第 3 章

Arduino 编程语言

Arduino 编程语言是建立在 C/C++语言基础上的，即以 C/C++语言为基础，通过把 AVR 单片机(微控制器)相关的一些寄存器参数设置等进行函数化，以利于开发者更加快速地使用，其主要使用的函数包括：数字 I/O 操作函数、模拟 I/O 操作函数、高级 I/O 操作函数、时间函数、中断函数、通信函数和数学库等多种函数。

3.1 Arduino 编程基础

Arduino 的程序结构主要包括两部分：void setup()和 void loop()。其中，前者是声明变量及接口名称(例如：int val;int ledPin＝13;)，是在程序开始时使用，初始化变量、引脚模式、调用库函数等(例如：pinMode(ledPin,OUTUPT);)。而 void loop()，是在 setup()函数之后，voidloop()程序不断地循环执行，是 Arduino 的主体。

主要使用的关键字有：if、if…else、for、switch、case、while、do…while、break、continue、return、goto。

语法符号：每条语句以分号“;”结尾，每段程序以花括号“{}”括起来。

数据类型：boolean、char、int、unsigned int、long、unsigned long、float、double、string、array、void。

常量：HIGH 或者 LOW，表示数字 I/O 口的电平，HIGH 表示高电平(1)，LOW 表示低电平(0)。INPUT 或者 OUTPUT，表示数字 I/O 口的方向，INPUT 表示输入(高阻态)，OUTPUT 表示输出(AVR 能提供 5V 电压，40mA 电流)。TRUE 或者 FALSE，TRUE 表示真(1)，FALSE 表示假(0)。

1. setup()

当程序开始运行时，函数调用一次。用于在循环 loop()开始执行之前定义初始环境属性，如引脚模式(INPUT 或 OUTPUT)、启动串行端口等。在 setup()中声明的变量在 loop()中是不可访问的。语法规则为 void setup() {}，举例如下：

```
void setup()
{
```

```
pinMode(8, OUTPUT);
Serial.begin(9600);
}

void loop()
{
Serial.print('.');
delay(1000);
}
```

2. loop()

连续执行包含在其块内的代码行，直到程序停止。loop()函数与 setup()一起使用。每秒执行 loop()的次数可以用 delay()和 delayMicroseconds()函数来控制。

语法规则为 loop() {}，举例如下：

```
void setup()
{
  pinMode(WLED, OUTPUT);          //设置板载 LED 引脚为输出
}
void loop()
{
  digitalWrite(WLED, HIGH);       //设置 LED 为开
  delay(1000);                    //延迟 1s
  digitalWrite(WLED, LOW);        //设置 LED 为关
  delay(1000);                    //延迟 1s
}
```

3.2 数字 I/O 口的操作函数

数字 I/O 口的操作函数主要有 pinMode()、digitalWrite()、digitalRead()，下面分别介绍各自的用法。

1. pinMode()

pinMode(pin,mode)函数将指定的数字 I/O 引脚设置为 INPUT、OUTPUT 或 INPUT_PULLUP。可以使用 digitalWrite()和 digitalRead()方法设置或读取数字 I/O 引脚的值，它是一个无返回值函数。函数有两个参数：pin 和 mode。pin 参数表示要配置的引脚，mode 参数表示设置的参数 INPUT(输入)、OUTPUT(输出)，也可以使用 INPUT_PULLUP 模式使能内部上拉电阻。此外，INPUT 模式显式禁用内部上拉。

INPUT 参数用于读取信号，OUTPUT 用于输出控制信号。PIN 的范围是数字引脚 0～13，也可以把模拟引脚(A0～A5)作为数字引脚使用，此时编号为 14 脚对应模拟引脚 0，19 脚对应模拟引脚 5。一般会放在 setup 里，先设置再使用。

在下面的例子中，将数字引脚 3 定义为 pinIN，将数字引脚 4 定义为 pinOUT。通过方

法 pin 将 pinIN 编程为 INPUT，将 pinOUT 设置为 OUTPUT。在循环中，用数字读数读取 pinIN 的值，如果为高电平（值为 1），引脚输出设置为高电平或低电平。

```
int pinIN = 3;
int pinOUT = 4;
int value = 0;
void setup() {
  pinMode(pinIN, INPUT);
  pinMode(pinOUT, OUTPUT);
}
void loop() {
  value = digitalRead(pinIN);
  if (value == HIGH)
  {
  digitalWrite(pinOUT, HIGH);
  } else {
  digitalWrite(pinOUT, LOW);
  }
}
```

2. digitalWrite()

digitalWrite(pin，value)函数的作用是设置引脚的输出电压为高电平或低电平。该函数也是一个无返回值的函数。pin 参数表示所要设置的引脚，value 参数表示输出的电压 HIGH（高电平）或 LOW（低电平），使用前必须先用 pinMode 设置。

digitalWrite()将 HIGH 或 LOW 设置为数字输出引脚的值。如果引脚配置为 INPUT，则 digitalWrite()将使能（HIGH）或禁止（LOW）输入引脚上的内部上拉电阻。在这个例子中，它每秒更改 pinOUT 的值。

```
int pinOUT = 3;
void setup()
{
  pinMode(pinOUT, OUTPUT);
}
void loop()
{
  digitalWrite(pinOUT, HIGH);
  delay(1000);
  digitalWrite(pinOUT, LOW);
  delay(1000);
}
```

