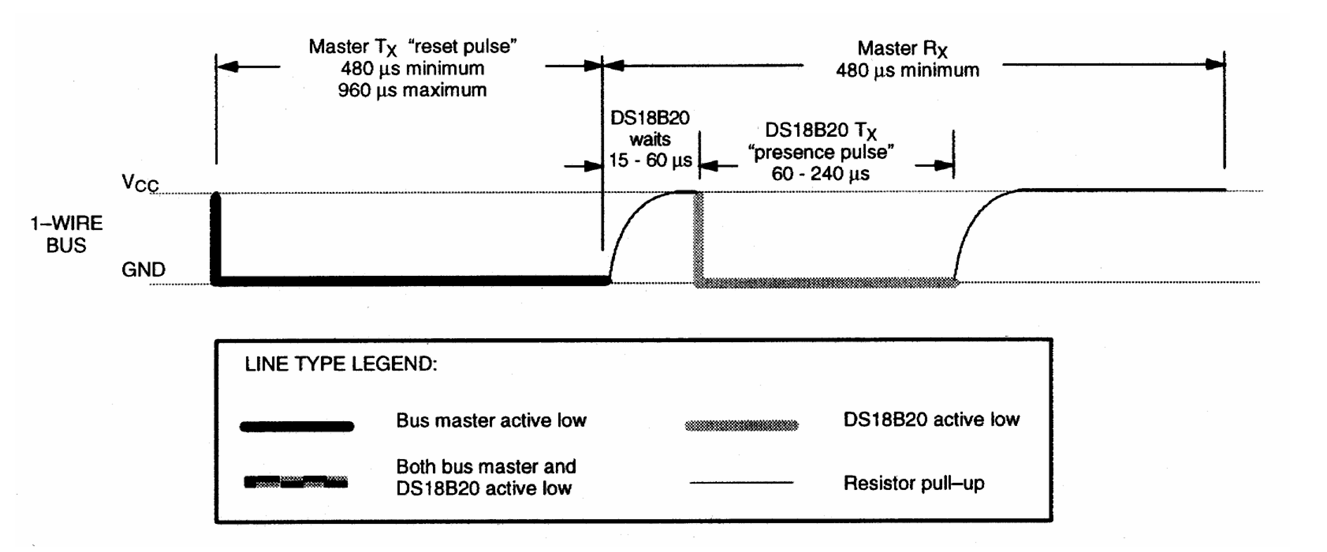
AckBit=OneWire ;

i = 247;

while (--i); *//Delay 500μs*

return AckBit;

}



**图3.6 DS18B20初始化时序图**

**Fig3.6 DS18B20 Initialization Timing Diagram**

DS18B20接收到ROM指令[CCH]后，OneWire总线就不会返回64位ROM编码，便于驱动单个DS18B20。如果单总线上挂载多个DS18B20，则需发送ROM指令以执行ROM循环搜索算法，从而识别所有从机设备。向DS18B20写数据命令时序可分为两部分，每部分周期为60~120μs，如图6所示。第一部分写0，时序开始时单片机将总线拉低至少1ps，并维护低电平至少60μs至写周期结束，随后释放总线为高电平。第二部分写1，时序开始时单片机把总线拉低1μs后，释放总线为高电平，直至写周期结束。DS18B20检测到OneWire总线被拉低后，将等待15μs，并在15~45ps期间对总线采样，在周期内总线若采样为低电平，则判定为0，若采样为高电平，则判定为1。DS18B20读数据单比特函数代码如下：

usigned char OneWire\_ReceiveBit()

{

unsigned char i;

unsigned char Bit;

OneWire\_DQ=0;

i = 2;

while (--i); *//Delay 5us*

OneWire\_DQ=1;

i = 2;

while (--i); *//Delay 5us*

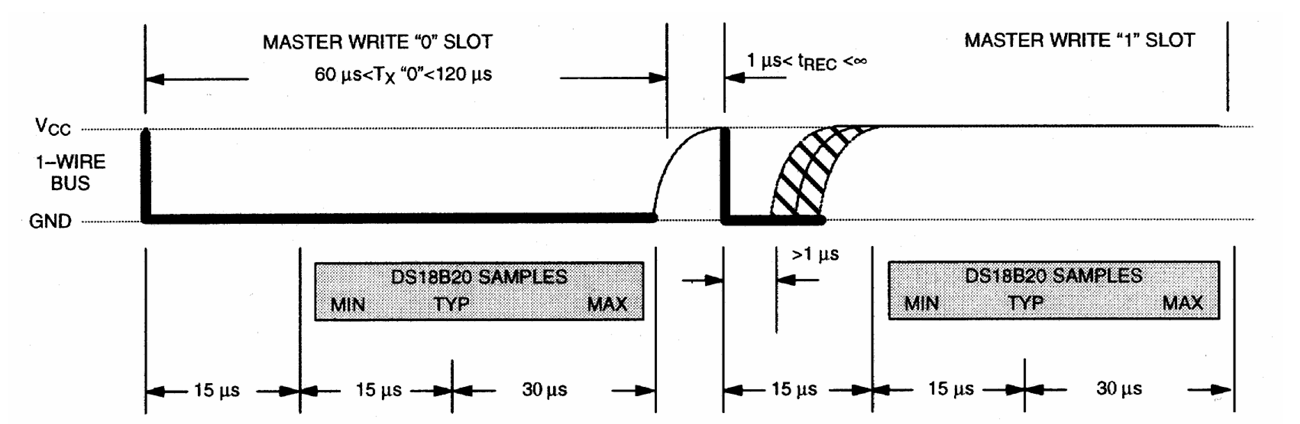
Bit=OneWire\_DQ;

i = 24;

while (--i); *//Delay 50us*

return Bit;

}



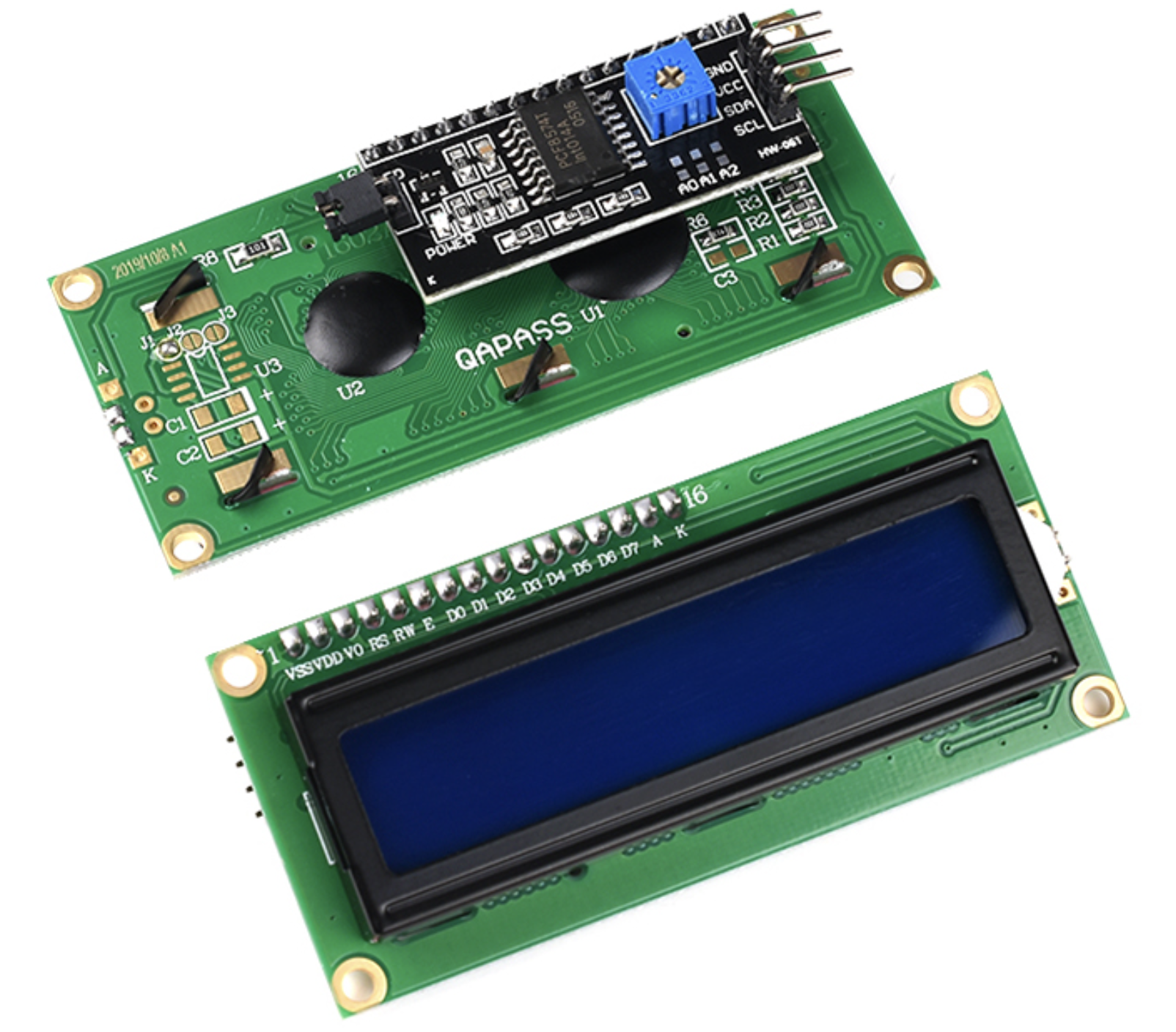
**图3.7 DS18B20读数据时序图**

**Fig3.7 DS18B20 Read Data Timing Diagram**

**3.4 LCD1602模块分析**

LCD1602液晶为点阵型液晶模块，专为字母，数字和符号显示而设计。它采用了一一点阵技术，将每一行分成若干小块，并在每个

小块间填充不同颜色的像素，形成具有特殊结构的阵列图形，从而产生所需要的各种图像。LCD1602外观如图7所示。



**图3.8 LCD1602外观图**

**Fig3.8 Apperence of LCD1602**

LCD1602液晶的管脚VSS接5V电源负极，即GND。管脚VDD接5V电源正极。管脚VO的主要作用是提供LCD1602液晶的偏压信号的说明，显示液晶的对比度并做出对应调整措施，对比度处于最低点时是管脚VO接电源的时候，处于最高点时是管脚VO接地面的

