StduinoWiki

2020 年 06 月 10 日

Contents

CHAPTER 1

欢迎使用 Stduino

CHAPTER 2

Stduino 简介

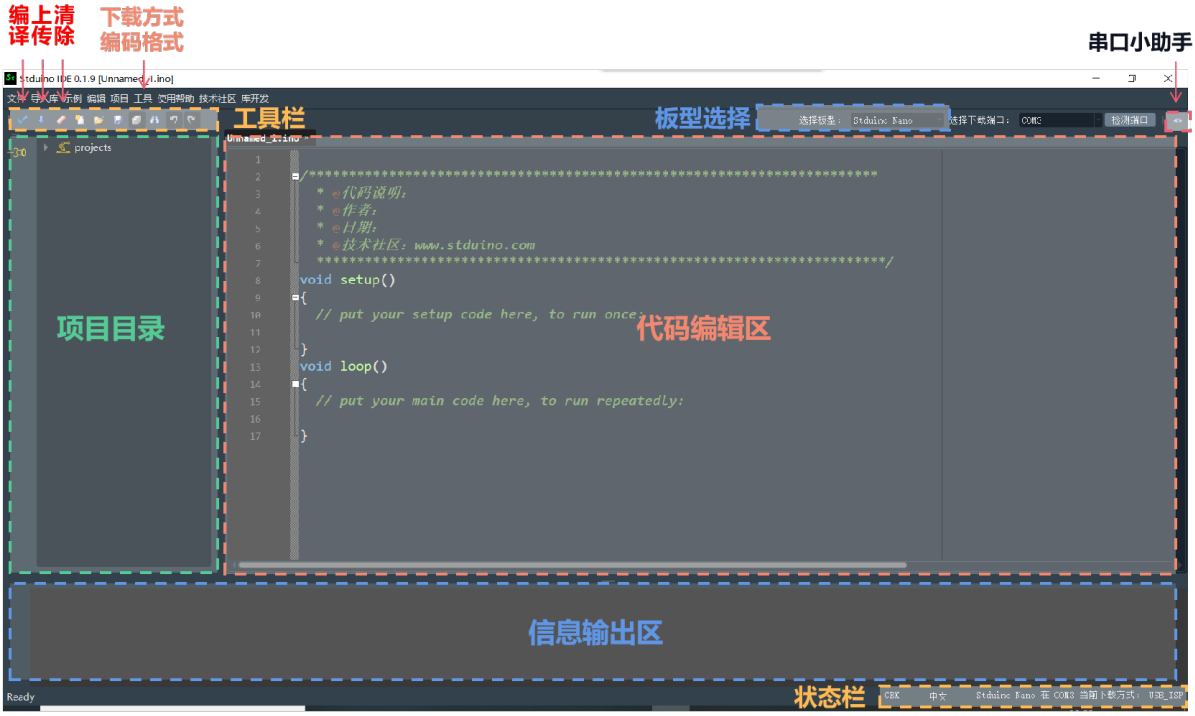
Stduino 是基于 C/C++ 封装的微控制器开发语言，它可以助您高效开发包括 STM32F103 系列在内的微控制器，为您省去大量琐碎的底层操作，从而把时间用在更重要的地方。

CHAPTER 3

软件简介

Stduino IDE 是一款针对 32 位处理器芯片的集成开发工具，当前已经完成了对 STM32F103C8T6 芯片的适配，后续将逐一对 STM32F 系列芯片进行适配；该开发工具为提高代码运行效率，核心底层库函数 80% 以上直接基于寄存器进行封装；同时兼具 Arduino 语法函数所有特点，还实现了 代码自动补全提示，中英文模式，UTF-8\GBK 编码格式，一键格式化，一键 St-link\串口下载方式等相关功能，极大地降低了 STM32F103C8T6 芯片的入门学习及后续开发成本。最后封装库将采用开源共享的理念进行分发，一处分享全球共用，极大保障后续软件维护的动力支撑。

- Win 版软件无需安装，下载解压后，双击 Stduino.exe 即可运行。
- Mac&Linux 版软件正在开发中 ~


编上清
译传除
下载方式
编码格式
串口小助手
工具栏
板型选择
项目目录
代码编辑区
信息输出区
状态栏

CHAPTER 4

教程说明

Stduino 是一个基于易用硬件和软件的原型平台 (开源)。它包括可编程的电路板（简称微控制器）和称为 Stduino IDE（集成开发环境）的现成软件组成，用于将计算机代码烧入微控制器。Stduino 目前主要支持 ST（意法半导体）32 位芯片开发。在开始教程之前，您需要提前安装 Stduino IDE。

CHAPTER 5

Stduino 快速开发教程

5.1 函数部分

5.1.1 数字 I/O 函数

pinMode()

描述

初始化指定引脚的模式。

语法

pinMode(pin,mode)

参数

pin: 指定需要设置模式的引脚

mode: 可选择以下模式设置。

输出模式	OUTPUT	推挽输出
	OUTPUT_PULSE	脉冲输出
输入模式	INPUT	浮空输入
	INPUT_PULSE	脉冲输入
	INPUT_PULLUP	上拉输入
	INPUT_PULLDW	下拉输入
	INPUT_ANALOG	模拟输入