3. digitalRead()

digitalRead(pin)函数在引脚设置为输入的情况下，可以获取引脚的电压情况 HIGH（高电平）或者 LOW（低电平）。数字 I/O 口操作函数使用例程如下：

```
int button = 9;                    //设置第 9 脚为按钮输入引脚
int LED = 13;                      //设置第 13 脚为 LED 输出引脚,内部连上板上的 LED 灯
void setup()
{ pinMode(button,INPUT);           //设置为输入
  pinMode(LED,OUTPUT);             //设置为输出
}
void loop()
{ if(digitalRead(button) == LOW)   //如果读取高电平
        digitalWrite(LED,HIGH);    //13 脚输出高电平
  else
        digitalWrite(LED,LOW);     //否则输出低电平
}
```

3.3 模拟 I/O 口的操作函数

本部分主要包括 analogReference()、analogRead()、analogWrite()，以及用于 ARM 开发板的 analogReadResolution()、analogWriteResolution()函数，下面分别介绍各自的用法。

1. analogReference()

首先，analogReference()方法设置用于指定用作 analogRead()命令的参考电压的模式，该值将作为参考的最大电压。选项是：

DEFAULT：默认模拟参考电压为 5V 或 3.3V。

INTERNAL：内部参考电压，ATmega168/ATmega328 为 1.1V，ATmega8/ATmega32U4 为 2.56V。

INTERNAL1V1：内置 1.1V 参考电压。

INTERNAL2V56：内置 2.56V 参考电压。

EXTERNAL：仅作为参考，使用在 0～5V 范围内，加到 AREF 引脚的电压。

其次，函数 analogReference(ref)在 Arduino M0 和 Arduino M0PRO 上为 A/D 转换器设置参考电压，需要一个参数(ref)，参考的可能值为：

AR_DEFAULT：在这种情况下，Vref 为 VDDana，VDDana 为 3.3V，Vref 为 3.3V。

AR_INTERNAL：在这种情况下 Vref＝1V。

AR_EXTERNAL：Vref 可根据板上可用 Vref 引脚上的电压进行变化。注意在 Vref 引脚上不要超过 VDDana－0.6V(3.3V－0.6V＝2.7V)，因为 ATSAMD21G18A 不能容忍高于上述值的电压，这在其数据表中有所描述。

最后，函数 analogReference(ref)为 Arduino Primo 设置 A/D 转换器的参考电压，需要一个参数(ref)，参考的可能值为：

DEFAULT：在这种情况下，Vref 为 3.3V。

INTERNAL：在这种情况下，Vref 为 3V。

INTERNAL3V6：在这种情况下，Vref 为 3.6V。

需要注意的是，在设置模拟参考值后，使用 analogRead()读取的前几个数字可能不是精确的。另外，不要将任何 0～5V 范围内的电压应用于 AREF，如果在 AREF 引脚上使用外部引用，则必须在调用 analogRead()之前将模拟引用设置为 EXTERNAL。如果 AREF 引脚连接到外部源，则不要使用应用中的其他参考电压选项，因为它们将短路到外部电压，并导致电路板上的微控制器永久性损坏。参考例程如下：

```
int inpin = 0;
int val = 0;
void setup() {
analogReference(INTERNAL2V56);   //设置模拟参考电压为内置的 2.56V
Serial.begin(9600);
}
void loop() {
  val = analogRead(inpin);       //读取模拟端口 0 的值
  Serial.println(val);           //把数值写入串口
}
```

2. analogRead()

analogRead(pin)用于读取引脚的模拟量电压值，每读取一次需要花 100μs 的时间。参数 pin 表示所要获取模拟量电压值的引脚，返回为 int 型。精度 10 位，返回值为 0～1023，其中 0 等于 0V，1023 等于 5V。通常，单位的分辨率为 4.9mV，但可以使用 analogReference()进行更改。注意：函数参数的 pin 范围是 0～5，对应板上的模拟口 A0～A5。

```
int pinIN = 3;
int value = 0;
void setup() {
  Serial.begin(9600);
}
void loop() {
  value = analogRead(pinIN);
  Serial.printl(value);
}
```

3. analogWrite()

analogWrite(pin，value)函数是通过 PWM(Pulse-Width Modulation)，即脉冲宽度调制的方式在引脚上输出一个模拟量，图 3-1 为 PWM 输出的一般形式，也就是在一个脉冲的周期内高电平所占的比例。主要用于 LED 亮度控制、电机转速控制等方面的应用。

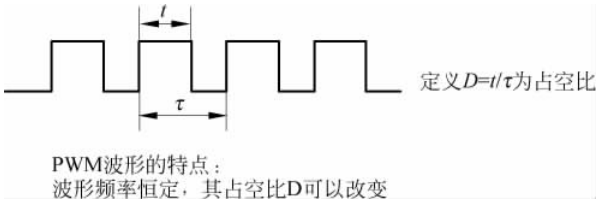

图 3-1 占空比的定义

analogWrite()方法设置 PWM 输出引脚的值。大多数引脚上的 PWM 信号的频率约为 490Hz。在 Arduino UNO 和类似的板上，引脚 5 和 6 的频率约为 980Hz。Arduino LEONARDO 的引脚 3 和 11 也工作在 980Hz。在大多数 Arduino 板卡（ATmega168 或 ATmega328）上，此功能适用于针脚 3、5、6、9、10 和 11。在 Arduino MEGA 上，它可以在 2～13 和 44～46 的引脚上工作。带有 ATmega8 的电路板只支持 9、10 和 11 引脚上的 analogWrite()，可能的值为 0～255。

注意：在 Arduino M0/Arduino M0 Pro 板上 A0 引脚有 DAC，可以通过 analogWrite()使用，选择 0～1023 的值。

模拟 I/O 口的操作函数使用例程如下：

```
int sensor = A0;                //A0 引脚读取电位器
int LED = 11;                   //第 11 引脚输出 LED
void setup()
{ Serial.begin(9600);
}
void loop()
{ int v;
  v = analogRead(sensor);
  Serial.println(v,DEC);        //可以观察读取的模拟量
  analogWrite(LED,v/4);         //读回的值范围是 0～1023 结果除以 4,才能得到 0～255 的区间值
}
```