时候，如果对比度超过常规的范围，那么在这个时候会产生“残影”的现象，此时液晶屏幕上产生的字符显示会较为不清晰。

管脚RS功能为LCD1602液晶选择不同的寄存器，有两种寄存器可供选择，如若此时电平是高电平，数据寄存器会被管脚RS选择，如若此时是低电平，指令寄存器会被管脚RS选择。管脚RW为读/写选择管脚，当该管脚为高电平时进行读操作，当该管脚为低电平时进行写操作。

管脚E是用来控制信号的输入端，当电平从高电平转变为低电平时，相关命令就会被液晶调用执行。D0~D7为液晶8位双向数据线，分别接STC89C52单片机的P00~PO7引脚。BLA管脚接液晶模块背光源正极。BLK管脚接液晶模块背光源负极。将LCD1609的所有引脚功能汇总内容如表3.1所示。

**表3.1 LCD1602引脚功能表**

**Table3.1 LCD1602 Pin Function Table**

|  |  |  |
| --- | --- | --- |
| **编号** | **符号** | **引脚说明** |
| 1 | VSS | 电源地 |
| 2 | VDD | 电源正极 |
| 3 | VL | 液晶显示偏压 |
| 4 | RS | 数据/命令选择 |
| 5 | R/W | 读/写选择 |
| 6 | E | 使能信号 |
| 7 | D0 | 数据 |
| 9 | D2 | 数据 |
| 10 | D3 | 数据 |
| 11 | D4 | 数据 |
| 12 | D5 | 数据 |
| 13 | D6 | 数据 |
| 14 | D7 | 数据 |
| 15 | BLA | 背光源正极 |

在系统初始化阶段，AT89C52单片机首先对LCD1602显示模块进行底层端口配置与时序复位。依据硬件电路连接，单片机的P2.6、P2.5及P2.7引脚分别被定义为LCD\_RS（数据/指令选择）、LCD\_RW（读写选择）及LCD\_EN（使能信号），而P0端口则作为8位并行数据总线负责信息的传输。初始化过程LCD\_Init严格遵循HD44780控制器的标准指令集：首先写入指令0x38以设定8位数据接口及两行显示模式，确保显示缓冲区格式正确；随后写入0x0C开启屏幕显示并关闭光标，以获得清晰的视觉效果；写入0x06设置光标在读写操作后自动加一，实现字符的连续输入；最后发送0x01指令清除屏幕残留数据并复位光标。

void LCD\_Init()

{

LCD\_WriteCommand(0x38);*//八位数据接口，两行显示，5\*7点阵*

LCD\_WriteCommand(0x0c);*//显示开，光标关，闪烁关*

LCD\_WriteCommand(0x06);*//数据读写操作后，光标自动加一，画面不动*

LCD\_WriteCommand(0x01);*//光标复位，清屏*

}

LCD1602的底层驱动核心依赖于指令写入与数据写入的时序配合。函数LCD\_WriteCommand与LCD\_WriteData通过控制RS引脚的电平状态来区分操作类型：当RS置低电平时，P0口传输的数据被视为控制指令；当RS置高电平时，数据被视为待显示的字符ASCII码。在RW引脚拉低确立写模式后，系统通过EN引脚由高电平跳变至低电平，利用下降沿将P0端口的数据锁存至LCD内部控制寄存器或数据寄存器中。为了适配LCD内部处理速度，每个操作周期内均插入LCD\_Delay延时函数，防止因单片机指令执行过快而导致的数据丢失。

void LCD\_WriteCommand(unsigned char Command)

{

LCD\_RS=0;

LCD\_RW=0;

LCD\_DataPort=Command;

LCD\_EN=1;

LCD\_Delay();

LCD\_EN=0;

LCD\_Delay();

}

void LCD\_WriteData(unsigned char Data)

{

LCD\_RS=1;

LCD\_RW=0;

LCD\_DataPort=Data;

LCD\_EN=1;

LCD\_Delay();

LCD\_EN=0;

LCD\_Delay();

}

LCD\_SetCursor函数据输入的行号与列号计算出对应的DDRAM地址（第一行起始地址为0x80，第二行起始地址为0x80+0x40），并通过写指令方式将光标移动至指定位置。

void LCD\_SetCursor(unsigned char Line,unsigned char Column)

{

if(Line==1)

{

LCD\_WriteCommand(0x80|(Column-1));

}

else if(Line==2)

{

LCD\_WriteCommand(0x80|(Column-1+0x40));

}

}

LCD\_ShowString函数函数利用循环结构遍历传入的字符串数组，逐字节调用写数据接口，直至遇到结束符。

void LCD\_ShowString(unsigned char Line,unsigned char Column,char \*String)

{

unsigned char i;

LCD\_SetCursor(Line,Column);

for(i=0;String[i]!='\0';i++)

{

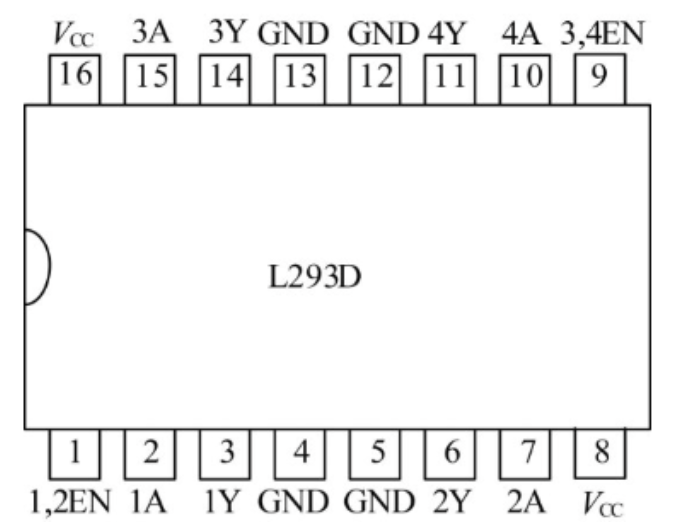
LCD\_WriteData(String[i]);

}

}

**3.5 L293D电机驱动模块分析**

L293D主要接收控制器指令，控制电机运转、停止。L293D各引脚图如图8所示，其中引脚8、引脚16为电源端；引脚4、引脚5、引脚12、引脚13为接地端；引脚1、引脚9为使能端；引脚2和引脚3、引脚6和引脚7、引脚14和引脚15、引脚10和引脚11为4个通道，其中A为输入，Y为输出。其逻辑功能如表2所示。



**图3.9 L293D引脚图**

**Fig3.9 L293D Pin Diagram**

**表3.2 L293D逻辑功能**

**Table3.2 L293D Logic Function**

|  |  |  |
| --- | --- | --- |
| **输入** | | **输出Y** |
| EN | A |
| H | H | H |
| H | L | L |
| L | X | Z |

注：H为高电平，L为低电平，X为任意值，Z为高阻态。

从表2可以看出：当使能端EN为高电平时，输人A的状态与输出Y的状态一致。当使能端为低电平时，无论输人A为高电平还是低电平，输出Y均为高组态。

**3.5.1 直流电机驱动模块软件设计**

L293D芯片内部集成了双H桥电路，能够有效隔离单片机弱电控制信号与电机强电驱动回路，增强系统的抗干扰能力。在硬件连接逻辑中，AT89C52单片机的P1.0与P1.1引脚分别连接至L293D的逻辑输入端IN1与IN2，通过设定固定电平（IN1=0, IN2=1）确保风扇单向旋转；P1.2引脚连接至使能端ENA，作为PWM信号的输出端口。初始化函数Fan\_Init主要负责将上述控制引脚复位，并配置定时器0工作于模式2。该模式下，TL0计数溢出后会自动装载TH0的值，产生精确且稳定的时间基准。

void Fan\_Init(void)

{

FAN\_IN1 = 0;

FAN\_IN2 = 1;

FAN\_ENA = 0;

TMOD = (TMOD & 0xF0) | 0x02; *// Timer0 方式2，8位自动重装*

TH0 = 156; *// 定时周期约100us @12MHz*

TL0 = 156;

ET0 = 1; *// 开启定时器0中断*

EA = 1; *// 开启总中断*

TR0 = 1; *// 启动定时器*

}

使用AT89C52RC单片机中的定时器中断服务程序Timer0\_Routine实现的软件PWM算法。pwm\_counter作为周期计数器，PWM\_Duty作为占空比阈值。每当定时器溢出触发中断，计数器自增；单片机实时比较当前计数值与设定阈值：当计数值小于PWM\_Duty时，拉高ENA引脚电平；反之则拉低。

void Timer0\_Routine(void) interrupt 1 *// 定时器0中断服务程序*

{

pwm\_counter++;

if (pwm\_counter >= 100) pwm\_counter = 0;

if (pwm\_counter < PWM\_Duty) {

FAN\_ENA = 1; *// 输出高电平*

} else {

FAN\_ENA = 0; *// 输出低电平*

}

}

为了实现直流电机的分速调节，定义0至3的档位参数：档位0对应0%占空比，风扇停止；档位1对应约70%占空比，风扇维持低速运行；档位2对应约83%占空比；档位3则输出100%全速占空比，风扇全速运行。对应的直流电机风扇转速如表3所示，