返回

无

例程

该例程展示如何改变引脚 13 引脚状态。

```
const int LedPin = 13;//定义灯引脚号为常量，即 D13 引脚

void setup()
{
 pinMode(LedPin,OUTPUT); //设置灯引脚为推挽输出模式
}

void loop()
{
 digitalWrite(LedPin,LOW);//灯引脚输出状态更改为低电平
 delay(1000);//维持现有状态 1000ms
 digitalWrite(LedPin,HIGH);//灯引脚输出状态更改为高电平
 delay(1000);//维持现有状态 1000ms
}
```

注意

模拟引脚 (A 类口) 均能作数字引脚 (D 类口) 使用，例如 A0，A1 等。

digitalRead()

描述

读取指定引脚的电平状态。

语法

digitalRead(pin)

参数

pin: 需要读取电平状态的引脚号

返回

0 或 1

- 0：低电平
- 1：高电平

例程

该例程展示一个输入引脚控制输出引脚的状态

```
const int ButtonPin = 8;//定义按键引脚号为常量，即 D8 引脚
const int LedPin = 13;//定义灯引脚号为常量，即 D13 引脚
int ButtonState = 0;//定义按键状态为变量，初始值为 0，即低电平

void setup()
{
 pinMode(ButtonPin,INPUT);//设置 D8 引脚为浮空输入模式
 pinMode(LedPin,OUTPUT);//设置 D13 引脚为推挽输出模式
}

void loop()
{
  ButtonState = digitalRead(ButtonPin);//读取按键引脚，并赋值给按键状态

  if(ButtonState == HIGH)//若是按键状态为高电平
  {
        digitalWrite(LedPin,HIGH);//灯引脚输出状态更改为高电平
  }
  else
  {
        digitalWrite(LedPin,LOW);//灯引脚输出状态更改为低电平
```

(下页继续)

(续上页)

```
  }
}
```

注意

如果引脚输入模式设为 INPUT（浮空输入），该引脚若未接入（例如导线断连等），引脚输入易受干扰，返回值不确定（可能为高，可能为低）。

为保证输入更稳定，可设置输入模式为上拉电阻输入或者下拉电阻输入。详细信息可参考 pinMode() 中描述的各种输入模式。

模拟引脚 (A 类口) 均能作数字引脚 (D 类口) 使用，例如 A0，A1 等。

digitalWrite()

描述

设置引脚接受的数字信号

语法

digitalWrite(pin,state)

参数

pin: 要设置输出的数字信号引脚编号。

state:HIGH (1) 或者 LOW (0)

返回

无

例程

该例程展示如何改变引脚状态。

```
const int LedPin = 13;//定义灯引脚号为常量，即 D13 引脚

void setup()
```

(下页继续)

(续上页)

```
{
 pinMode(LedPin,OUTPUT); //设置灯引脚为推挽输出模式
}

void loop()
{
 digitalWrite(LedPin,LOW);//灯引脚输出状态更改为高电平
 delay(1000);//维持现有状态 1000ms
 digitalWrite(LedPin,HIGH);//灯引脚输出状态更改为低电平
 delay(1000);//维持现有状态 1000ms
}
```

注意

模拟引脚 (A 类口) 均能作数字引脚 (D 类口) 使用，例如 A0，A1 等。

访问Stduino 官网 ，了解更多 Stduino 动态

5.1.2 模拟 I/O 函数

analogRead()

描述

用于从输入引脚读取模拟信号。

语法

analogRead(Pin)

参数

pin: 被读取的模拟信号接收引脚。

返回

0~4095 之间的值

例程

该例程展示读取引脚的模拟量 (电压) 值变化，并在串口输出显示

```
const int PotentiometerPin=A0;//定义电位器引脚号为常量，即 A0 引脚
int PotentiometerValue = 0;//定义电位器读取值为变量

void setup()
{
  Serial.begin(9600); //串口 0 开启，波特率设置为 9600
  pinMode(PotentiometerPin,INPUT_ANALOG);//设置电位器引脚为模拟输入模式
}

void loop(){
  PotentiometerValue = analogRead(PotentiometerPin);//将 A0 输入信号转换为 0-4096 之间的数
值
  Serial.println(PotentiometerValue); //通过串口监视器显示电位器读取值
  delay(1000);//维持现有状态 1000ms
}
```

注意

只有如 A1、A2 这样的 A 类端口可以读取模拟信号。

A 类端口引脚可接入电位器或其他模拟量元器件，可检测输入电压为 0-3.3V。输入电压值 0~3.3V 将被映射到数值 0-4095 之间。超过 3.3V 的视为满值，例如 5V，但是不建议接入 5V 设备。

analogWrite()

描述

将模拟信号写入引脚。可用于制作呼吸灯或者以不同的速度驱动电动机。

语法

digitalWrite(pin,value)

参数

pin: 要设置的模拟信号引脚（int 类型）value: 占空比，在 0~255 之间。

返回

无

例程

该例程展示一个呼吸灯的制作。

```
int analogPin = A2;

void setup()
{
  pinMode(analogPin,OUTPUT_PULSE);//初始化
}

void loop()
{
  for(int i=0; i<4096; i++)
  {
        //for 循环语句，让亮度从 0 到 255
        analogWrite(analogPin,i);
        delay(15);//变化太快可能看不清
  }

  for(int i=4095;i>-1;i--)
  {
        //for 循环语句，让亮度从 255 到 0
        analogWrite(analogPin,i);
        delay(15);
  }
}
```