4. analogReadResolution()

analogReadResolution(bit)是 Arduino DUE/M0/M0 Pro/Primo 的模拟 API 的扩展。设置由 analogRead()返回值（以位为单位）。它默认为 10 位（返回值为 0～1023），但是在 Arduino 中，可以将 DUE/M0/M0 Pro 设置为 12 位（可能的返回值介于 0～4095），而在 Arduino Primo 中可以设置为 14 位（可能的返回值介于 0～16384 之间）。由 analogRead()函数返回值的分辨率（以位为单位）。可以设置高于 12 的分辨率，但是由 analogRead()返回值将会近似，使用例程如下：

```
int pinIN = 3;
int value = 0;
void setup() {
  Serial.begin(9600);
}
void loop() {
  analogReadResolution(12);
  value = analogRead(pinIN);
  Serial.println(value);
}
```

5. analogWriteResolution()

analogWriteResolution(bit)是 Arduino DUE/M0/M0 Pro 和 Primo 的模拟 API 的扩展。此函数设置 analogWrite()函数的分辨率。它默认为 8 位(0～255),但是在 Arduino 中,可以将写入分辨率设置为 12,其值为 0～4095,以利用完整的 DAC 分辨率或 PWM 信号设置防止滚动。另外对于 Arduino Primo,它默认设置为 8 位,但可以将其更改为 10 位或 12 位。使用例程如下:

```
void setup(){
  Serial.begin(9600);
  pinMode(11, OUTPUT);
}
void loop(){
  int sensorValue = analogRead(A0);//使用 LED 读取输入 A0,并映射为 PWM 引脚
  Serial.print("Analog Read: ");
  Serial.print(sensorValue);
  analogWriteResolution(12);       //改变 PWM 分辨率到 12bit,只有 DUE 支持全分辨率
  analogWrite(11, map(sensorValue, 0, 1023, 0, 4095));
  Serial.print(" , 12 - bit PWM value : ");
  Serial.print(map(sensorValue, 0, 1023, 0, 4095));
}
```

3.4 高级 I/O 操作函数

本部分主要介绍 tone()、noTone()、shiftOut()、shiftIn()、pulseIn()高级操作函数,下面一一介绍其使用方法。

1. tone()

tone(pin,frequency,time)函数用于在一个引脚上产生一定时间的确定频率。该功能可以一次表达一个音调。如果一个音调在不同的引脚上播放,则新的音调功能将不起作用。如果音调在相同的引脚上播放,则调用设置其频率。

pin 参数为 int 类型,用来设置输出音频的引脚,frequency 参数为 unsigned int 类型,用来设置基音频率,time 参数为 unsigned long,用来设置时间长度。下列音调频率被预定义使用音调:

```
NOTE_B0 = 31, NOTE_C1 = 33, NOTE_CS1 = 35, NOTE_D1 = 37, NOTE_DS1 = 39, NOTE_E1 = 41, NOTE_F1 =
44, NOTE_FS1 = 46, NOTE_G1 = 49, NOTE_GS1 = 52, NOTE_A1 = 55, NOTE_AS1 = 58, NOTE_B1 = 62, NOTE_
C2 = 65, NOTE_CS2 = 69, NOTE_D2 = 73, NOTE_DS2 = 78, NOTE_E2 = 82, NOTE_F2 = 87, NOTE_FS2 = 93,
NOTE_G2 = 98, NOTE_GS2 = 104, NOTE_A2 = 110, NOTE_AS2 = 117, NOTE_B2 = 123, NOTE_C3 = 131, NOTE_
CS3 = 139, NOTE_D3 = 147, NOTE_DS3 = 156, NOTE_E3 = 165, NOTE_F3 = 175, NOTE_FS3 = 185, NOTE_G3
 = 196, NOTE_GS3 = 208, NOTE_A3 = 220, NOTE_AS3 = 233, NOTE_B3 = 247, NOTE_C4 = 262, NOTE_CS4 =
277, NOTE_D4 = 294, NOTE_DS4 = 311, NOTE_E4 = 330, NOTE_F4 = 349, NOTE_FS4 = 370, NOTE_G4 = 392,
NOTE_GS4 = 415, NOTE_A4 = 440, NOTE_AS4 = 466, NOTE_B4 = 494, NOTE_C5 = 523, NOTE_CS5 = 554,
```

```
NOTE_D5 = 587, NOTE_DS5 = 622, NOTE_E5 = 659, NOTE_F5 = 698, NOTE_FS5 = 740, NOTE_G5 = 784, NOTE
_GS5 = 831, NOTE_A5 = 880, NOTE_AS5 = 932, NOTE_B5 = 988, NOTE_C6 = 1047, NOTE_CS6 = 1109, NOTE_
D6 = 1175, NOTE_DS6 = 1245, NOTE_E6 = 1319, NOTE_F6 = 1397, NOTE_FS6 = 1480, NOTE_G6 = 1568,
NOTE_GS6 = 1661, NOTE_A6 = 1760, NOTE_AS6 = 1865, NOTE_B6 = 1976, NOTE_C7 = 2093, NOTE_CS7 =
2217, NOTE_D7 = 2349, NOTE_DS7 = 2489, NOTE_E7 = 2637, NOTE_F7 = 2794, NOTE_FS7 = 2960, NOTE_G7
 = 3136, NOTE_GS7 = 3322, NOTE_A7 = 3520, NOTE_AS7 = 3729, NOTE_B7 = 3951, NOTE_C8 = 4186, NOTE_
CS8 = 4435, NOTE_D8 = 4699, NOTE_DS8 = 4978
```