**表3.3直流电机风扇分速调节**

**Table3.3** **DC motor fan speed regulation**

|  |  |
| --- | --- |
| **档位** | **方波占空比** |
| 0 | 0% |
| 1 | 70% |
| 2 | 83% |
| 3 | 100% |

直流电机风扇转速软件代码如下所示。

void Fan\_SetLevel(unsigned char level)

{

FAN\_IN1 = 0;

FAN\_IN2 = 1;

switch (level) {

case 0: PWM\_Duty = 0; break; *// 停止*

case 1: PWM\_Duty = 70; break; *// 低速*

case 2: PWM\_Duty = 83; break; *// 中速*

case 3: PWM\_Duty = 100; break; *// 全速*

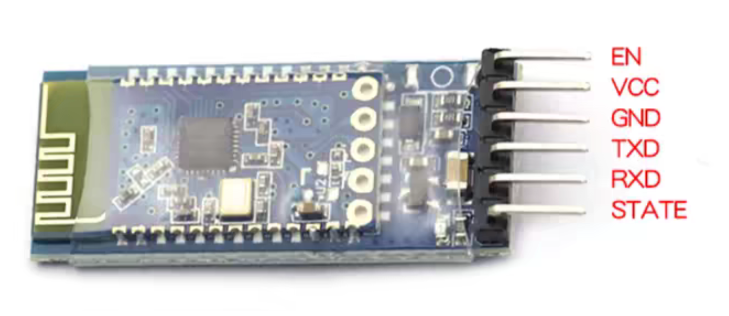
default: PWM\_Duty = 0; break;

}

}

**3.6 HC05蓝牙串口模块分析**

HC05蓝牙串口模块与单片机接口相连，体积小，指令简洁，仅有6个管脚，管脚间距为2.54m，焊接方便。蓝牙模块如图9所示。



**图3.10 HC05蓝牙外观图**

**Fig3.10 Appearance of HC05 Bluetooth**

具体的管脚功能如表所示。

**表3.4 HC05管脚功能表**

**Table3.4 HC05 Pin Function Table**

|  |  |
| --- | --- |
| **引脚** | **方波占空比** |
| STATE | 连接状态引脚 |
| RXD | 串口输入引脚（TTL电平） |
| TXD | 串口输出引脚（TTL电平） |
| GND | 电源地 |
| VCC | 电源（支持3.6V~6V） |
| EN | 空 |

主控制器MCU中蓝牙通信单元采用HC05蓝牙模块，HC05为主从一体式串口蓝牙模块，即在进行通信时无需考虑蓝牙协议，把其当作串口即可。该模块的工作频率范围为4800-1382400Hz，可用在5V或3.3V单片机系统中。具有易开发、灵敏度高和性价比高等优点。HC05的连线方式如图3.2所示。



**图3.11 HC05蓝牙模块工作原理图**

**Fig3.11 Working principle of HC05 Bluetooth module**

由图10可得，移动设备向控制器发送数据时，先以无线通信的方式发送给HC05，然后HC05通过TXD端发给控制器的RXD端。该模块的VCC和GND分别接到3.3V电源和地。AAT59C52RC通过串口与蓝牙进行通信。即HC05的串口接收引脚与AT59C52RC的串口发送引脚连接，HC05的串口发送引脚与AT59C52RC的串口接收引脚连接。

蓝牙HC05具有命令响应和自动连接两种工作模式。并且在处于自动连接工作模式下时，又可以分为主（Master），从（Slave）和回环（Loopback）三种工作角色。当HC05处于自动连接模式，便会自动根据之前设定的连接方式传输数据。当HC05处于命令响应模式时，可以执行AT命令，用户可通过向蓝牙发送AT指令，为蓝牙设定控制参数或者发布控制指令.通过控制模块外部引脚输入电平，可以实现模块工作状态的动态转换。

采用HC-05蓝牙模块实现下位机AT89C52RC与上位机（安卓手机移动终端）之间的无线数据透传。软件设计主要包含底层串口驱动的配置与应用层数据传输协议的实现两部分。系统通过通用异步收发传输器（UART）接口，将DS18B20采集的温度数据及风扇当前的运行状态实时发送至远程终端。

首先使用USB转串口模块CH340连接HC05，通过XCOM串口调试软件进行串口AT指令集的设定，对于AT指令的设定可以参考下列表。

**表3.5 串口AT指令集**

**Table3.5 UART AT Commands**

|  |  |  |
| --- | --- | --- |
| **序列** | **指令** | **功能** |
| 1 | AT + VERSION | 版本号 |
| 2 | AT + RESET | 软复位 |
| 3 | AT + DISC | 断开连接（连接状态下有效） |
| 4 | AT + LADDR | 查询模块的MAC地址 |
| 5 | AT + PIN | 连接密码设置与查询 |
| 6 | AT + BAUD | 波特率设置与查询（默认9600bps） |
| 7 | AT + NAME | 广播名设置与查询 |
| 8 | AT + DEFAULT | 恢复出厂设置 |
| 9 | AT + ENLOG | 串口状态输出使能 |

我们在安卓手机上安装“蓝牙调试助手”应用，连接目标蓝牙，在AT89C52RC烧录之后即可看到DS18B20采集的温度数据，以及直流电机驱动的风扇的转速等级。



**图3.12 上位机蓝牙接收UI界面**

**Fig3.12 Bluetooth Receiver UI**

**3.7 本章小结**

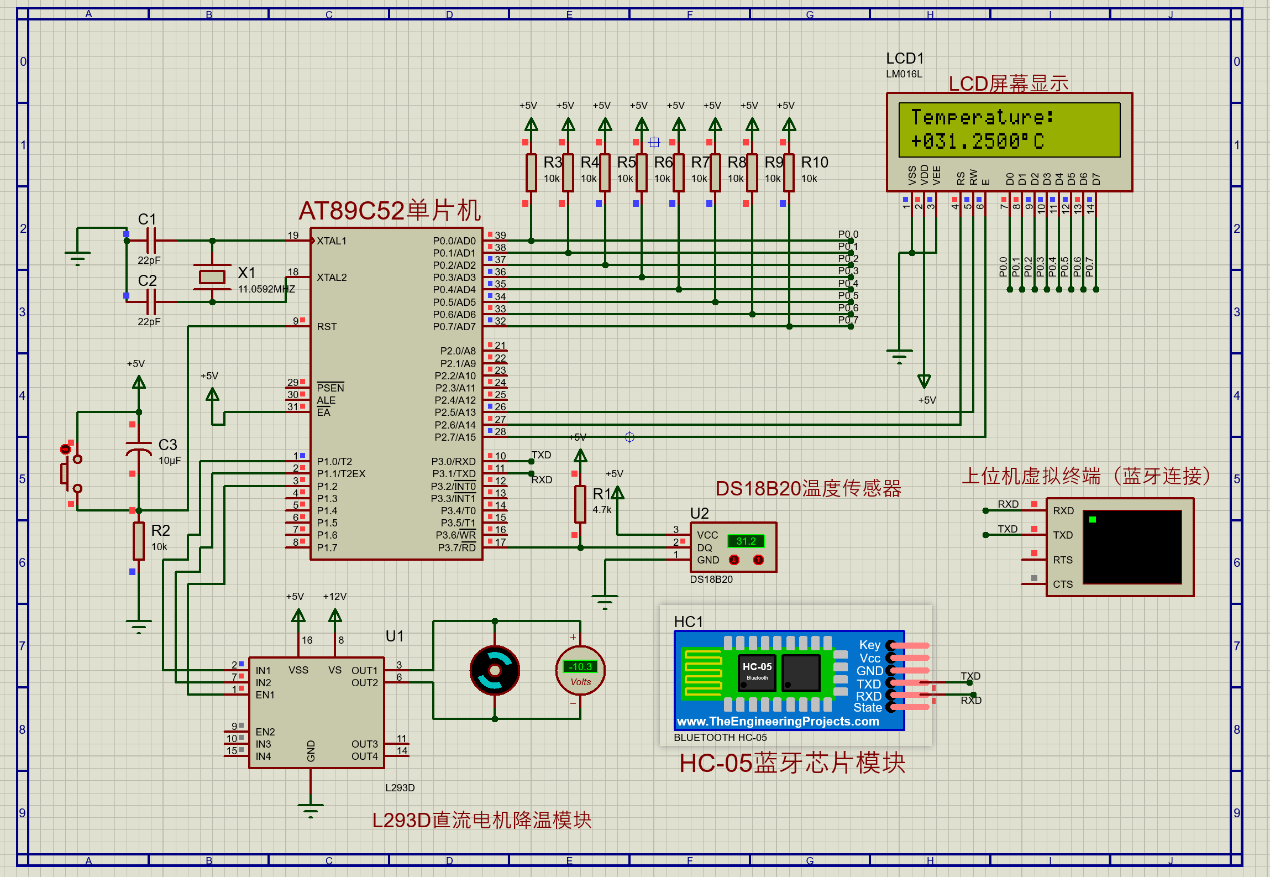
本章对无线温度监测与散热温控系统的硬件设计进行详细描述。首先研究该系统的硬件架构，其由主控核心AT89C52RC、温度传感器DS18B20、县食品LCD1602、电机驱动L293D及蓝牙模块HC-05构成。本章对DS18B20单总线读写时序、LCD1602指令驱动流程、L293D双H桥驱动原理及PWM脉宽调制算法进行详细设计。同时还对数字温度传感器特性、液晶模块引脚定义、电机驱动逻辑真值表、蓝牙串口通信协议及AT指令配置方法进行简单介绍。