注意

初始化引脚时，需要设置为 OUT_PWM 模式。

访问Stduino 官网 ，了解更多 Stduino 动态

5.1.3 通讯函数

串口通讯函数

begin()

描述

初始化选择的串口，并设置串行数据传输速率（波特率）。

语法

Serial.begin(speed)

参数

speed 为串行数据传输速率（波特率），单位为比特每秒（bit/s）。与计算机进行通信时，常用的波特率为 9600 和 115200 两种。更多信息参考串口通讯与波特率。

返回

无

例程

以下例程通过 Serial.begin() 实现 Stduino UNO 与电脑间的串口通讯。

```
void setup()
{
  // put your setup code here, to run once:
  Serial.begin(9600);//初始化
}

void loop()
{
  // put your main code here, to run repeatedly:
  Serial.println("Hello,World");
  delay(1000);
}
```

注意

无

访问Stduino 官网 ，了解更多 Stduino 动态

print()

描述

以人们可理解的 ASCII 文本形式打印数据到串口监视窗口。

语法

Serial.print(data,format)

(该语法暂不支持)

或者

Serial.print(data)

参数

data: 可以是多种类型。

程序语句	监视窗口显示
Serial.print(78)	78
Serial.print(1.23456)	1.23
Serial.print(‘N’)	N
Serial.print(“Hello,world!”)	Hello,world!

格式：可以自己定义输出为几进制（格式）；可以是 BIN（二进制，或以 2 为基数），OCT（八进制，或以 8 为基数），DEC（十进制，或以 10 为基数），HEX（十六进制，或以 16 为基数）。对于浮点型数字，可以指定输出的小数数位。(浮点型数据，仅 float 单精度变量可设置，最多小数点后 7 位数输出。double 双精度不适用于该设置。）例如

format 值	格式种类	程序语句	监视窗口显示
BIN	二进制	Serial.print(78,BIN)	1001110
OCT	八进制	Serial.print(78,OCT)	116
DEC	十进制	Serial.print(78,DEC)	78
HEX	十六进制	Serial.print(78,HEX)	4E
阿拉伯数字	指定小数位	Serial.println(1.23456,2)	1.23

返回

返回写入的字节数

例程

以下例程通过 Serial1.print() 实现 stduino 与电脑间的串口通讯，每秒输出”Hello,world!”。

```
void setup() {

  // put your setup code here, to run once:
Serial.begin(9600);//初始化
}

void loop() {
  // put your main code here, to run repeatedly:
Serial.print("Hello,World");
        delay(1000);
}
```

注意

访问Stduino 官网 ，了解更多 Stduino 动态

println()

描述

以人们可理解的 ASCII 文本形式打印数据到串口监视窗口, 并且每次传输数据会添加一个回车符与换行符。(ASCII 13, 或‘r’及换行符 ASCII 10, 或‘n’)。

语法

Serial.println(data,format)

或者

Serial.println(data)

参数

data: 可以是多种类型。

程序语句	监视窗口显示
Serial.println(78)	78
Serial.println(1.23456)	1.23
Serial.println(‘N’)	N
Serial.println(“Hello,world!”)	Hello,world!

格式：可以自己定义输出的基数（整数数据类型）或小数位数（浮点类型）；可以是 BIN（二进制，或以 2 为基数），OCT（八进制，或以 8 为基数），DEC（十进制，或以 10 为基数），HEX（十六进制，或以 16 为基数）。对于浮点型数字，可以指定输出的小数数位。例如

format 值	格式种类	程序语句	监视窗口显示
BIN	二进制	Serial.println(78,BIN)	1001110
OCT	八进制	Serial.println(78,OCT)	116
DEC	十进制	Serial.println(78,DEC)	78
HEX	十六进制	Serial.println(78,HEX)	4E
阿拉伯数字	指定小数位	Serial.println(1.23456,2)	1.23

返回

返回写入的字节数

例程

以下例程通过 Serial1.println() 实现 stduino 与电脑间的串口通讯，每秒输出”Hello,world!”，并换行。

```
void setup() {

  // put your setup code here, to run once:
Serial.begin(9600);//初始化
}

void loop() {
  // put your main code here, to run repeatedly:
Serial.println("Hello,World");
        delay(1000);
}
```

注意

无

访问Stduino 官网 ，了解更多 Stduino 动态

read()

描述

读取传入的串口的数据。

语法

Serial1.read()

参数

为空

返回

返回传入的串口数据的第一个字节（如果没有可用的数据，则返回-1）

例程

```
int newByte = 0;   // 传入的串行数据
void setup()
{
 Serial.begin(9600);     // 打开串口，设置数据传输速率 9600
}

void loop()
{
 if (Serial.available() > 0)
{// 如果接收到数据
 newByte = Serial.read();// 读取传入的数据:

 //打印得到的数据
 Serial.print("I received: ");
```

(下页继续)