使用例程如下：

```
void setup() {
tone(12, 432, 3000);              //产生 432Hz 音频,输出在 12 引脚,3000ms 时长
}
void loop() {
}
```

2. noTone()

noTone(Pin)函数停止在指定引脚中产生频率。Pin 参数为指定相关的引脚。使用例程如下：

```
void setup() {
pinMode(12, INPUT);               //在 12 设置开关并设为输入
tone(8, 432);                     //在 8 引脚产生 432Hz 音频,没有设置时间表示无限大
}
void loop() {
  if (digitalRead(12) == HIGH) { //如果按下开关,8 引脚的音频就会停止
  noTone(8);
  }
}
```

3. shiftOut()

shiftOut()函数将数据写入引脚，一次一位。它可以从最大或最低有效位(最左边或最右边的位)开始写入位。有如下几种使用方法：

```
shiftOut(dataPin,clockPin,bitOrder,data);
shiftOut(dataPin,clockPin,bitOrder,data,count);
shiftOut(dataPin,clockPin,bitOrder,data,count,delayTime);
```

参数介绍如下：

dataPin：int，用于发送数据的引脚；

clockPin：int，该引脚用作时钟；

bitOrder：MSBFIRST 或 LSBFIRST，要使用的位顺序，MSBFIRST 代表最高有效位(最左位)，LSBFIRST 代表较低有效位优先(最右位)；

data：byte 或 unsigned int，要发送的数据，如果没有计数指定了一个字节(8 位)计数；

count：要发送的位数(1～16)；

delayTime(可选参数)：在时钟引脚内部产生时钟脉冲的延迟(以 ms 为单位)。这对于配置移位寄存器或设备速度很有用，可以处理新的比特和数据。

使用例程如下：

```
int data = 0;                                    //数据引脚
int clock = 1;                                   //时钟引脚
int strobe = 2;                                  //锁定的引脚
byte value = 0;
void setup() {
  pinMode(data, OUTPUT);
  pinMode(clock, OUTPUT);
  pinMode(strobe, OUTPUT);
}
void loop() {
  digitalWrite(strobe, LOW);                     //移位寄存器设定
  shiftOut(data, clock, LSBFIRST, value);        //向寄存器中写入值
  digitalWrite(strobe, HIGH);
  delay(1000);
  value = value + 1;
}
```

4. shiftIn()

函数读取引脚上的数据，一次一位。它可以读取从最高或最低有效位开始的位(最左位或最右位)。语法格式如下：

```
shiftIn(dataPin,clockPin,bitOrder);
shiftIn(dataPin,clockPin,bitOrder,count);
shiftIn(dataPin,clockPin,bitOrder,count, delayTime);
```

参数介绍如下：

dataPin：int，用于发送数据的引脚；

clockPin：int，该引脚用作时钟；

bitOrder：MSBFIRST 或 LSBFIRST，要使用的位顺序，MSBFIRST 代表最高有效位(最左位)，LSBFIRST 代表较低有效位优先(最右位)；

count：要读取的比特数(1 到 16)，如果没有指定计数 8b(假定一个字节的大小)；

delayTime(可选参数)：在时钟引脚内部产生时钟脉冲的延迟(以 ms 为单位)。这对于配置移位寄存器或设备速度很有用，可以处理新的比特和数据。

使用例程如下：

```
int data = 0;                                    //数据引脚
int clock = 1;                                   //时钟引脚
int strobe = 2;                                  //锁定的引脚
byte value = 0;
void setup() {
  pinMode(data INPUT);
```

```
  pinMode(clock, OUTPUT);
  pinMode(latch, OUTPUT);                    //设定引脚为移位寄存器
  Serial.begin(9600);
}
void loop() {
  digitalWrite(latch, LOW);                  //设定移位寄存器
  value = shiftIn(data, clock, LSBFIRST, 8); //从移位寄存器读取 8b (一个字节)
  digitalWrite(latch, HIGH);
  Serial.println(value);
  delay(1000);
}
```

5. Pulseln()

Pulseln(pin,state,timeout)函数用于读取引脚脉冲的时间长度,脉冲可以是 HIGH 或者 LOW。如果是 HIGH,该函数将先等引脚变为高电平,然后开始计时,一直等到变为低电平。返回脉冲持续的时间长度,单位为 ms,如果超时没有读到时间的话,返回 0。语法格式如下:

```
pulseIn(pin, value);
pulseIn(pin, value, timeout);
```

pin:要读取脉冲的引脚编号。

value:int 类型,要读取的脉冲类型,HIGH 或 LOW。

timeout:unsigned long,可选,等待脉冲启动的微秒数,默认值为 1s。

例程说明:做一个按钮脉冲计时器,测一下按钮的持续时间,测测谁的反应快,看谁能按出最短的时间,按钮接在第 3 引脚,程序如下:

```
int button = 3;
int count;
void setup()
{
    pinMode(button,INPUT);
}
void loop()
{ count = pulseIn(button,HIGH);
    if(count!= 0)
    { Serial.println(count,DEC);
      count = 0;
    }
}
```

3.5 时间函数

本部分主要包括 delay()、delayMicrseconds()、millis()、micros(),下面一一介绍。

1. delay()

delay(ms),延时函数,参数是延时的时长,单位是 ms(毫秒)。应用延时函数的典型例

程是跑马灯的应用，使用 Arduino 开发板控制四个 LED 灯依次点亮，程序如下：

```
void setup()
{
  pinMode(6,OUTPUT);                    //定义为输出
  pinMode(7,OUTPUT);
  pinMode(8,OUTPUT);
  pinMode(9,OUTPUT);
}
void loop()
{
  int i;
  for(i=6;i<=9;i++)                     //依次循环四盏灯
  {
  digitalWrite(i,HIGH);                 //点亮 LED
  delay(1000);                          //持续 1s
  digitalWrite(i,LOW);                  //熄灭 LED
  delay(1000);                          //持续 1s
    }
}
```

