智能仓储系统的开发研究

先进计算与机器人研究所

2023 年 7 月 9 日

目录

1 第四章: I2C 通信

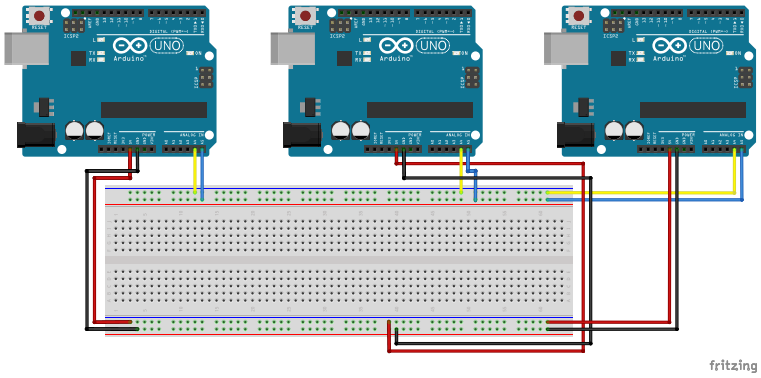

图 1: Wire 板间通信电路图

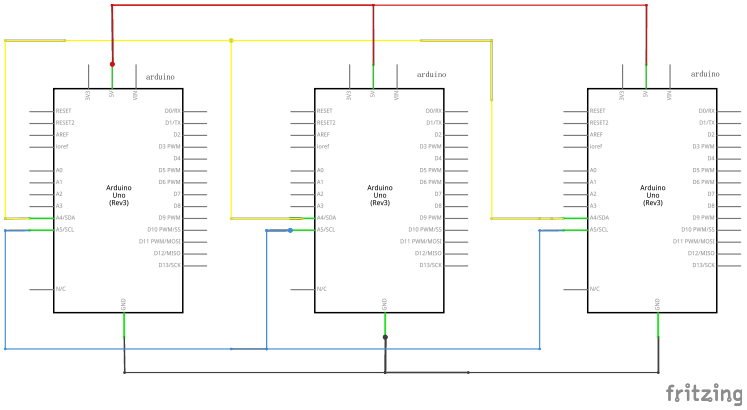

图 2: Wire 板间通讯电路原理图

- IIC 通信开始之前需要确定主从机地址。主机用 Master 类表示, 从机用 Slave 类表示, 主从机可以互相按地址发送消息。
- Message 类中会规定发送的消息格式, 以及消息发送前的打包和消息收到后的解包。

1.1 Master

Master 类的核心变量是发送的字符数组 Data, 主机的地址 address。为了便于 Slave 类继承, 就设成公共变量。

```
#include <Wire.h>
#include <Arduino.h>
#include <string.h>

class Master {
public:
  int address;
  char Data[17];
static bool Gotitflag;

  Master(int add);
  void init();
  void Set_Data(String message);
  void Send(int slave_add);
  bool isData_receivedNull();
static void receiveEvent(int howmany);
};
```

在构造函数 Master 中, 通过赋值更新地址 address。

```
Master::Master(int add){
  address=add;
};
```

由于 Wire.begin() 不能在 setup 之前运行, 所以在初始化函数 init 中初始化了 IIC 通信, 并绑定了回调函数 receiveEvent, 在 IIC 收到消息时调用。

```
void Master::init() {
  Wire.begin(address);
  Wire.onReceive(receiveEvent);
  memset(Data, '0', 16);
};
```

Set_Data 通过赋值更新 Data。

```
void Master::Set_Data(String message) {
  for (int i=0; i<message.length(); i++)
    Data[i] = message[i];
};
```

Send 是主机向地址为 slave_add 的从机发送消息 Data, 为了方便判断是否发送了, 发送成功则会在串口打印"Sended", 否则就会卡住, 不打印"Sended"。

```
void Master::Send(int slave_add) {
  Wire.beginTransmission(slave_add);
  Wire.write(Data);
  Wire.endTransmission(slave_add);
  Serial.println("Sended");
};
```