(续上页)

```
 Serial.println(newByte);
}
}
```

注意

无

访问Stduino 官网 ， 了解更多 Stduino 动态

write()

描述

写入二进制数据到串口。发送的数据以一个字节或者一系列的字节为单位。例如 Serial.write(78), 会传输 78 的二进制 01001110，机器会识别为 ASCII 码，在串口监视窗口打印 “N”（即 ASCII 码 78 对应的字符）。

语法

Serial.write(val)

或

Serial.write(str)

或

Serial.write(buf, len)

参数

val: 字节

str: 一串字节

buf: 字节数组

len: buf 的长度

返回

返回写入的字节数

例程

该例程测试 println() 和 write() 参数为 78 时的不同输出结果。

```
int byteTest = 78;

void setup(){
        Serial1.begin(9600);
}

void loop(){
        Serial.println("输出的结果为：");
        Serial.print("使用 println 函数：");
        Serial.println(byteTest);

        Serial.print("使用 write 函数：");
        Serial.write(byteTest);

        delay(1000);
}
```

注意

无

访问Stduino 官网 ， 了解更多 Stduino 动态

available()

描述

获取从串口读取有效的字节数（字符）。常用来判断串口是否接受到信号

语法

Serial.available()

参数

空

返回

可读取的字节数

例程

以下例程实现 Stduino 与电脑间串口通讯，利用 Serial.available() 判断串口是否初始化。

```
void setup()
{
  // put your setup code here, to run once:
  Serial.begin(9600);//串口初始化
}

void loop()
{
  // put your main code here, to run repeatedly:
  if(Serial.available())
  {
        //判断是否初始化
        if(Serial.read()==49)
        {//判断是否接收到了 1，49 是 1 的 ASCII 码值
          Serial.println("Hello,world!");
        }
  }
}
```

注意

测试时注意收到的是十进制还是 ascii 码。

访问Stduino 官网 ，了解更多 Stduino 动态

peek()

描述

返回传入的串行数据的下一个字节（字符）。串口接收到的数据都会暂时存放在接受缓冲区中，使用 peek() 读取时，不会移除接受缓冲区中的数据。而使用 read() 读取数据后，会将该数据从接收缓冲区移除。也就是说，连续调用 peek() 将返回相同的字符，其他与调用 read() 方法相同。

语法

Seria.peek()

参数

无

返回

返回传入的串口数据的第一个字节（String 类型）（如果没有可用的数据，则返回-1）

例程

本例程利用 peek() 方法打印接受到的信号由于 peek() 读取的时候不会清除缓存，而是直接将缓存中的数据复制一份。

```
String newChar= "";   // 传入的串行数据
void setup()
{
 Serial.begin(9600);     // 打开串口，设置数据传输速率 9600
}

void loop()
{
 if (Serial.available())
{// 如果接收到数据
 newChar = Serial.peek();// 读取传入的数据:

 //打印得到的数据
 Serial.print("I received: ");
 Serial.println(newChar);
}
}
```

注意

无

访问Stduino 官网 ，了解更多 Stduino 动态

readString()

描述

从串口缓存区读取全部数据到一个字符串型的变量

语法

Serial.readString()

参数

无

返回

字符串

例程

利用 Serial.readString() 接收串口通讯的所有数据，并打印出来

```
string serialData ="";

void setup()
{
  Serial.begin(9600);
  while(Serial.read()>= 0){}
}
void loop()
{
  if(Serial.available()>0)
  {
    delay(100);
        seriaData = Serial.readString();
        Serial.print("接收数据为:");
        Serial.println(" ");
  }
}
```

注意

无

访问Stduino 官网 ，了解更多 Stduino 动态

串口通信可能会影响个别数字引脚，这是由于串口复用了相应的引脚。不同开源板情况可参照下表:

访问Stduino 官网 ，了解更多 Stduino 动态

访问Stduino 官网 ，了解更多 Stduino 动态

5.1.4 时间函数

delay()

描述

delay() 函数用来保持某一状态。它接受单个整数，此数字表示这一状态持续的时间（以毫秒为单位）。当程序遇到这个函数时，将会等待时间过去，再继续执行下一命令。然而，delay() 函数并不是让程序等待的好方法。

语法

delay(ms)

参数

ms: 延迟的以毫秒为单位的时间

返回

无

例程

该例程展示一个闪烁灯（接 D13 引脚），为了让闪烁人眼可见，设置完 D13 引脚输入电平后，需要等待 1000 毫秒后，再执行下一语句。

```
void setup() {
        pinMode(D13,OUTPUT);//初始化 13 号引脚，控制 LED
}
```

（下页继续）

(续上页)

```
void loop() {
        digitalWrite(D13,LOW);   //LED 灯亮
        delay(1000);              //保持 LED 灯亮 1000 毫秒
        digitalWrite(D13,HIGH); //LED 灯灭
        delay(1000);             //保持 LED 灯灭 1000 毫秒
}
```

注意

无

访问Stduino 官网 ， 了解更多 Stduino 动态

millis()

描述

用来返回微处理器从开始运行到该条命令的时间（以毫秒为单位）可以一直计时 50 天左右。50 天之后则会溢出，重新归零。

语法

millis()