2. delayMicroseconds()

delayMicroseconds(μs)，延时函数，参数是延时的时长，单位是 μs(微秒)。1ms=1000μs。该函数可以产生更短的延时。例程如下：

```
//blink 例程用于演示延迟功能
int ledPin = 13;                        //连接在 13 引脚的 LED
void setup()
{
  pinMode(ledPin, OUTPUT);              //设置数字引脚为输出
}
void loop()
{
  digitalWrite(ledPin, HIGH);           //LED 开
  delayMicroseconds(5000);              //等待
  digitalWrite(ledPin, LOW);            //LED 关
  delayMicroseconds(5000);              //等待
}
```

需要注意的是，它仅适用于 Arduino M0 和 M0 Pro。此函数停止程序执行一段时间(以 μs 表示)，作为参数给出。对于低于 20μs 的值，该功能不能被精确使用。其最大误差约为 500ns。在 20～60μs 之间，最大误差约为 400ns。超过 60μs 这个函数可以给出最大的精度。长时间停顿(例如 10 000μs 或更长)，建议使用 delay()。

3. millis()

计时函数，应用该函数，可以获取单片机通电到现在运行的时间长度，单位是 ms。系统

最长的记录时间为 9 小时 22 分，超出从 0 开始。返回值是 unsigned long 型。

该函数适合作为定时器使用，不影响单片机的其他工作。（而使用 delay 函数期间无法进行其他工作）。计时时间函数使用示例，延时 10s 后自动点亮的灯，程序如下：

```
int LED = 13;
unsigned long i,j;
void setup()
{
  pinMode(LED,OUTPUT);
  i = millis();                              //读入初始值
}
void loop()
{
   j = millis();                             //不断读入当前时间值
   if((j - i)> 10 000)                       //如果延时超过 10s,点亮 LED
   {
   digitalWrite(LED,HIGH);
   }
   else digitalWrite(LED,LOW);
}
```

4. micros()

计时函数，该函数返回开机到现在运行的微秒值。返回值是 unsigned long 类型值，70min 溢出。例程如下：显示当前的微秒值。

```
unsigned long time;
void setup()
{
Serial.begin(9600);
}
void loop()
{
Serial.print("Time: ");
time = micros();                             //读取当前的微秒值
Serial.println(time);                        //打印开机到目前运行的微秒值
delay(1000);                                 //延时 1s
}
```

例程之二，跑马灯的另一种实现方式：

```
int LED = 13;
unsigned long i,j;
void setup()
{
  pinMode(LED,OUTPUT);
  i = micros();                              //读入初始值
}
```

```
void loop()
{
j = micros();                                    //不断读入当前时间值
   if((j - i)> 1000000)                          //如果延时超过 10s,点亮 LED
      {
   digitalWrite(LED1 + k,HIGH);
       }
   else digitalWrite(LED,LOW);
}
```

3.6 中断函数

什么是中断？实际上在人们的日常生活中非常常见。中断概念如图 3-2 所示。

假如正在看书，电话铃响，于是在书上做上记号，去接电话，与对方通话；门铃响了，有人敲门，让打电话的对方稍等一下，然后去开门，并在门旁与来访者交谈，谈话结束，关好门；回到电话机旁，继续通话，接完电话后再回来从做记号的地方接着看书。

同样的道理，在单片机中也存在中断概念，如图 3-3 所示，在计算机或者单片机中中断是由于某个随机事件的发生，计算机暂停原程序的运行，转去执行另一程序（随机事件），处理完毕后又自动返回原程序继续运行的过程，也就是说高优先级的任务中断了低优先级的任务。在计算机中中断包括如下几部分：

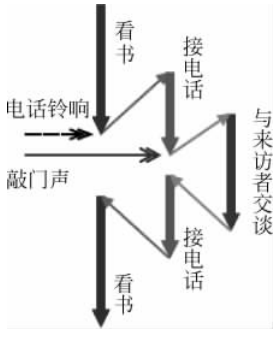

图 3-2 中断的概念

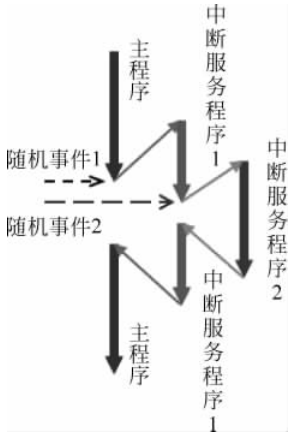

图 3-3 单片机中的中断

中断源——引起中断的原因，或能发生中断申请的来源；

主程序——计算机现行运行的程序；

中断服务子程序——处理突发事件的程序。

1. attachInterrupt()

attachInterrupt()函数用于指定外部中断发生时调用的命名中断服务程序(ISR)。大

多数 Arduino 板有两个外部中断：中断 0(引脚 2)和中断 1(引脚 3)。有一些 Arduino 板有别的中断。例如，在 Arduino Mega 2560 中有其他四个中断：中断 2(引脚 21)、中断 3(引脚 20)、中断 4(引脚 19)、中断 5(引脚 18)。在 Arduino LEONARDO 中有中断 0(引脚 3)、中断 1(引脚 2)、中断 2(引脚 0)、中断 3(引脚 1)、中断 4(引脚 7)。