为了确认从机是否收到完整消息, 在下一节从机 Slave 类中会提到, 当 Slave 确认收到了完整消息, 就会向主机发送一个字符串"Gotit!"。那么相应地, 主机也需要判断接收到的是不是"Gotit!"。如果是则会将一个静态变量 Gotitflag 置为 1, 在演示代码 Master.ino 中, 会再次提到 Gotitflag。

```
bool Master::Gotitflag = 0;

void Master::receiveEvent(int howmany){
  String standard = "Gotit!";
  String buffer = "";
  while (Wire.available())
    buffer += (char) Wire.read();
  Serial.println(buffer);
  for (int i=0; i<6; i++)
    if (buffer[i] == standard[i])
      Gotitflag = 1;
};
```

1.1.1 Master.ino

演示代码实现的是地址为 8 的主机会一直向地址为 11 的从机发送字符串"SS0812345RR", 直到 Gotitflag 置为 1。Gotitflag 置为 1 意味着从机发送回了收到的信号。则在串口打印结束发送。

```
#include "Master.h"

Master master(8);

void setup() {
  Serial.begin(9600);
  master.init();
  master.Set_Data("SS0812345RR");
}
```

```
10
11  void loop() {
12    while (!master.Gotitflag) {
13      master.Send(11);
14      delay(5000);
15    }
16    Serial.println("endsending");
17    delay(500);
18  }
```

1.2 Slave

Slave 类的核心变量是储存主机发送过来的完整消息 Data_received, 储存主机的地址 address_to_send。Slave 储存自身的地址变量是从 Master 类中继承过来的。

```
1   #include "Master.h"
2
3   class Slave : public Master{
4   private:
5   static String Data_received;
6
7   public:
8   static int address_to_send;
9
10    Slave(int add);
11    void init();
12    void feedback();
13  static void slave_receiveEvent(int howmany);
14  };
```

构造函数 Slave() 完全继承于 Master 的构造函数

```
1   Slave::Slave(int add) : Master(add) {};
```

初始化需要重定义, 因为回调函数 slave_receiveEvent 与 Master 类的 receiveEvent 不一样。

```
1   void Slave::init(){
2     Wire.begin(address);
3     Wire.onReceive(slave_receiveEvent);
4     memset(Data, '0', 16);
5   }
```

从机的回调函数 slave_receiveEvent 需要在有消息的时候判断, 是不是"SS" 开头, 如果是, 则开始读, 并更新 Data_received。并在读到"RR" 时, 结束更新 Data_received。结束更新后, 需要读取指定位置表示主机地址的字符。这里退出 while 循环需要严格以 Wire.available() 为准, 如果是读到完整消息就直接退出, 很可能出现附录中第一节消息堵塞之陷入死循环。

```
1   void Slave::slave_receiveEvent(int howmany){
2     memset(&Data_received, 0, Data_received.length());
3     bool addressflag = 1;
4     while(Wire.available()) {
5       Serial.println("connect");
6       if (Data_received[0] != 'S' && Data_received[1] != 'S') {
7         char start1 = (char) Wire.read();
8         if (start1 == 'S') {
9           char start2 = (char) Wire.read();
10          if (start2 == 'S') {
11            Data_received += 'S';
12            Data_received += 'S';
13          }
14        }
```

```
    }

    if (Data_received[0] == 'S' && Data_received[1] == 'S') {
      if (Data_received[Data_received.length()-2]=='R' && Data_received[Data_received.
          length()-1]=='R') {
        Wire.read();
        if (addressflag) {
          String address;
          for (int i=0; i<2; i++) address += Data_received[2+i];
          address_to_send = address.toInt();
          addressflag = 0;
        }
      } else {
        char end1 = (char) Wire.read();
        Data_received += end1;
        // Serial.println(Data_received);
      }
    }
  }
  Serial.println(address_to_send);
};
```

在上一节 Master 类中有提到”Gotit!” 作为从机读到完整消息的反馈。这里 address_to_send 是先设为不可能有的地址 99, 在 slave_receiveEvent 中, 如果接收到完整消息, 则 address_to_send 会更新为有效地址, 所以利用 address_to_send 作为要不要发送”Gotit!” 的 flag。

```
void Slave::feedback() {
  if (address_to_send!=99) {
    Wire.beginTransmission(address_to_send);
    Wire.write("Gotit!");
    Wire.endTransmission(address_to_send);
    address_to_send = 99;
  }
};
```