参数

无

返回

返回一个无符号长整形类型的值

例程

该例程利用串口通讯报告程序持续时间

unsigned int time; void setup() {

// put your setup code here, to run once: Serial.begin(9600);

}

void loop() { // put your main code here, to run repeatedly:

Serial.print((char*) "Time:"); time = millis(); Serial.println(time); //打印从程序开始到现在的时间 delay(1000); // 等待一秒钟，避免发送大量数据

}

注意

serial.print() 目前还不能接受 string 类型，需要强转为 char* 类型。

访问Stduino 官网 ，了解更多 Stduino 动态

delayMicroseconds()

描述

函数接受单个整数（或数字）参数。此数字表示时间，以微秒 (us) 为单位。一毫秒等于一千微秒，一秒等于一千毫秒。

对于超过几千微秒的延迟，应该使用 delay() 函数。实际上微妙级别的延时会有误差。

语法

delayMicroseconds(us)

参数

us: 延迟的以微秒为单位的时间

返回

无

例程

```
void setup() {
        pinMode(D13,OUTPUT);//初始化 13 号引脚，控制 LED
}
```

(下页继续)

(续上页)

```
void loop() {
        digitalWrite(D13,LOW);   //LED 灯亮
        delayMicroseconds(1000);  //保持 LED 灯亮 1000 微秒
        digitalWrite(D13,HIGH);  //LED 灯灭
        delayMicroseconds(1000);  //保持 LED 灯灭 1000 微秒
}
```

注意

无

访问Stduino 官网 ， 了解更多 Stduino 动态

micros()

描述

返回 Stduino 从运行当前程序开始的微秒数。这个数字将在约 71 分钟后溢出。

语法

micros()

参数

无

返回

返回从运行当前程序开始的微秒数（无符号整型 unsigned int）。

例程

```
unsigned int time;
void setup()
{
 Serial1.begin(9600);
}
```

(下页继续)

(续上页)

```
void loop()
{
 //Serial.print("Time:");
 time = micros();
 //打印从程序开始的时间
 Serial1.println(time);
 //等待一秒钟
 delay(1000);
}
```

注意

1 毫秒 =1000 微秒,1 秒 =1000 毫秒。

访问Stduino 官网 ，了解更多 Stduino 动态

访问Stduino 官网 ，了解更多 Stduino 动态

5.1.5 位操作函数

bit()

描述

返回指定位的值（第 0 位是 1，第 1 位是 2，第 2 位是 4，以此类推）。

语法

bit(n)

参数

n: 需要计算的位

返回

位值

例程

无

注意

无

访问Stduino 官网 ，了解更多 Stduino 动态

bitRead()

描述

读取某一个数特定位的值。

语法

bitRead(x,n)

参数

x: 想要被读取的数

n：被读取的位，0 是最低有效位（最右边）

返回

该位的值（0 或 1）。

例程

该例程展示利用串口通信，从右向左传输 11101110 的每一位的值。

```
void setup() {
        // put your setup code here, to run once:
        Serial.begin(9600);
}

void loop() {
        // put your main code here, to run repeatedly:
```

(下页继续)

(续上页)

```
        for(int i=0;i <8;i++){
        Serial.println(bitRead(0xEE,i));
        delay(1000);
        }
}
```

注意

访问Stduino 官网 ， 了解更多 Stduino 动态

bitSet()

描述

将数值的某一位设置为 1

语法

bitSet(x,n)

参数

x: 被操作的数字变量

n: 被设置为 1 的位数（右起第一位为 0 位，第二位位 1，以此类推）

返回

无

例程

无

注意

无

访问Stduino 官网 ， 了解更多 Stduino 动态

bitClear()

描述

将数值的某一位设置为 0

语法

bitClear(x,n)

参数

x: 被操作的数字变量

n: 被设置为 1 的位数（右起第一位为 0 位，第二位位 1，以此类推）

返回

无

例程

无

注意

无

访问Stduino 官网 ， 了解更多 Stduino 动态

bitWrite()

描述

修改一个数的某位上的数值。

语法

bitWrite(x,n,b)

参数

x: 想要修改的数值变量。

n：要写入的数值的位。右边为最低位，n=0。

b: 写入的数值（0 或 1）。

返回

该位的值（0 或 1）。

例程

修改 11101110（0xEE）的右边第二位的值为 0，并利用窗口通信传输改变后的值。

```
void setup() {
        // put your setup code here, to run once:
        Serial.begin(9600);
}

void loop() {
        // put your main code here, to run repeatedly:
        Serial.println(bitRead(0xEE,1));//传输没有修改前的右边第二位值
        delay(500);

        bitWrite(0xEE,1,0);//修改右边第二位的值为 0;
        Serial.println(bitRead(0xEE,1));//传输修改后的右边第二的值
        delay(500);
}
```

注意

访问Stduino 官网 ，了解更多 Stduino 动态

highByte()

描述

返回变量高字节。如果该变量有两个以上的字节，则取第二低字节（如有四个字节时，返回的是从右数第二个字节）

一般一个 16 位（双字节）的数据，比如 FF1E（16 进制）。那么高位字节就是 FF，低位是 1E，highByte() 返回 FF。

如果是 32 位（四字节）的数据，比如 3E68 C16A（16 进制）。高位字（不是字节）是 3E68，低位字是 C16A，但是利用 highByte() 返回的是 C1（第二低字节）。