Arduino DUE 板允许在所有可用引脚上附加中断功能。Arduino M0、M0 Pro 和 Zero Pro 板可以使用除 D2 以外的所有引脚作为中断引脚。attachInterrupt()函数用于设置中断，语法格式如下：

```
attachInterrupt(digitalPinToInterrupt(pin), ISR, mode);
attachInterrupt(interrupt,function, mode);
attachInterrupt(pin,ISR, mode) (Arduino DUE, ZERO, MKR1000, 101 使用);
```

interrupt：允许外部中断源。

pin：使能中断的引脚号。

ISR：中断事件发生时要调用的函数名称。此函数不能使用任何参数，并且不返回任何内容。

mode：有四种有效模式：LOW 为当引脚为低电平时触发中断；CHANGE 为当引脚改变值时设置触发中断；RISING 为引脚从低电平变为高电平时触发中断；FALLING 为当引脚从高电平变为低电平时，设置为触发中断。HIGH(仅限 Arduino DUE、ZERO、MKR1000、101 使用)，当引脚为高电平时触发中断。

中断源可选 0 或 1，对应 2 或 3 号数字引脚。中断处理函数是一段子程序，当中断发生时执行该子程序部分，例程功能如下：

数字 D2 口接按钮开关，D4 口接 LED1(红色)。D5 口接 LED2(绿色)。在例程中，LED3 为板载的 LED 灯，每秒闪烁一次。使用中断 0 来控制 LED1，中断 1 来控制 LED2。按下按钮，马上响应中断，由于中断响应速度快，LED3 不受影响，继续闪烁。使用不同的 4 个参数，例程 1 试验 LOW 和 CHANGE 参数，例程 2 试验 RISING 和 FALLING 参数。

例程 1：

```
volatile int state1 = LOW,state2 = LOW;
int LED1 = 4;
int LED2 = 5;
int LED3 = 13;                                    //使用板载的 LED 灯
void setup()
{
  pinMode(LED1,OUTPUT);
  pinMode(LED2,OUTPUT);
  pinMode(LED3,OUTPUT);
  attachInterrupt(0,LED1_Change,LOW);             //低电平触发
  attachInterrupt(1,LED2_Change,CHANGE);          //任意电平变化触发
}
void loop()
```

```
{
  digitalWrite(LED3,HIGH);
  delay(500);
  digitalWrite(LED3,LOW);
  delay(500);
}
void LED1_Change()
{
   state1 = !state1;
   digitalWrite(LED1,state1);
   delay(100);
}
void LED2_Change()
{
  state2 = !state2;
  digitalWrite(LED2,state2);
  delay(100);
}
```

例程 2：

```
volatile int state1 = LOW,state2 = LOW;
int LED1 = 4;
int LED2 = 5;
int LED3 = 13;
void setup()
{
   pinMode(LED1,OUTPUT);
   pinMode(LED2,OUTPUT);
   pinMode(LED3,OUTPUT);
   attachInterrupt(0,LED1_Change,RISING);     //电平上升沿触发
   attachInterrupt(1,LED2_Change,FALLING);    //电平下降沿触发
}
void loop()
{
   digitalWrite(LED3,HIGH);
   delay(500);
   digitalWrite(LED3,LOW);
   delay(500);
}
void LED1_Change()
{
   state1 = !state1;
   digitalWrite(LED1,state1);
   delay(100);
}
void LED2_Change()
```

```
{
   state2 = !state2;
   digitalWrite(LED2,state2);
   delay(100);
}
```

2. detachInterrupt()

detachInterrupt()函数用于取消外部中断，语法如下：detachInterrupt(interrupt)，interrupt 表示所要取消的中断源；detachInterrupt(pin)（Arduino DUE 使用），pin 表示中断的引脚号。例程如下：

```
intpinLed = 4;
volatile int state = LOW;
int count = 0;
void setup()
{
  pinMode(pinLed, OUTPUT);
  attachInterrupt(1, blink, RISING);
  Serial.begin(9600);
}
void loop()
{
  count = count + 1;
  if(count > 2000)
  {
  detachInterrupt(1);
  Serial.print("Interrupt disabled");
  }
  else{
  digitalWrite(pinLEd, state);
  }
}
void blink()
{
  state = !state;
}
```

3. interrupts()

interrupts()方法在 noInterrupt()禁用中断之后启用中断。默认情况下，启用中断以允许重要任务在后台进行。某些功能在中断被禁用时不起作用，输入的通信可能会被忽略。对于特别关键的代码段，中断可能被禁用，使用例程如下：

```
void setup() {
}
void loop() {
  noInterrupts();                              //禁用所有中断
```

```
  //重要代码部分
  interrupts();                                    //允许中断
  //其他代码
}
```

4. noInterrupts()

noInterrupts()函数的作用是禁用中断,可以使用 interrupt()重新启用它们。中断允许某些重要任务在后台发生,默认情况下启用。某些功能在中断被禁用时不起作用,输入通信可能会被忽略。中断可能会稍微中断代码的时序,但是对特别关键的代码段可能会被禁用。

相关例程如下:

```
void setup() {}
void loop()
{
  noInterrupts();
  //重要的时间敏感代码放置于此
  interrupts();
  //其他代码
}
```

3.7 串口通信函数

串行通信接口 Serial Interface 是指数据一位位地顺序传送,其特点是通信线路简单,只要一对传输线就可以实现双向通信的接口,如图 3-4 所示。

图 3-4 串口通信接口

串口通信接口出现是在 1980 年前后,数据传输率是 115～230kbps。串口通信接口出现的初期是为了实现计算机外设的通信,初期串口一般用来连接鼠标和外置 Modem 以及老式摄像头和写字板等设备。

由于串口通信接口(COM)不支持热插拔及传输速率较低,目前部分新主板和大部分便携电脑已开始取消该接口,目前串口多用于工控和测量设备以及部分通信设备中。包括各种传感器采集装置、GPS 信号采集装置、多个单片机通信系统、门禁刷卡系统的数据传输、机械手控制、操纵面板控制电机等等,特别是广泛应用于低速数据传输的工程应用。下面将串口的函数做详细的介绍。