1.2.1 Slave.ino

演示代码是实现 slave 接收到 IIC 通信传来得的消息后, 反馈”Gotit!” 给发送过来的主机。

```
#include "Slave.h"

Slave slave(11);

void setup() {
  Serial.begin(9600);
  slave.init();
}

void loop() {
  slave.feedback();
  delay(500);
}
```

1.3 Message

Message 类的核心变量有 5 个, 储存完整消息的字符数组 Data, 物品种类 Type, 发送方地址 DeviceAddress, 物品种类 Amount, 物品总质量 TotalMass。关于字符数组的长度设置原因详见本章附录。由于 unsigned long 方便转化成 char*, 而 char* 不便于计算有效长度, 故需要在转化前就通过数值确定转化后的长度, 储存在 totalmass_str_len 中。

```
#include <Arduino.h>
#include <string.h>

class Message {
private:
  char Data[17];
  char Type[3];
  int DeviceAddress, Amount;
  unsigned long TotalMass;
  int totalmass_str_len;

public:
  Message();

  void Set_Messagevalues(char *type, int device, int amount, unsigned long totalmass);
  void pack();
  void Output_pack();

  void Set_Data(char* message);
  void unpack();
  void Output_unpack();
};
```

构造函数里会对五个核心变量储存完整消息的字符数组 Data, 物品种类 Type, 发送方地址 DeviceAddress, 物品种类 Amount, 物品总质量 TotalMass 进行赋值。这里规定消息前两位和后两位是标志位。”SS” 表示消息开头, ”RR” 表示消息结尾。

```
Message::Message() {
  memset(Data, '0', 16);
  Data[0] = 'S';
  Data[1] = 'S';
  Data[14] = 'R';
  Data[15] = 'R';
  Type[0] = '0';
  Type[1] = '0';
  DeviceAddress = 0;
  Amount = 0;
  TotalMass = 0;
}
```

Set_Messagevalues 通过赋值更新物品种类 Type, 发送方地址 DeviceAddress, 物品种类 Amount, 物品总质量 TotalMass, 长度 totalmass_str_len。

```
void Message::Set_Messagevalues(char *type, int device, int amount, unsigned long
    totalmass) {
  Type[0] = type[0]; //物品种类
  Type[1] = type[1];

  DeviceAddress = device;
  Amount = amount;
  TotalMass = totalmass;
  if (totalmass<10) totalmass_str_len = 1;
  else if (totalmass<100) totalmass_str_len = 2;
  else if (totalmass<1000) totalmass_str_len = 3;
  else if (totalmass<10000) totalmass_str_len = 4;
};
```

pack 用物品种类 Type, 发送方地址 DeviceAddress, 物品种类 Amount, 物品总质量 TotalMass 更新 Data。unsigned long 不能用 String 强制转换, 但可以用 sprintf 转化成 char*。

```
void Message::pack() {
  String device_str = String(DeviceAddress);
  String amount_str = String(Amount);
```

```
  char totalmass_str[5];
  sprintf(totalmass_str, "%d", TotalMass);

  for (int i=0; i<2; i++)
    Data[2+i] = Type[i]; //物品种类

  int device_str_len = device_str.length();
  if (device_str_len == 1) {    //发送方地址
    Data[5] = device_str[0];
  }
  else if (device_str_len == 2) {
    Data[4] = device_str[0];
    Data[5] = device_str[1];
  }

  int amount_str_len = amount_str.length(); //物品数量
  if (amount_str_len == 1) Data[9] = amount_str[0];
  else if (amount_str_len == 2)
    for (int i=0; i<2; i++) Data[8+i] = amount_str[i];
  else if (amount_str_len == 3)
    for (int i=0; i<3; i++) Data[7+i] = amount_str[i];
  else if (amount_str_len == 4)
    for (int i=0; i<4; i++) Data[6+i] = amount_str[i];

  if (totalmass_str_len == 1) //物品总质量
    Data[13] = totalmass_str[0];
  else if (totalmass_str_len == 2)
    for (int i=0; i<2; i++) Data[12+i] = totalmass_str[i];
  else if (totalmass_str_len == 3)
    for (int i=0; i<3; i++) Data[11+i] = totalmass_str[i];
  else if (totalmass_str_len == 4)
    for (int i=0; i<4; i++) Data[10+i] = totalmass_str[i];
};
```