语法

highByte(x)

参数

x：被取高字节的变量（可以是任何变量类型）

返回

字节（byte）

例程

利用 highByte() 返回整数 300 的高位字节

```
int intValue = 300; // 300 的 16 进制为 0x12C
void setup()
{
  Serial.begin(9600);
}
void loop()
{
  byte hiByte;
  hiByte = highByte(intValue); //取出 intValue 的高位

  Serial.println(intValue,DEC);//打引输出变量的十进制数值
  Serial.println(intValue,HEX);//打印输出变量的十六进制数值

  Serial.println(hiByte,DEC);
  delay(10000);
}
```

注意

无

访问Stduino 官网 ， 了解更多 Stduino 动态

lowByte()

描述

返回一个变量的低位（最右边）的字节。

语法

lowByte(x)

参数

x: 任何类型的值

返回

字节（Byte）

例程

利用 lowByte() 返回整数 300 的低位字节

```
int intValue = 300; // 300 的 16 进制为 0x12C
void setup()
{
  Serial.begin(9600);
}
void loop()
{
  byte loByte;
  loByte = lowByte(intValue); //取出 intValue 的高位

  Serial.println(intValue,DEC);//打引输出变量的十进制数值
  Serial.println(intValue,HEX);//打印输出变量的十六进制数值
```

(下页继续)

(续上页)

```
  Serial.println(loByte,DEC);
  delay(10000);
}
```

注意

无

访问Stduino 官网 ， 了解更多 Stduino 动态

访问Stduino 官网 ， 了解更多 Stduino 动态

5.1.6 拓展 (Advance) I/O 函数

noTone()

描述

停止由 tone() 产生的方波。如果没有使用 tone() 将不会有效果。

语法

noTone(pin)

参数

pin: 所要停止产生声音接了扬声器等元件的引脚

返回

无

例程

利用 A6 和 A7 引脚接扬声器来制作警报声。

```
void setup()
{
        pinMode(A6,OUTPUT_PULSE);
        pinMode(A7,OUTPUT_PULSE)
}

void loop()
{
        tone(A6, 440);  //A6 号引脚发声 200 毫秒
        delay(200);
        noTone(A6);   //停止 A6 号引脚发声

        tone(A7, 494); //A7 号引脚发声 500 毫秒
        delay(500);
        noTone(A7); //停止 A7 号引脚发声

        delay(300);

}
```

注意

如果你想在多个引脚上产生不同的声音，你要在对下个引脚使用 tone() 前对刚才的引脚调用 noTone().

访问Stduino 官网 ，了解更多 Stduino 动态

tone()

描述

可以产生固定频率的 PWM 信号驱动扬声器发声。发出声音的长度和声调（即频率）都可以通过参数来控制。发声持续时间需要利用 tone() 开启，noTone() 来停止发声这种方式来控制。

语法

tone(pin,frequency)

参数

pin: 接入引脚（该引脚需要连接扬声器）

frequency: 发声频率（单位：赫兹）——无符号整数型数据（unsigned int）

返回

无

例程

利用 A6 和 A7 引脚接扬声器来制作警报声。

```
void setup()
{
        pinMode(A6,OUTPUT_PULSE);
        pinMode(A7,OUTPUT_PULSE);
}

void loop()
{
 tone(A6, 440);  //A6 号引脚发声 200 毫秒
 delay(200);
 noTone(A6);   //停止 A6 号引脚发声

 tone(A7, 494); //A7 号引脚发声 500 毫秒
 delay(500);
 noTone(A7); //停止 A7 号引脚发声

 delay(300);

}
```

注意

无

访问Stduino 官网 ，了解更多 Stduino 动态

shiftIn()

描述

将一个数据的字节一位一位的移入。从哪个最高有效位（最左边）或最低有效位（最右边）开始。对于每个位，先拉高时钟电平，再从数据传输线中读取一位，再将时钟线拉低。

语法

shiftIn(dataPin,clockPin,bitOrder)

参数

dataPin: 数据引脚

clockPin: 时钟引脚

bitOrder: 移位顺序，高位先入（MSBFIRST）还是低位先入（LSBFIRST）。

返回

读取到的数据

例程

注意

如果与 Stduino 进行数据通讯的设备是在时钟引脚脉冲信号上升沿发送数据，请确保在调用 shiftIn() 前，应先通过 digitalWrite(clockPin, LOW) 语句，将时钟引脚设置为 LOW。

访问Stduino 官网 ，了解更多 Stduino 动态

shiftOut()

描述

将一个字节的数据通过移位输出方式逐位输出。数据可以从最高位（最左位）或从最低位（最右位）输出。在输出数据时，当一位数据写入数据输出引脚时，时钟引脚将输出脉冲信号，指示该位数据已被写入数据输出引脚等待读取。

Stduino 开发板的普通 IO 引脚是有限的，必要时要对 IO 口进行拓展，才能满足外部设备对 IO 口的需求。如利用 74HC595 芯片来拓展 Stduino 的 IO 口，来设计流水灯。