1. Serial.begin()

该函数开启串口,通常置于 setup()函数中,用于设置串口的波特率,即数据的传输速率,每秒钟传输的符号个数。语法如下:

```
Serial.begin(speed);
```

```
Serial.begin(speed,config);
```

speed：波特率，一般取值 300、1200、2400、4800、9600、14 400、19 200、28 800、38 400、57 600、115 200；

config：设置数据位、校验位和停止位。例如，Serial.begin(speed,Serial_8N1)；Serial_8N1 中：8 表示 8 个数据位，N 表示没有校验，1 表示有 1 个停止位，例程如下：

```
void setup() {
Serial.begin(9600);                              //打开串口,设置速率为 9600 波特
}
```

2. Serial.end()

禁止串口传输函数。此时串口传输的引脚可以作为数字 I/O 脚使用。该函数没有参数，没有返回值，使用语法为 Serial.end()。

3. Serial.flush()

1.0 版本之前为清空串口缓存，现在该函数作用为等待输出数据传送完毕。如果要清空串口缓存，可以使用 while(Serial.read() >= 0)来代替。该函数没有参数，没有返回值，使用语法为 Serial.flush()。

4. Serial.print()

串口输出数据函数，写入字符串数据到串口，即该函数向串口发数据。可以发变量，也可以发字符串。语法格式为：

```
Serial.print(val);
Serial.print(val,format);
```

val：打印的值，任意数据类型；

format：输出的数据格式，包括整数类型和浮点型数据的小数点位数。

例 1：

```
Serial.print("today is good");
```

例 2：

```
Serial.print(x,DEC);以十进制发送 x
```

例 3：

```
Serial.print(x,HEX);以十六进制发送变量 x
```

5. Serial.println()

写入字符串数据，并换行，该函数与 Serial.print()类似，只是多了换行功能。语法格式为：

```
Serial.println(val)
Serial.println(val,format)
```

val：打印的值，任意数据类型；

format：输出的数据格式，包括整数类型和浮点型数据的小数点位数。

串口通信函数使用例程：

```
int x = 0;
void setup()
{ Serial.begin(9600);                          //波特率为 9600
}
void loop()
{
  if(Serial.available())
     {  x = Serial.read();
        Serial.print("received:");
        Serial.println(x,DEC);                 //输出并换行
     }
    delay(200);
}
```

6. Serial.available()

判断串口缓冲器的状态函数，用以判断数据是否送达串口。注意使用时通常用delay(100)以保证串口字符接收完毕，即保证 Serial.available()返回的是缓冲区准确的可读字节数。该函数用来判断串口是否收到数据，函数的返回值为 int 型，不带参数，使用例程如下：

```
void setup() {
  Serial.begin(9600);
  while(Serial.read()>=0){}                    //清除串口缓存
}
void loop() {
   if (Serial.available() > 0) {
    delay(100);                                //等待数据传完
    int ndata = Serial.available();
    Serial.print("Serial.available = :");
    Serial.println(ndata);
  }
  while(Serial.read()>=0){}                    //清空串口缓存
}
```

7. Serial.read()

读取串口数据，一次读一个字符，读完后删除已读数据。将串口数据读入，该函数不带参数，返回串口缓存中第一个可读字节，当没有可读数据时返回－1，整数类型，例程如下：

```
char char1;
void setup() {
  Serial.begin(9600);
  while(Serial.read()>=0){}                    //清除串口缓存
}
```

```
void loop() {                                    //从串口读数据
  while(Serial.available()>0){
    comchar = Serial.read();                     //读串口第一个字节
    Serial.print("Serial.read: ");
    Serial.println(char1);
    delay(100);
    }
  }
```

8. Serial.peek()

读串口缓存中下一字节的数据(字符型),但不从内部缓存中删除该数据。也就是说,连续的调用 peek()将返回同一个字符。而调用 read()则会返回下一个字符。该函数没有参数,返回串口缓存中下一字节(字符)的数据,如果没有,返回－1,整数类型。Serial.peek()每次从串口缓存中读取一个字符,并不会将读过的字符删除。第二次读取时仍然为同一个字符。

```
char char1;
void setup() {
  Serial.begin(9600);
  while(Serial.read()>=0){}                      //清除串口缓存
}
void loop() {                                    //读取串口数据
  while(Serial.available()>0){
    comchar = Serial.peek();
    Serial.print("Serial.peek: ");
    Serial.println(char1);
    delay(100);
    }
  }
```

9. Serial.readBytes()

从串口读取指定长度 length 的字符到缓存数组 buffer。返回存入缓存的字符数,0 表示没有有效数据。语法格式为:

Serial.readBytes(buffer,length);参数如下:

buffer:缓存变量;

length:设定的读取长度。

参考例程如下:

```
char buff[18];
int ndata=0;
void setup() {
  Serial.begin(9600);
  while(Serial.read()>=0){}                      //清串口
}
```

```
void loop() {                                    //从串口读取数据
  if(Serial.available()>0){
      delay(100);
      ndata = Serial.readBytes(buff,3);
      Serial.print("Serial.readBytes:");
      Serial.println(buffer);
    }
  while(Serial.read() >= 0){}                    //清串口缓存
  for(int i = 0; i<18; i++){
       buff[i] = '\0';
    }
}
```

10. Serial.readBytesUntil()

从串口缓存读取指定长度的字符到数组 buffer，遇到终止字符 character 后停止。返回存入缓存的字符数，0 表示没有有效数据。语法为：

```
Serial.readBytesUntil(character ,buffer,length);
```