**四、结果分析**

**4.1 Proteus仿真报告**

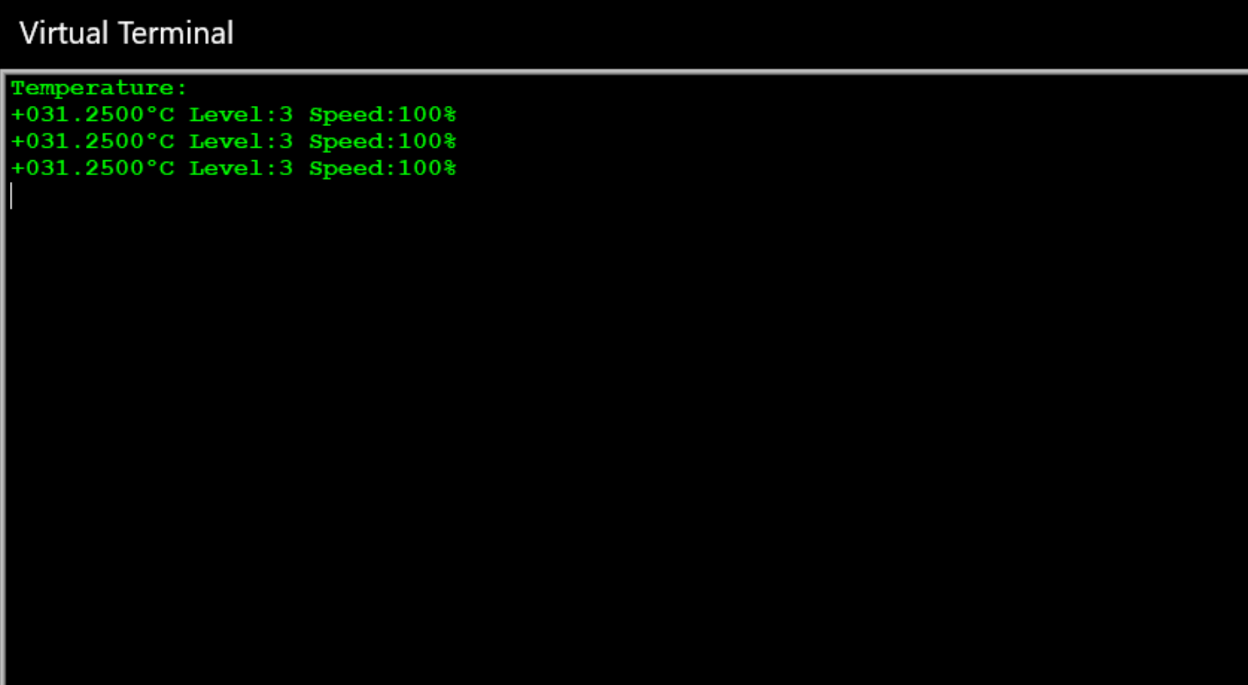
在Proteus8.13环境下单间仿真电路，首先通过“Pick Devices”功能添加AT89C52RC、DS18B20、LCD1602、L293D等模块。从外部导入HC05蓝牙模块的Lib库，按照硬件设计连接电路，注意在仿真时添加10KΩ的上拉电阻。并在单片机属性中设置晶振频率为12MHz，加载编译生成的HEX文件。

当温度超过25℃的时候，AT89C52RC单片机发送使能信号给L293D的EN1端口，此时电机正常旋转。此时虚拟终端能够正常现实温度和电机的转速等级。如图4.13和4.14所示。



**图4.13 Protues仿真运行图**

**Fig4.13 Proteus Simulation**

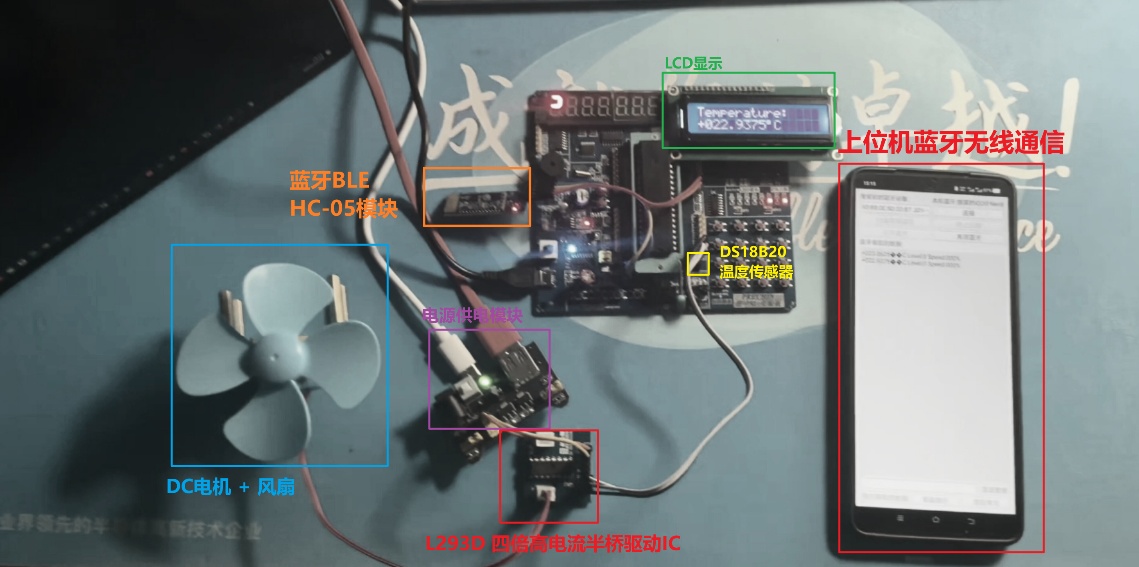


**图4.14 Proteus虚拟终端输出**

**Fig 14 Proteus Virtual Terminal**

**4.2 实物演示**

按照原理图对所有模块进行连接，注意在使用stc-isp烧录的过程中，需要断开AT89C52RC的P3.0和P3.1，然后再连接蓝牙BLE HC-05模块。手机打开“蓝牙调试助手”，然后连接“蓝牙JDY-31-SPP”，输入密码1234，即可成功配对，具体的实物图如图4.15所示。



**图4.15 实物图**

**Fig4.15 Prototype Photograph**

烧录完程序之后，系统会自动检测周围环境的温度，当温度达到一定的范围之后，风扇会自动进行启动，在一定的范围会执行不同的转速，如表4.6所示。

**表4.6 温度与电机转速对照表**

**Table4.6 Temperature vs. Motor Speed**

|  |  |  |
| --- | --- | --- |
| **温度** | **电机转速等级** | **PWM占空比** |
| ＜25℃ | 0 | 0% |
| 25℃~27℃ | 1 | 70% |
| 27℃~30℃ | 2 | 83% |
| ＞30℃ | 3 | 100% |