Output_pack 将打包后的 Data 打印在串口。

```
void Message::Output_pack() {
  Serial.print("Data:");
  Serial.println(Data);
};
```

Set_Data 通过赋值更新 Data。

```
void Message::Set_Data(char* message) {
  for (int i=2; i<14; i++)
    Data[i]=message[i];
}
```

pack 用 Data 更新物品种类 Type, 发送方地址 DeviceAddress, 物品种类 Amount, 物品总质量 TotalMass。中间用字符串 device_str, amount_str, tm_str 过渡，然后用 toInt 转化成数值。

```
void Message::unpack() {
  String device_str, amount_str;
  char totalmass_str[5];

  for (int i=0; i<2; i++)
    Type[i] = Data[2+i];

  for (int i=0; i<2; i++)
    device_str += Data[4+i];
  DeviceAddress = device_str.toInt();

  for (int i=0; i<4; i++)
```

```
    amount_str += Data[6+i];
  Amount = amount_str.toInt();

  String tm_str;
  for (int i=10; i<14; i++)
    tm_str += Data[i];
  TotalMass = tm_str.toInt();
};
```

Output_unpack 将打包后的物品种类 Type, 发送方地址 DeviceAddress, 物品种类 Amount, 物品总质量 TotalMass 打印在串口。

```
void Message::Output_unpack() {
  Serial.print("Type:");
  Serial.println(Type);
  Serial.print("DeviceAddress:");
  Serial.println(DeviceAddress);
  Serial.print("Amount:");
  Serial.println(Amount);
  Serial.print("TotalMass:");
  Serial.println(TotalMass);
};
```

1.3.1 Message.ino

演示代码 1 是实现消息发送前的打包:

```
#include "Message.h"

Message message;

char type[2] = {'0', '3'}; //unpack
int address = 8;
int num = 321;
unsigned long totalmass=19;

void setup() {
  Serial.begin(9600);
  message.Set_Messagevalues(type, address, num, totalmass); //pack
  message.pack();
}

void loop() {
  message.Output_pack(); //pack
  delay(500);
}
```

演示代码 2 是实现消息收到后的解包:

```
#include "Message.h"

Message message;

char data[17] = {'S','S','0','4','0','9','0','5','0','5','2','0','0','0','R','R'}; //
    pack

void setup() {
  Serial.begin(9600);
  message.Set_Data(data); //unpack
  message.unpack();
}
```

```
12
13  void loop() {
14    message.Output_unpack(); //unpack
15    delay(500);
16  }
```

1.4 附录

1.4.1 IIC 通信之消息堵塞

遇到过的消息堵塞有两种情况。

- 接发消息一方陷于死循环;
- 主从机同时用 Wire.beginTransmis() 向对方发消息。

比如, 主机发消息是一个字符一个字符发的, 而从机是通过 Wire.read() 一个字符一个字符读的。如果从机一直判断当前读取的字符是不是所需要的字符而不用 Wire.read() 读取的话, 就会造成死循环。只有用 Wire.read() 读完了上一次发送的消息里的所有字符, 才能读到下一次发送的消息。

1.4.2 字符数组的长度

字符数组的长度这里要设置的比实际储存的长度长, 如果字符数组的长度这里要设置的比实际储存的长度一样的话, 会出现问题, 比如, Type 字符数组在 Data 字符数组的后面声明的话, 当 Type 在某一方法函数里赋值的时候, 会使得 Data 字符数组的结尾追加 Type 新赋值的字符。而当字符数组的长度这里要设置的比实际储存的长度长时, 字符数组中未赋值的部分会自动填充结尾终止符斜杠 0, 这样就不会赋值时互相影响。