语法

shiftOut(dataPin,clockPin,bitOrder,val)

参数

dataPin：输出数据的引脚，引脚需配置成输出模式。clockPin：时钟引脚。bitOrder：移位顺序。有高位先出（MSBFIRST）和低位优先（LSBFIRST）两种方式。val：所要输出的数据值，该数据值将以 byte 形式输出。

返回

无

例程

Stduino 外接 74HC595 芯片，用 3 个 IO 口控制 8 路 LED 灯。

```
int latchPin = 10;//锁存引脚
int clockPin = 9; //时钟引脚
int dataPin = 8; //数据引脚

void setup ()
{
  pinMode(10, OUTPUT);//锁存引脚
  pinMode(9, OUTPUT);//时钟引脚
  pinMode(8, OUTPUT); //数据引脚
}
void loop()
{
  for (int data = 0; data < 255; data++)
  {
        digitalWrite(10, LOW); //将 ST_CP 口上加低电平让芯片准备好接收数据
        shiftOut(8, 9, LSBFIRST, data);
        digitalWrite(10, HIGH); //将 ST_CP 这个针脚恢复到高电平
        delay(1000); //延迟 1 秒钟观看显示
  }
}
```

注意

- 使用 shiftOut() 函数前，数据引脚（dataPin）和时钟引脚（clockPin）必须先通过 pinMode() 指令设置为输出（OUTPUT）模式。
- 如果读取数据的设备是在 Stduino 的时钟引脚脉冲信号上升沿读取 Stduino 的输出数据，请确保在调用 shiftOut() 输出数据前，应先通过 digitalWrite(clockPin, LOW) 语句，将时钟引脚设置为 LOW，以确保数据读取准确无误。
- shiftOut 一次只能输出 1 字节(8 位)数据。如果需要输出大于 255 的数值,需要通过两次使用 shiftOut() 输出数据。

如下程序所示：

```
// 高位字节先出模式
int data = 500; //待输出数据
shiftOut(dataPin, clock, MSBFIRST, (data >> 8)); // 输出高位字节
shiftOut(dataPin, clock, MSBFIRST, data); // 输出低位字节

// 低位字节先出模式
data = 500; //待输出数据
shiftOut(dataPin, clock, LSBFIRST, data); // 输出低位字节
shiftOut(dataPin, clock, LSBFIRST, (data >> 8)); // 输出高位字节
```

访问Stduino 官网 ，了解更多 Stduino 动态

访问Stduino 官网 ，了解更多 Stduino 动态

5.2 运算部分

5.2.1 Math

abs()

描述

计算一个数的绝对值。

语法

abs(x)

参数

x: 接受到的数值

返回

返回 x 的绝对值

取值范围	返回值
$x > 0$	x
$x = 0$	0
$x < 0$	-x

例程

```
val1 = abs(val1);//求 val1 的绝对值，从而保证 val1 的值永远不为负
```

注意

无

访问Stduino 官网 ， 了解更多 Stduino 动态

constrain()

描述

将一个数约束在一个固定范围内

语法

constrain(x,a,b)

参数

x：要被约束的数字。

a：该范围的最小值。

b：该范围的最大值。

返回

取值范围	返回值
a < x < b	x
x <= a	a
x >= b	b

例程

```
val1 = constrain(10, val1, 30);//将 val1 的值限定在 10 到 30 之间。
```

注意

map()

描述

将变量从范围 A 映射到范围 B。例如将 10 位模拟输入结果转换至 8 位模拟输出、利用模拟输入值控制舵机角度等。

语法

x = map(value,fromLow,fromHigh,toLow,toHigh)

参数

value: 需要映射的值。

fromLow/fromHigh: 映射前域的上/下限。

toLow/toHigh: 需要映射后的域的上/下限。

返回

被映射后的值

例程

将一个 10 位模拟结果转换至 8 位模拟输出。

```
void setup()
{
  pinMode(A1,Input);//设置 A1 引脚位输入引脚
  pinMode(8,OUTPUT);//设置 8 号引脚为输出引脚
}

void loop()
{
  int x = analogRead(A1); 将 A1 引脚模拟输入值存入 x
  x = map(x, 0, 1023, 0, 255); 将 x 的值从 10 位缩放到 8 位，以符合模拟输出取值范围
  analogWrite(8, x);//从 8 号引脚以 PWM 方式输出
}
```

注意

map() 函数使用整型数进行运算，因此小数的余数部分会被舍弃。

访问Stduino 官网 ，了解更多 Stduino 动态

max()

描述

比较两个数的最大值。

语法

max(x,y)

参数

x: 第一个数字（任何数据类型）

y: 第二个数字（任何数据类型）

返回

参数中较大的那一个

例程

```
int val1 =40;
val1 = max(val1, 30);//将 30 与 val1 中最大的值赋给 val1
```

注意

无

访问Stduino 官网 ， 了解更多 Stduino 动态

min()

描述

比较两个数字中的最小值

语法

min(x,y)

参数

x: 第一个数字（任何数据类型）

y: 第二个数字（任何数据类型）

返回

两个数字中的较小者

例程

```
int val1 =40;
val1 = min(val1, 30);//将 30 与 val1 中最小的值赋给 val1
```

注意

无

访问Stduino 官网 ， 了解更多 Stduino 动态

sq()