此时打开手机的“蓝牙调试助手”，即可在手机中查看单片机传输的数据。如图4.16所示。



**图4.16 上位机蓝牙接收UI界面**

**Fig4.16 Bluetooth Receiver UI**

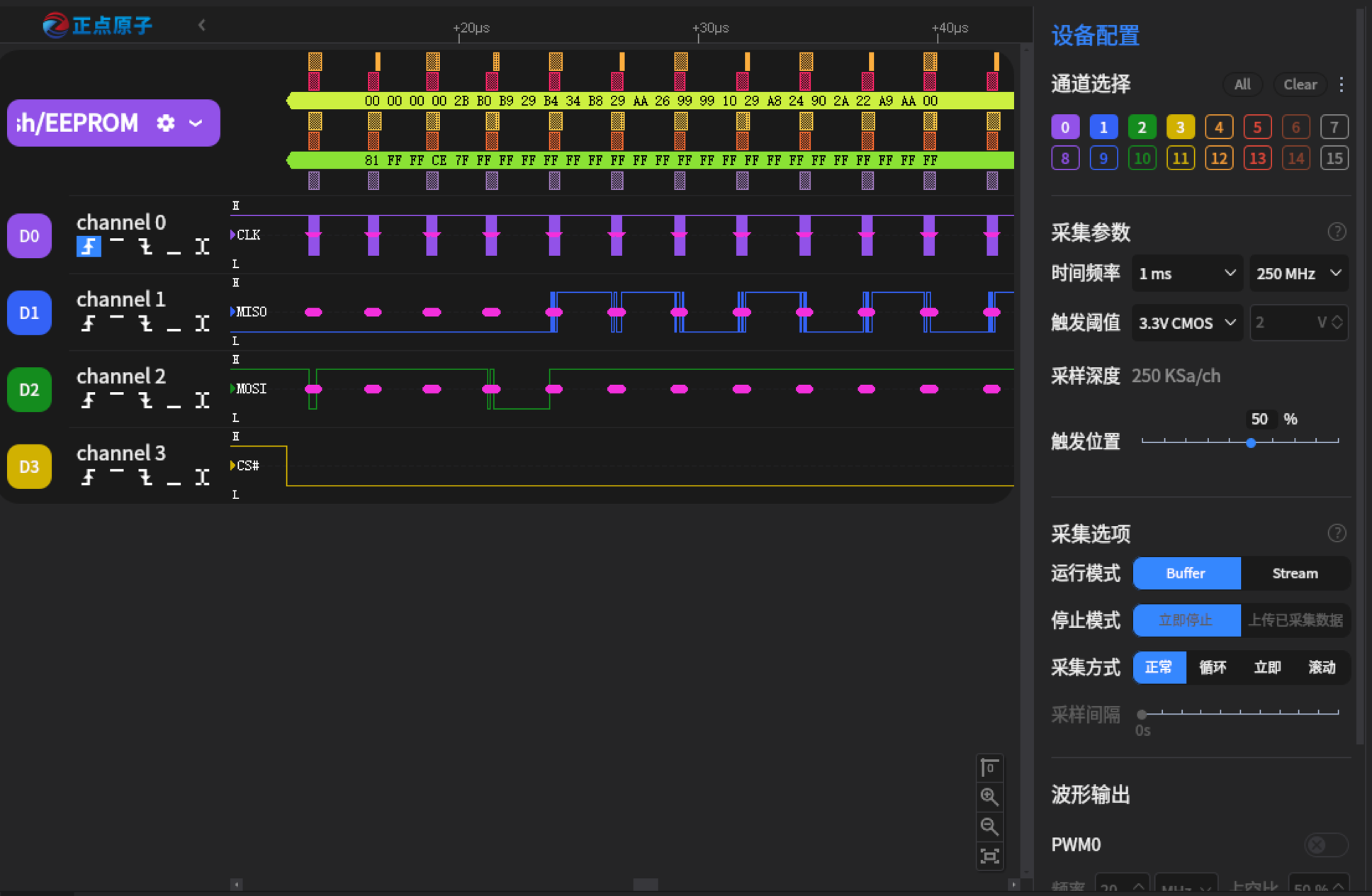
**4.3 One-Wire总线的时序分析**

对于One-Wire总线驱动的DS18B20温度传感器系统，我们使用外置的逻辑分析仪对总线上的信号进行抓取，本次实验采用的是Alientek公司的DL16 PLUS逻辑分析仪，逻辑分析仪实物图如图4.17所示。在嵌入式系统开发中，实际的硬件原型开发中会遇到一些时序错误的问题，怀疑总线受到干扰，使用逻辑分析仪测量数据总线和地址总线，利用上位机显示采集到的波形，从而可以有效排除在原型开发中遇到的问题，上位机的实例采集图如图4.18所示。



**图4.17 逻辑分析仪实物图**

**Fig 17 Appearance of logic analyzer**

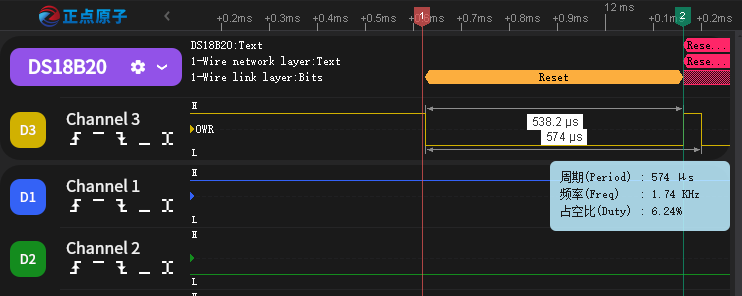


**图4.18逻辑分析仪示例采集图**

**Fig4.18 Waveform Capture from Logic Analyzer**

将逻辑分析仪的Channel3（黄色波形）与开发板上的P3.7接口进行相连。采集参数的频率配置为20MHz，采样深度设定为10Msa/ch，触发阈值选择标准为3.3V CMOS逻辑电平，使用上升沿触发进行采样。使用Stream流采集模式，确保能够抓取一段长时间的时序。

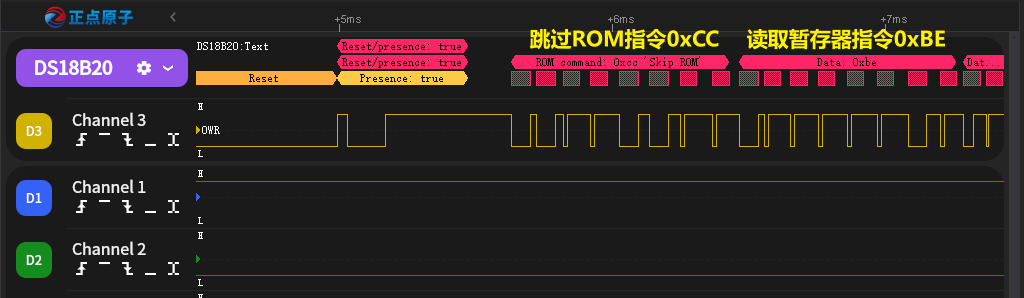
抓取DS18B20的One-Wire总线信号如下图4.19所示。信号起始于主机拉低总线产生的复位脉冲，通过逻辑分析仪的时序测量功能可以观察到该低电平维持时间约为538.2μs，满足协议中480~960μs的复位要求。



**图4.19 One-Wire总线复位信号**

**Fig4.19 One-Wire bus reset signal**

随后，总线释放后DS18B20反馈的低电平存在脉冲被成功捕获，如下图4.20所示。解码器判定为“Presence: true”状态。上位机解码还原出单片机发出的跳过ROM指令（0xCC），此命令允许总线主机读取DS18B20的8位家族码、唯一的48位序列号和8位CRC。此命令只能在总线上仅有一个DS18B20时使用。如果总线上有多个从设备，当所有从设备同时发送数据时会发生数据冲突（开漏输出将产生线与逻辑结果）。紧接着解码出读取暂存器指令（0xBE），此命令读取暂存器的内容。读取将从字节0开始，持续进行直到读取到第九个（字节8，CRC）字节。如果不需要读取所有位置，主设备可以在任何时候发出复位命令以终止读取。



**图4.20 One-Wire指令集ROM功能指令和存储器命令功能**

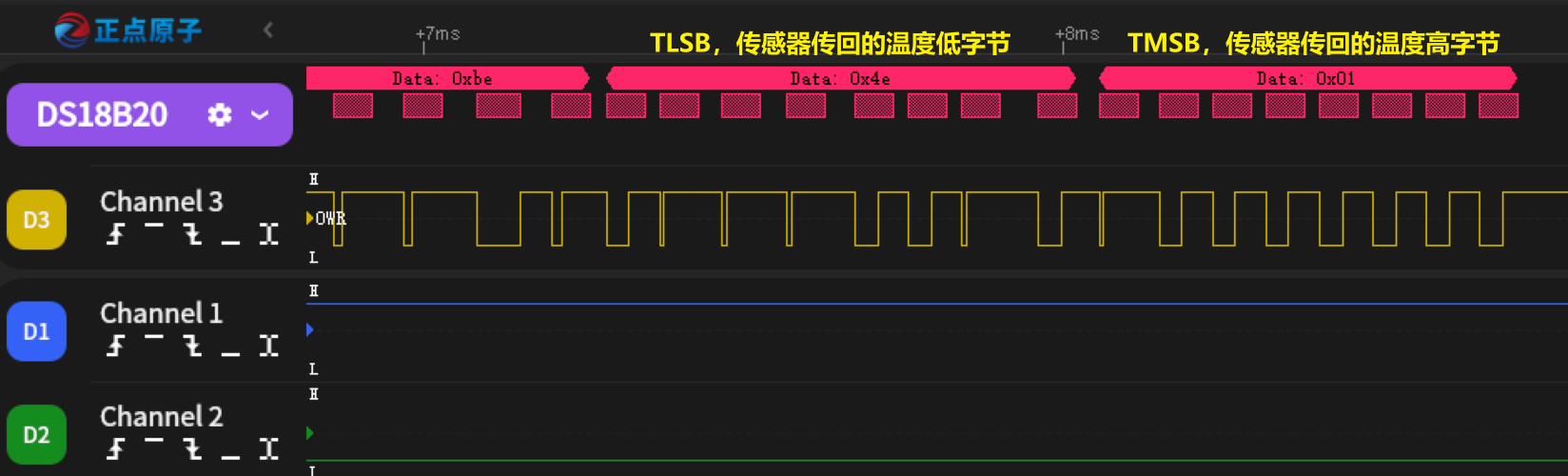
**Fig4.20 One-Wire instruction set ROM function instructions**

**and memory command functions**

随着读取完两条指令集之后，逻辑分析仪开始抓取温度的信号，如图4.21所示。高字节TMSB=0x01，低字节TLSB=0x4E。合并成16位的整数：0x014，转化为十进制数据位：334。根据DS18B20的特性，在12分辨率下，每个数值代表0.0625℃，即除以16。



计算得出此时逻辑分析仪抓取到的环境温度是20.875℃。



**图4.21 One-Wire总线温度数据信号**

**Fig 4.21 One-Wire bus temperature data signal**

**4.4 本章小节**

本章针对三个方面的内容进行介绍，分别是Proteus软件仿真、硬件实物调试以及逻辑分析仪时序分析。首先基于Proteus进行整体硬件的仿真，模拟上位机和硬件系统进行通信。在实物硬件的功能实现上，手机可以接收到单片机发送的信息，当环境的温度超过25℃是，电机风扇可以进行实现对环境的降温功能。最后我们使用逻辑分析仪抓取DS18B20温度传感器的One-Wire的总线信号。

五**、**硬件实训总结

**5.1 学习记录与问题的解决**

1.基于高通Arduino uno Q实现激光传感的摩斯密码通讯（实验时间：11.28日）

2.基于单片机实现继电器驱动外部设备（实验时间：11.29日）

3.实现IIC驱动OLED英文显示（实验时间：12月8日）

4.成功优化GB2312字库的存储容量，实现中文字体在OLED屏幕的显示（实验时间：12月9日）

注：前四项任务的程序是基于Arduino单片机开发的功能，得益于Arduino单片机高度封装的库，可以高效快速地对硬件电路进行原型开发

5.学习使用硬件电路仿真软件Proteus，数码管点灯参考教程：https://blog.csdn.net/qq\_42762607/article/details/108961205（实验时间：12月20日）