参数为：

character：查找的字符(char)；

buffer：存储读取数据的缓存(char[]或 byte[])；

length：设定的读取长度。

相关例程如下：

```
char buff[18];
char char1 = ',';                                //终止字符
int ndata = 0;
void setup() {
  Serial.begin(9600);
  while(Serial.read()>=0){}                      //清串口
}
void loop() {                                    //从串口读数据
  if(Serial.available()>0){
      delay(100);
      ndata = Serial.readBytesUntil(char1,buff,3);
      Serial.print("Serial.readBytes:");
      Serial.println(buff);
    }
  while(Serial.read() >= 0){}                    //清串口
  for(int i = 0; i<18; i++){
       buff[i] = '\0';
    }
}
```

11. Serial.readString()

从串口缓存区读取全部数据到一个字符串型变量。语法为 Serial.readString()，没有

参数，返回从串口缓存区中读取的一个字符串。相关例程如下：

```
String cdata = "";
void setup() {
  Serial.begin(9600);
  while(Serial.read()>=0){}                    //清串口
}
void loop() {                                  //从串口读取数据
  if(Serial.available()>0){
      delay(100);
      comdata = Serial.readString();
      Serial.print("Serial.readString:");
      Serial.println(cdata);
    }
    cdata = "";
}
```

12. Serial.readStringUntil()

从串口缓存区读取字符到一个字符串型变量，直至读完或遇到某终止字符。返回从串口缓存区中读取的整个字符串，直至检测到终止字符。语法为：Serial.readStringUntil(terminator)，参数为：terminator：终止字符(char 型)。

相关例程如下：

```
String cdata = "";
char terminator = ',';
void setup() {
  Serial.begin(9600);
  while(Serial.read()>=0){}                    //清串口
}
void loop() {                                  //从串口读数据
  if(Serial.available()>0){
      delay(100);
      comdata=Serial.readStringUntil(terminator);
      Serial.print("Serial.readStringUntil: ");
      Serial.println(cdata);
    }
    while(Serial.read()>=0){}
}
```

13. Serial.parseFloat()

读串口缓存区第一个有效的浮点型数据，数字将被跳过。当读到第一个非浮点数时函数结束。没有参数，返回串口缓存区第一个有效的浮点型数据，数字将被跳过。从串口缓存中读取第一个有效的浮点数，第一个有效数字之前的负号也将被读取，独立的负号将被舍弃。语法为：

```
Serial.parseFloat();
```

相关的例程如下：

```
float cfloat;
void setup() {
  Serial.begin(9600);
  while(Serial.read()>=0){}                    //清串口
}
void loop() {                                  //从串口读数据
  if(Serial.available()>0){
      delay(100);
      comfloat = Serial.parseFloat();
      Serial.print("Serial.parseFloat:");
      Serial.println(cfloat);
    }
    while(Serial.read()>=0){}                  //清串口缓存
}
```

14. Serial.parseInt()

从串口接收数据流中读取第一个有效整数(包括负数)。需要注意的是,非数字的首字符或负号将被跳过；当可配置的超时值没有读到有效字符时,或者读不到有效整数时,分析停止；如果超时且读不到有效整数,返回0。从串口缓存中读取第一个有效整数,第一个有效数字之前的负号也将被读取,独立的负号将被舍弃。语法为：Serial.parseInt(),返回下一个有效整型值。相关例程如下：

```
int cInt;
voidsetup() {
  Serial.begin(9600);
  while(Serial.read()>= 0){}                   //清串口
}
void loop() {                                  //从串口读数据
  if(Serial.available()>0){
      delay(100);
      comInt = Serial.parseInt();
      Serial.print("Serial.parseInt:");
      Serial.println(cInt);
    }                                          //清串口缓存
    while(Serial.read()>=0){}
}
```

15. Serial.find()

从串口缓存区读取数据,寻找目标字符串,找到目标字符串返回真,否则为假,语法如下：

```
char target[] = "目标字符串";
Serial.find(target);
```

参数为：

target：目标字符串(char 型)。

相关例程如下：

```
char target[] = "test";
void setup() {
  Serial.begin(9600);
  while(Serial.read()>=0){}                    //清串口
}
void loop() {                                  //从串口读数据
   if(Serial.available()>0){
      delay(100);
      if(Serial.find(target)){
         Serial.print("find target:");
         Serial.println(target);
        }
     }                                         //清串口
    while(Serial.read()>=0){}
}
```

16. Serial.findUntil()

从串口缓存区读取数据，寻找目标字符串 target(char 型数组)，直到出现给定字符串 terminal(char 型)，找到为真，否则为假。语法为：

```
Serial.findUntil(target,terminal);
```

参数为：

target：目标字符串(char 型)；

terminal：结束搜索字符串(char 型)。

如果在找到终止字符 terminal 之前找到目标字符 target，返回真，否则返回假。相关例程如下：

```
char target[] = "test";
char terminal[] = "end";
void setup() {
  Serial.begin(9600);
  while(Serial.read()>=0){}                    //清串口
}

void loop() {                                  //从串口读数据
   if(Serial.available()>0){
      delay(100);
      if(Serial.findUntil(target,terminal)){
         Serial.print("find target:");
         Serial.println(target);
```

```
        }
      }                                          //清串口
    while(Serial.read()>=0){}
}
```

17. Serial.write()

串口输出数据函数。写二进制数据到串口。返回字节长度,语法为:

```
Serial.write(val);
Serial.write(str);
Serial.write(buf, len);
```

参数为:

val:字节;

str:一串字节;

buf:字节数组;

len:buf 的长度。

例程如下:

```
void setup(){
Serial.begin(9600);
}
void loop(){
  Serial.write(45);                          //发送 45
  int bytesSent = Serial.write("hello");     //发送"hello",返回字符串长度
}
```