描述

求算术平方根

语法

sq(num)

参数

num: 整型或浮点类型数值（int/float）

返回

返回 float 类型

例程

```
int num0 = 16;
int num1 = 1;
num1 = sq(num0);//将 num 开平方，并将算术平方根的值赋给 num1
```

注意

访问Stduino 官网 ， 了解更多 Stduino 动态

file:///C:/Users/Astilbe/Desktop/Stduino/Stduino_Wiki/source/Computation/chart2.rst

5.3 变量部分

5.3.1 数据类型

string-char 数组

描述

文本可以用两种方式表示。您可以使用 String 数据类型，也可以从 char 类型的数组中生成字符串并以 null 终止。本页描述了后一种方法。有关 String 对象的更多详细信息，请参阅：String 对象 ，它以消耗较多的内存为代价提供了更多功能。

语法

以下节为有效的字符串。

```
char Str1[15];//声明一个五个字符大小的 char 数组
char Str2[8] = {'S', 't', 'd', 'u', 'i', 'n', 'o'};//声明一个字符数组，并为其赋值
char Str3[8] = {'S', 't', 'd', 'u', 'i', 'n', 'o', '\0'};//添加空字符
char Str4[] = "Stduino";//声明一个空数组，并赋值
char Str5[8] = "Stduino";//声明 8 个字符长度的数组，并赋值
char Str6[15] = "Stduino";
```

例程

声明字符串

```
char *myStrings[] = {
  "This is string 1", "This is string 2", "This is string 3",
  "This is string 4", "This is string 5", "This is string 6"
};

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

void loop() {
  for (int i = 0; i < 6; i++) {
        Serial.println(myStrings[i]);
        delay(500);
  }
}              // 将浮点数换保留到小数点后三位，并转化为字符串
```

注意

通常，字符串以空字符（ASCII 代码 0）结尾。这使函数（如 Serial.print()）可以判断字符串的结尾在哪里。

也就是说，字符串需要比需要容纳的文本多留一个空格，这也就是为什么 Str2 和 Str5 必须为八个字符的原因，即使" Stduino" 只有七个，最后一个位置也会自动填充一个空字符。Str4 的大小将自动设置为八个字符，其中一个用于表示多余的 null。在 Str3 中，我们自己明确包含了空字符（写为 '0'）。

访问Stduino 官网 ，了解更多 Stduino 动态

String()-类

描述

该语句用来构造 `String` 类的实例。可以利用不同的数据类型构造字符串（即将这些数据格式化为字符串）

包括：

- 用" "引起来的常量字符串，即 char 数组
- 用' '引起来的单个常量字符
- String 对象的另一个实例
- 常数整数或长整数
- 指定的基数的常数整数或长整数
- 整数或长整数变量
- 指定基数的整数或长整数变量
- 具有指定位数的浮点数或双精度浮点数（double）

语法

可以使用下列三者之一声明 String 类型变量。

- String(val)
- String(val,base)
- String(val,decimalPlaces)

参数

val: 需要转换为字符串的变量。支持转换的数据类型包括：字符串、字符、字节、整数、长整数、无符号整数、无符号长整数、浮点数、双精度数。

base:（可选）需要转换整数型的基数（即对进制的规定）。

decimalPlaces: 仅当 val 为 float 或 double 时，所需的小数位数。

返回

String 类的一个实例。

例程

声明字符串

```
String stringOne = "Hello,Stduino!";                    // 构建一个常量字符串
String stringOne = String("a");                         // 将字符转换为字符串
String stringTwo = String("This is a string");          // 将一个常量字符串转化为字符串
String stringOne = String(stringTwo + " with more");  // 将两个字符串连接成一个字符串
String stringOne = String(13);                          // 将一个常量整数型转化为字符串
String stringOne = String(analogRead(0), DEC);          // 将整数换算为十进制，并转化为字符串
String stringOne = String(45, Hex);                     // 将整数换算为八进制，并转化为字符串
String stringOne = String(255,BIN);                     // 将整数换算为二进制，并转化为字符串
String stringOne = String(millis(), DEC);               // 将长整形数换算为十六进制，并转化为
字符串
String stringOne = String(5,698,3);                     // 将浮点数换保留到小数点后三位，并转
化为字符串
```

注意

访问Stduino 官网 ，了解更多 Stduino 动态

访问Stduino 官网 ，了解更多 Stduino 动态

5.4 待补充

5.5 修改记录

2020/3/29

2020/3/26

- micros() 函数更新
- peek() 方法更新，返回的是 char* 类型，而其他人返回的是 char

- tone 以及 noTone 更新，tone 不能直接控制 duration

2020/3/22

- Serial1.print() 参数需要写成（char* “hello,world”）
- 更新了 Serial 相关的函数。
- 串口通讯索引页需要再补充更多内容

2020/3/7

- 增加了位操作的两个函数 bitRead() 和 bitWrite(); 时间函数的 delayMicrosecond() 和 millis();

2020/6/8

- analogWrite 输出应该分两种写以及明确输出量范围，是否去掉 OUTPUT_ANALOG

访问Stduino 官网 ，了解更多 Stduino 动态