6.学习“江科大51单片机”LCD1602驱动，并上板验证。参考教程https://www.bilibili.com/video/BV1Mb411e7re?p=32&vd\_source=49808dc57cd95afa3fee40f351db7ec7（实验时间：12月20日）

7.1-Wire总线设计，完成总线的驱动（实验时间12月21日）

8.设计主体程序，添加LCD显示，上板验证（实验时间12月22日）

9.使用外置逻辑分析仪（正点原子DL16 Plus）抓取1-Wire总线的时序逻辑，并且解析1-Wire的时序是否能够和51软件的定时器代码对应。经过逻辑分析仪抓取的reset低电平信号比单片机程序中542μs要少4微秒，我估计是我使用软件定时不是很准确，但是也满足1-Wire总线设计的基本规范，通常1-Wire总线的复位信号要达到480~960μs之间，本质上不影响DS18B20传感器的正常驱动（实验时间12月22日）

10.使用Proteus搭建硬件电路进行仿真，要注意一点：P0口一定要在仿真中加上拉电阻，否则LCD连接的P0口只有数据传输，实际的LCD屏幕会没有显示。加上拉电阻之后，LCD就可以正常显示实际的温度了。（实验时间12月22日）

11.购置HC-05蓝牙芯片，调试HC-05蓝牙芯片驱动，准备为项目添加上位机蓝牙无线连接的功能（时间12月24日）

12.调试蓝牙芯片的驱动，使用串口调试助手在软件上设置蓝牙芯片的波特率，蓝牙名称。我用的是国产杰理科技的蓝牙芯片，和国外的芯片不一样，后来查芯片手册发现，一开始的蓝牙芯片的通信波特率是9600bps，当时按照网上的博客，把上位机的波特率调整是38400bps，怎么也连不上蓝牙芯片。(实验日期12月25日下午16:00）

13.我把项目迁移到了Arduino UNO R4 WIFI单片机平台上，我在Arduino IDE开发环境中可以使用Arduino标准库中的SoftwareSerial库，对蓝牙串口进行调试与开发，后续会把蓝牙功能部署到51单片机上（实验时间12月25日晚18:10）

14.SoftwareSerial库有一个好处就是可以将单片机的IO口通过软件模拟的方式，在普通的数字IO引脚上实现串行通信(UART)的功能,这样不会占用单片机的硬件UART引脚。SoftwareSerial库的资料链接https://blog.51cto.com/u\_16099353/12159501

15.上位机成功连接上了HC05蓝牙芯片，现在手机只要连接上蓝牙JCY-31-SPP，就可以在“蓝牙调试助手”中接收Arduino无线传输的数据，并且能够正常显示单片机所在的环境温度。（实验时间：12月25日晚19:06）

16.我学习Arduino SoftwareSerial标准库，尝试为51单片机制作一个模拟UART串口通讯的库。（实验时间：12月25日晚19:43）

17.我设计了一个UART的硬件端口传输的模块，用于给51单片机的P3.0和P3.1连接的蓝牙芯片进行通信。

18.我发现一个很重要的问题，51单片机在烧录的时候UART端口是不能连接其他外设的，这很容易导致51单片机烧录不上去，这是51单片机的一个通病，烧录的时候一定要先把链接在P3.0和P3.1上的蓝牙模块给拔下来，然后再进行烧录。

19.经过一番折腾，手机终于收到了蓝牙模块发送的数据，现在上位机可以清楚地显示DS18B20传感器采集到的温度。（实验时间：12月25日晚上22:00）

20.在Proteus中对蓝牙模块的传输进行仿真，导入高通Cambridge Silicon Radio的HC-05蓝牙芯片模块的lib库，lib库下载链接🔗https://www.theengineeringprojects.com/2016/03/bluetooth-library-for-proteus.html（实验时间12月25日晚上22:47）

21.在Proteus ISIS中建立虚拟终端，参考博客：https://www.theengineeringprojects.com/2013/05/how-to-use-virtual-terminal-in-proteus.html（实验时间12月25日晚上23:00）

22.在Proteus ISIS中的虚拟终端对应的波特率一定要和AT89C52单片机的波特率对应，否则会出现中文乱码。（实验时间12月25日23:13）

23.Proteus中的虚拟终端仅支持ASCII码，不支持UTF-8，摄氏度符号显示不正确，再询问AI大模型之后，我将软件代码中的UTF8编码改为GBK编码之后，虚拟终端显示正常。（实验时间12月26日14:10）

24.使用外置逻辑分析仪（正点原子DL16 Plus）抓取HC05蓝牙模块的通讯信号。（实验时间：12月28日）

26.使用Arduino UNO R4 WIFI完成对意法半导体的L293D四倍高电流半桥驱动器电机的PWM驱动，准备为项目添加温控功能。（实验时间12月28日）

27.在Proteus中对电机驱动的仿真进行调试测试，电机能够正常运行，无BUG。（实验时间：12月29日中午13:00）

28.成功将代码迁移至AT89C52RC平台，对于不同的温度，单片机通过调整方波信号的占空比驱动电机风扇以实现对环境温度的控制。（实验时间12月29日中午14:00）

25.阅读美信半导体的DS18B20手册，进一步深入研究DS18B20的One-Wire总线时序。（实验时间12月28日）

29.修BUG，新增的电机驱动中的定时器Timer0影响到了温度传感器的时序控制。将One-Wire总线模块中工作时的定时器关闭即可（实验时间12月28日晚21:25）

30.调试上位机的蓝牙功能，使其能够正常显示电机此时的转速以及风扇的转速等级。之前在Proteus ISIS虚拟终端中使用的是GBK编码，到了上位机蓝牙就会显示乱码，蓝牙传送的上位机必须使用UTF-8编码。（实验时间：12月28日晚上22:01）

31.AT89C52RC的电源引脚不够使用，额外扩展了一个电源供电模块（见下方单元格图）（实验时间：12月28日晚22:10）

32.在调试硬件实物的时候，发现风扇的正面风向是吸风，风扇背面出风。直接将L293D四倍高电流半桥驱动IC的IN1和IN2导线相互换，风扇即可实现正面吹风。（实验时间：12月28日23:00）

33.撰写《硬件程序设计报告》，阅读杰理科技HC-05蓝牙手册（时间：12月28日23:16）

34.硬件部分实物演示视频已上传到Bilibili，视频地址：《无线蓝牙温度检测与散热温控系统》实物演示-哔哩哔哩】 https://b23.tv/m64e1y5

35.硬件项目C源代码、Proteus原理图和报告文档已全部上传至Github平台，网址：https://github.com/Foxywort

**5.2 收获和体会**

通过此次《程序设计》课程的学习，最让我难以忘怀的是我和队友一起做实验的日子。12月25日晚上我（赵霖霏）和队友（张宏羽、方俊杰）一起在笃南微电子实验室一起并肩作战，25号那一天晚上我们在实验室一直做实验到**傍晚11点**。当初我们选择做蓝牙通讯是一次机缘巧合，当时我们11月份打完嵌赛，看到隔壁高云组用FPGA开发逻辑分析仪的时候，使用蓝牙作为串口与手机进行通信，我们觉得非常高级，想着我们也搞一个蓝牙试一试。我们也是第一次接触无线通讯，25号那一天晚上调了好久，当手机上出现单片机发送的信息时，那种成就感是我们这种理工男无法描述的。

参考文献

[1]陈思颖,邹乐瑶,王少坤.基于ATMEGA32A单片机和HC05的蓝牙信号传输体操机器人设计[J].信息与电脑(理论版),2021,33(07):106-108.

[2]陈丽,赵宇杰,陈奎.基于Proteus的电工电子技术实验虚拟仿真设计[J].甘肃高师学报,2025,30(04):84-89.

六**、**附录

**main.c**

1. #include <REGX52.H>
2. #include "LCD1602.h"
3. #include "DS18B20.h"
4. #include "Delay.h"
5. #include "UART.h"
6. #include "Fan.h"
7. float T;
8. char Sign;
9. void main()
10. {
11. DS18B20\_ConvertT(); *// 上电先转换一次温度，防止第一次读数据错误*
12. Delay(1000); *// 等待转换完成*
13. LCD\_Init();
14. LCD\_ShowString(1, 1, "Temperature:");
15. UART\_Init();
16. UART\_SendString("Temperature:\r\n");
17. *// 初始化风扇模块*
18. Fan\_Init();
19. Fan\_SetLevel(0);
20. while (1)
21. {
22. DS18B20\_ConvertT(); *// 转换温度*
23. T = DS18B20\_ReadT(); *// 读取温度*
24. if (T < 0) *// 如果温度小于0*
25. {
26. LCD\_ShowChar(2, 1, '-');
27. Sign = '-';
28. T = -T;
29. }
30. else *// 如果温度大于等于0*
31. {
32. LCD\_ShowChar(2, 1, '+');
33. Sign = '+';
34. }
35. LCD\_ShowNum(2, 2, T, 3); *// 显示温度整数部分*
36. LCD\_ShowChar(2, 5, '.'); *// 显示小数点*
37. LCD\_ShowNum(2, 6, (unsigned long)(T \* 10000) % 10000, 4); *// 显示温度小数部分*
38. *// 显示单位：°C（HD44780度符号代码0xDF）*
39. LCD\_ShowChar(2, 10, (char)0xDF);
40. LCD\_ShowChar(2, 11, 'C');
41. UART\_SendByte(Sign);
42. {
43. unsigned int intPart = (unsigned int)T;
44. UART\_SendByte(intPart / 100 % 10 + '0');
45. UART\_SendByte(intPart / 10 % 10 + '0');
46. UART\_SendByte(intPart % 10 + '0');
47. }
48. UART\_SendByte('.');
49. {
50. unsigned int frac = (unsigned long)(T \* 10000) % 10000;
51. UART\_SendByte(frac / 1000 % 10 + '0');
52. UART\_SendByte(frac / 100 % 10 + '0');
53. UART\_SendByte(frac / 10 % 10 + '0');
54. UART\_SendByte(frac % 10 + '0');
55. }
56. UART\_SendByte(0xA1); *// GBK 编码的高八位*
57. UART\_SendByte(0xE3); *// GBK 编码的低八位*
58. UART\_SendByte('C');
59. *// 根据温度设置风扇档位并输出档位与占空比百分比*
60. {
61. unsigned char level;
62. if (T < 25.0)
63. {
64. level = 0;
65. }
66. else if (T < 27.0)
67. {
68. level = 1;
69. }
70. else if (T < 30.0)
71. {
72. level = 2;
73. }
74. else
75. {
76. level = 3;
77. }
78. Fan\_SetLevel(level);
79. UART\_SendString(" Level:");
80. UART\_SendByte(level + '0');
81. UART\_SendString(" Speed:");
82. {
83. unsigned int sp = Fan\_GetDutyPercent(); *// 0~100%*
84. UART\_SendByte(sp / 100 % 10 + '0');
85. UART\_SendByte(sp / 10 % 10 + '0');
86. UART\_SendByte(sp % 10 + '0');
87. }
88. UART\_SendByte('%');
89. }
90. UART\_SendByte('\r');
91. UART\_SendByte('\n');
92. Delay(1000); *// 每次发送后延迟1秒*
93. }
94. }

**OneWire.h**

1. #ifndef \_\_ONEWIRE\_H\_\_
2. #define \_\_ONEWIRE\_H\_\_
3. unsigned char OneWire\_Init(void);
4. void OneWire\_SendBit(unsigned char Bit);
5. unsigned char OneWire\_ReceiveBit(void);
6. void OneWire\_SendByte(unsigned char Byte);
7. unsigned char OneWire\_ReceiveByte(void);
8. #endif

**OneWire.c**

1. #include <REGX52.H>
2. *//引脚定义*
3. sbit OneWire\_DQ=P3^7;
4. */\*\**
5. \* @brief 单总线初始化
6. \* @param 无
7. \* @retval 从机响应位，0为响应，1为未响应
8. \*/
9. unsigned char OneWire\_Init(void)
10. {
11. unsigned char i;
12. unsigned char AckBit;
13. OneWire\_DQ=1;
14. OneWire\_DQ=0;
15. i = 247;while (--i); *//Delay 500us 总线拉低至少480us*
16. OneWire\_DQ=1; *// 然后释放总线*
17. i = 32;while (--i); *//Delay 70us 在60~240us内从机会拉低总线表示响应*
18. AckBit=OneWire\_DQ;
19. i = 247;while (--i); *//Delay 500us*
20. return AckBit;
21. }
22. */\*\**
23. \* @brief 单总线发送一位
24. \* @param Bit 要发送的位
25. \* @retval 无
26. \*/
27. void OneWire\_SendBit(unsigned char Bit)
28. {
29. unsigned char i;
30. OneWire\_DQ=0;
31. i = 4;while (--i); *//Delay 10us*
32. OneWire\_DQ=Bit;
33. i = 24;while (--i); *//Delay 50us*
34. OneWire\_DQ=1;
35. }
36. */\*\**
37. \* @brief 单总线接收一位
38. \* @param 无
39. \* @retval 读取的位
40. \*/
41. unsigned char OneWire\_ReceiveBit(void)
42. {
43. unsigned char i;
44. unsigned char Bit;
45. OneWire\_DQ=0;
46. i = 2;
47. while (--i); *//Delay 5us*
48. OneWire\_DQ=1;
49. i = 2;
50. while (--i); *//Delay 5us*
51. Bit=OneWire\_DQ;
52. i = 24;
53. while (--i); *//Delay 50us*
54. return Bit;
55. }
56. */\*\**
57. \* @brief 单总线发送一个字节
58. \* @param Byte 要发送的字节
59. \* @retval 无
60. \*/
61. void OneWire\_SendByte(unsigned char Byte)
62. {
63. unsigned char i;
64. for(i=0;i<8;i++)
65. {
66. OneWire\_SendBit(Byte&(0x01<<i));
67. }
68. }
69. */\*\**
70. \* @brief 单总线接收一个字节
71. \* @param 无
72. \* @retval 接收的一个字节
73. \*/
74. unsigned char OneWire\_ReceiveByte(void)
75. {
76. unsigned char i;
77. unsigned char Byte=0x00;
78. for(i=0;i<8;i++)
79. {
80. if(OneWire\_ReceiveBit()){Byte|=(0x01<<i);}
81. }
82. return Byte;
83. }

**Uart.h**

1. #ifndef \_\_UART\_H\_\_
2. #define \_\_UART\_H\_\_
3. void UART\_Init(void);
4. void UART\_SendByte(unsigned char Byte);
5. void UART\_SendString(char \*String);
6. #endif

**Uart.c**

1. #include <REGX52.H>
2. #include "UART.h"
3. *// 蓝牙芯片的RXD连接 51单片机 的P3.1(TXD)*
4. *// 蓝牙芯片的TXD连接 51单片机 的P3.0(RXD)*
5. #define FOSC 12000000UL
6. #define UART\_BAUD 9600
7. #define UART\_SMOD 0
8. #if UART\_SMOD==0
9. #define TH1\_RELOAD (256 - (FOSC/12/32/UART\_BAUD))
10. #else
11. #define TH1\_RELOAD (256 - (FOSC/12/16/UART\_BAUD))
12. #endif
13. void UART\_Init(void)
14. {
15. *// 串口模式1，允许接收*
16. SCON = 0x50; *// SM0=0, SM1=1, REN=1*
17. *// 设置 SMOD（波特率倍速）*
18. if (UART\_SMOD) {
19. PCON |= 0x80; *// SMOD = 1*
20. } else {
21. PCON &= ~0x80; *// SMOD = 0*
22. }
23. *// Timer1 方式2（8位自动重装）作为波特率发生器*
24. TMOD = (TMOD & 0x0F) | 0x20; *// 保留Timer0，设置 Timer1 为方式2*
25. TH1 = (unsigned char)TH1\_RELOAD;
26. TL1 = (unsigned char)TH1\_RELOAD;
27. TR1 = 1; *// 启动 Timer1*
28. }
29. void UART\_SendByte(unsigned char Byte)
30. {
31. SBUF = Byte;
32. while(!TI);
33. TI = 0;
34. }
35. void UART\_SendString(char \*String)
36. {
37. unsigned int i = 0;
38. while(String[i] != '\0')
39. {
40. UART\_SendByte((unsigned char)String[i]);
41. i++;
42. }
43. }

**LCD1602.h**

1. #ifndef \_\_LCD1602\_H\_\_
2. #define \_\_LCD1602\_H\_\_
3. *//用户调用函数：*
4. void LCD\_Init();
5. void LCD\_ShowChar(unsigned char Line,unsigned char Column,char Char);
6. void LCD\_ShowString(unsigned char Line,unsigned char Column,char \*String);
7. void LCD\_ShowNum(unsigned char Line,unsigned char Column,unsigned int Number,unsigned char Length);
8. void LCD\_ShowSignedNum(unsigned char Line,unsigned char Column,int Number,unsigned char Length);
9. void LCD\_ShowHexNum(unsigned char Line,unsigned char Column,unsigned int Number,unsigned char Length);
10. void LCD\_ShowBinNum(unsigned char Line,unsigned char Column,unsigned int Number,unsigned char Length);
11. #endif

**LCD1602.c**

1. #include <REGX52.H>
2. *//引脚配置：*
3. sbit LCD\_RS=P2^6;
4. sbit LCD\_RW=P2^5;
5. sbit LCD\_EN=P2^7;
6. #define LCD\_DataPort P0
7. *//函数定义：*
8. */\*\**
9. \* @brief LCD1602延时函数，12MHz调用可延时1ms
10. \* @param 无
11. \* @retval 无
12. \*/
13. void LCD\_Delay()
14. {
15. unsigned char i, j;
16. i = 2;
17. j = 239;
18. do
19. {
20. while (--j);
21. } while (--i);
22. }
23. */\*\**
24. \* @brief LCD1602写命令
25. \* @param Command 要写入的命令
26. \* @retval 无
27. \*/
28. void LCD\_WriteCommand(unsigned char Command)
29. {
30. LCD\_RS=0;
31. LCD\_RW=0;
32. LCD\_DataPort=Command;
33. LCD\_EN=1;
34. LCD\_Delay();
35. LCD\_EN=0;
36. LCD\_Delay();
37. }
38. */\*\**
39. \* @brief LCD1602写数据
40. \* @param Data 要写入的数据
41. \* @retval 无
42. \*/
43. void LCD\_WriteData(unsigned char Data)
44. {
45. LCD\_RS=1;
46. LCD\_RW=0;
47. LCD\_DataPort=Data;
48. LCD\_EN=1;
49. LCD\_Delay();
50. LCD\_EN=0;
51. LCD\_Delay();
52. }
53. */\*\**
54. \* @brief LCD1602设置光标位置
55. \* @param Line 行位置，范围：1~2
56. \* @param Column 列位置，范围：1~16
57. \* @retval 无
58. \*/
59. void LCD\_SetCursor(unsigned char Line,unsigned char Column)
60. {
61. if(Line==1)
62. {
63. LCD\_WriteCommand(0x80|(Column-1));
64. }
65. else if(Line==2)
66. {
67. LCD\_WriteCommand(0x80|(Column-1+0x40));
68. }
69. }
70. */\*\**
71. \* @brief LCD1602初始化函数
72. \* @param 无
73. \* @retval 无
74. \*/
75. void LCD\_Init()
76. {
77. LCD\_WriteCommand(0x38);*//八位数据接口，两行显示，5\*7点阵*
78. LCD\_WriteCommand(0x0c);*//显示开，光标关，闪烁关*
79. LCD\_WriteCommand(0x06);*//数据读写操作后，光标自动加一，画面不动*
80. LCD\_WriteCommand(0x01);*//光标复位，清屏*
81. }
82. */\*\**
83. \* @brief 在LCD1602指定位置上显示一个字符
84. \* @param Line 行位置，范围：1~2
85. \* @param Column 列位置，范围：1~16
86. \* @param Char 要显示的字符
87. \* @retval 无
88. \*/
89. void LCD\_ShowChar(unsigned char Line,unsigned char Column,char Char)
90. {
91. LCD\_SetCursor(Line,Column);
92. LCD\_WriteData(Char);
93. }
94. */\*\**
95. \* @brief 在LCD1602指定位置开始显示所给字符串
96. \* @param Line 起始行位置，范围：1~2
97. \* @param Column 起始列位置，范围：1~16
98. \* @param String 要显示的字符串
99. \* @retval 无
100. \*/
101. void LCD\_ShowString(unsigned char Line,unsigned char Column,char \*String)
102. {
103. unsigned char i;
104. LCD\_SetCursor(Line,Column);
105. for(i=0;String[i]!='\0';i++)
106. {
107. LCD\_WriteData(String[i]);
108. }
109. }
110. */\*\**
111. \* @brief 返回值=X的Y次方
112. \*/
113. int LCD\_Pow(int X,int Y)
114. {
115. unsigned char i;
116. int Result=1;
117. for(i=0;i<Y;i++)
118. {
119. Result\*=X;
120. }
121. return Result;
122. }
123. */\*\**
124. \* @brief 在LCD1602指定位置开始显示所给数字
125. \* @param Line 起始行位置，范围：1~2
126. \* @param Column 起始列位置，范围：1~16
127. \* @param Number 要显示的数字，范围：0~65535
128. \* @param Length 要显示数字的长度，范围：1~5
129. \* @retval 无
130. \*/
131. void LCD\_ShowNum(unsigned char Line,unsigned char Column,unsigned int Number,unsigned char Length)
132. {
133. unsigned char i;
134. LCD\_SetCursor(Line,Column);
135. for(i=Length;i>0;i--)
136. {
137. LCD\_WriteData(Number/LCD\_Pow(10,i-1)%10+'0');
138. }
139. }
140. */\*\**
141. \* @brief 在LCD1602指定位置开始以有符号十进制显示所给数字
142. \* @param Line 起始行位置，范围：1~2
143. \* @param Column 起始列位置，范围：1~16
144. \* @param Number 要显示的数字，范围：-32768~32767
145. \* @param Length 要显示数字的长度，范围：1~5
146. \* @retval 无
147. \*/
148. void LCD\_ShowSignedNum(unsigned char Line,unsigned char Column,int Number,unsigned char Length)
149. {
150. unsigned char i;
151. unsigned int Number1;
152. LCD\_SetCursor(Line,Column);
153. if(Number>=0)
154. {
155. LCD\_WriteData('+');
156. Number1=Number;
157. }
158. else
159. {
160. LCD\_WriteData('-');
161. Number1=-Number;
162. }
163. for(i=Length;i>0;i--)
164. {
165. LCD\_WriteData(Number1/LCD\_Pow(10,i-1)%10+'0');
166. }
167. }
168. */\*\**
169. \* @brief 在LCD1602指定位置开始以十六进制显示所给数字
170. \* @param Line 起始行位置，范围：1~2
171. \* @param Column 起始列位置，范围：1~16
172. \* @param Number 要显示的数字，范围：0~0xFFFF
173. \* @param Length 要显示数字的长度，范围：1~4
174. \* @retval 无
175. \*/
176. void LCD\_ShowHexNum(unsigned char Line,unsigned char Column,unsigned int Number,unsigned char Length)
177. {
178. unsigned char i,SingleNumber;
179. LCD\_SetCursor(Line,Column);
180. for(i=Length;i>0;i--)
181. {
182. SingleNumber=Number/LCD\_Pow(16,i-1)%16;
183. if(SingleNumber<10)
184. {
185. LCD\_WriteData(SingleNumber+'0');
186. }
187. else
188. {
189. LCD\_WriteData(SingleNumber-10+'A');
190. }
191. }
192. }
193. */\*\**
194. \* @brief 在LCD1602指定位置开始以二进制显示所给数字
195. \* @param Line 起始行位置，范围：1~2
196. \* @param Column 起始列位置，范围：1~16
197. \* @param Number 要显示的数字，范围：0~1111 1111 1111 1111
198. \* @param Length 要显示数字的长度，范围：1~16
199. \* @retval 无
200. \*/
201. void LCD\_ShowBinNum(unsigned char Line,unsigned char Column,unsigned int Number,unsigned char Length)
202. {
203. unsigned char i;
204. LCD\_SetCursor(Line,Column);
205. for(i=Length;i>0;i--)
206. {
207. LCD\_WriteData(Number/LCD\_Pow(2,i-1)%2+'0');
208. }
209. }

**DS18B20.h**

1. #ifndef \_\_DS18B20\_H\_\_
2. #define \_\_DS18B20\_H\_\_
3. void DS18B20\_ConvertT(void);
4. float DS18B20\_ReadT(void);
5. #endif

**DS18B20.c**

1. #include <REGX52.H>
2. #include "OneWire.h"
3. *//DS18B20指令*
4. #define DS18B20\_SKIP\_ROM 0xCC
5. #define DS18B20\_CONVERT\_T 0x44
6. #define DS18B20\_READ\_SCRATCHPAD 0xBE
7. */\*\**
8. \* @brief DS18B20开始温度变换
9. \* @param 无
10. \* @retval 无
11. \*/
12. void DS18B20\_ConvertT(void)
13. {
14. unsigned char ea = EA; *// 备份全局中断使能*
15. EA = 0; *// 关闭中断以保护单总线时序*
16. OneWire\_Init();
17. OneWire\_SendByte(DS18B20\_SKIP\_ROM);
18. OneWire\_SendByte(DS18B20\_CONVERT\_T);
19. EA = ea; *// 恢复原中断状态（转换过程可在后台进行）*
20. }
21. */\*\**
22. \* @brief DS18B20读取温度
23. \* @param 无
24. \* @retval 温度数值
25. \*/
26. float DS18B20\_ReadT(void)
27. {
28. unsigned char TLSB,TMSB;
29. int Temp;
30. float T;
31. {
32. unsigned char ea = EA; *// 备份并关闭中断，保证读取窗口不被打断*
33. EA = 0;
34. OneWire\_Init();
35. OneWire\_SendByte(DS18B20\_SKIP\_ROM);
36. OneWire\_SendByte(DS18B20\_READ\_SCRATCHPAD);
37. TLSB=OneWire\_ReceiveByte();
38. TMSB=OneWire\_ReceiveByte();
39. EA = ea; *// 读取完成，恢复中断*
40. }
41. Temp=(TMSB<<8)|TLSB;
42. T=Temp/16.0;
43. return T;
44. }

**Delay.h**

1. #ifndef \_\_DELAY\_H\_\_
2. #define \_\_DELAY\_H\_\_
3. void Delay(unsigned int xms);
4. #endif

**Delay.c**

1. void Delay(unsigned int xms)
2. {
3. unsigned char i, j;
4. while(xms--)
5. {
6. i = 2;
7. j = 239;
8. do
9. {
10. while (--j);
11. } while (--i);
12. }
13. }

**Fan.h**

1. #ifndef \_\_FAN\_H\_\_
2. #define \_\_FAN\_H\_\_
3. void Fan\_Init(void);
4. void Fan\_SetLevel(unsigned char level);
5. unsigned char Fan\_GetDutyPercent(void);
6. #endif

**Fan.c**

1. #include <REGX52.H>
2. #include "Fan.h"
3. sbit FAN\_IN1 = P1^0;
4. sbit FAN\_IN2 = P1^1;
5. sbit FAN\_ENA = P1^2;
6. static volatile unsigned char PWM\_Duty = 0; *// 0~100*
7. static unsigned char pwm\_counter = 0; *// 0~99*
8. void Fan\_Init(void)
9. {
10. FAN\_IN1 = 0;
11. FAN\_IN2 = 1;
12. FAN\_ENA = 0;
13. TMOD = (TMOD & 0xF0) | 0x02; *// Timer0 方式2*
14. TH0 = 156; *// ~100us tick @12MHz*
15. TL0 = 156;
16. ET0 = 1;
17. EA = 1;
18. TR0 = 1;
19. }
20. void Timer0\_Routine(void) interrupt 1 *// 表示定时器0的中断*
21. {
22. pwm\_counter++;
23. if (pwm\_counter >= 100) pwm\_counter = 0;
24. if (pwm\_counter < PWM\_Duty) {
25. FAN\_ENA = 1;
26. } else {
27. FAN\_ENA = 0;
28. }
29. }
30. void Fan\_SetLevel(unsigned char level)
31. {
32. FAN\_IN1 = 0;
33. FAN\_IN2 = 1;
34. switch (level) {
35. case 0: PWM\_Duty = 0; break; *// 0%*
36. case 1: PWM\_Duty = 70; break; *// ~180/255 ≈ 70%*
37. case 2: PWM\_Duty = 83; break; *// ~210/255 ≈ 83%*
38. case 3: PWM\_Duty = 100; break; *// 100%*
39. default: PWM\_Duty = 0; break;
40. }
41. }
42. unsigned char Fan\_GetDutyPercent(void)
43. {
44. return PWM\_Duty;
45. }