

代码钢琴家

谁为了生活不变。

- CnBlogs
- Home
- New Post
- Contact
- Admin
- Rss
- XML

公告



昵称：代码钢琴家
园龄：1年3个月
粉丝：22
关注：15
[+加关注](#)

Post Categories

- Android(4)
- Arduino(6)
- C Language(10)
- C++(9)
- C++练习题记录(2)
- C51(6)
- Java Web(4)
- Java 基础(16)
- Java 集合框架(1)
- Linux学习(3)
- Mysql(1)
- 扩展知识(5)
- 树莓派(5)
- 数据结构和算法(15)
- 重拾数据结构和算法(6)

Post Archives

- 2017/4 (5)
- 2017/3 (3)
- 2017/1 (3)
- 2016/12 (1)
- 2016/11 (9)
- 2016/10 (8)
- 2016/9 (4)
- 2016/8 (1)
- 2016/7 (1)
- 2016/6 (10)
- 2016/5 (9)
- 2016/4 (6)
- 2016/3 (7)

→ 树莓派wiringPi库详解

wiringPi是一个很棒的树莓派IO控制库，使用C语言开发，提供了丰富的接口：GPIO控制，中断，多线程，等等。java 的pi4j项目也是基于wiringPi的时候整理好了会放出来的。

下面开始wiringPi之旅吧！

安装

进入 wiringPi的github (<https://git.drogon.net/?p=wiringPi;a=summary>) 下载安装包。点击页面的第一个链接的右边的snapshot,下载安装压缩包。然后进入安装包所在的目录执行以下命令：

```
>tar xzf wiringPi-98bcb20.tar.gz //98bcb20为版本标号，可能不同
>cd wiringPi-98bcb20
>./build
```

验证wiringPi的是否安装成功，输入gpio -v，会在终端中输出相关wiringPi的信息。否则安装失败。

编译 和运行

假如你写了一个LEDtest.c 的项目，则如下。

```
编译：

g++ -Wall -o LEDtest LEDtest.cpp -lwiringPi //使用C++编程，-Wall 是为了使能所有警告，以便发现程序中的问题

gcc -Wall -o LEDtest LEDtest.c -lwiringPi //使用C语言编程

运行：

sudo ./LEDtest
```

查看引脚编号表格

使用如下控制台下命令

```
> gpio readall
```

也可以查看下面的图。

注意：查看时，将树莓派的USB接口面对自己，这样看才是正确的。

2016/2 (2)

2016/1 (3)

2015/12 (10)

2015/8 (6)

2015/7 (3)

2015/6 (7)

2015/4 (1)

2015/3 (5)

2015/2 (1)

Top Posts

1. 一次性搞清楚equals和hashCode(4497)

2. python之列表切片(slice)(3611)

3. 在Ubuntu上安装网易云音乐(2353)

4. 如何编写自己的Arduino库 ? (2121)

5. 【C51】单片机芯片之——图解74HC595(1588)

推荐排行榜

1. 树莓派wiringPi库详解(4)

2. C++头文件，预处理理解(4)

3. 【C51】单片机定时器介绍(3)

4. java中类继承，到底继承了什么 ? (3)

5. 一次性搞清楚equals和hashCode(2)

树莓派 40Pin 引脚对照表

wiringPi 编码	BCM 编码	功能名	物理引脚 BOARD编码		功能名	BCM 编码	wiringPi 编码
		3.3V	1	2	5V		
8	2	SDA.1	3	4	5V		
9	3	SCL.1	5	6	GND		
7	4	GPIO.7	7	8	TXD	14	15
		GND	9	10	RXD	15	16
0	17	GPIO.0	11	12	GPIO.1	18	1
2	27	GPIO.2	13	14	GND		
3	22	GPIO.3	15	16	GPIO.4	23	4
		3.3V	17	18	GPIO.5	24	5
12	10	MOSI	19	20	GND		
13	9	MISO	21	22	GPIO.6	25	6
14	11	SCLK	23	24	CE0	8	10
		GND	25	26	CE1	7	11
30	0	SDA.0	27	28	SCL.0	1	31
21	5	GPIO.21	29	30	GND		
22	6	GPIO.22	31	32	GPIO.26	12	26
23	13	GPIO.23	33	34	GND		
24	19	GPIO.24	35	36	GPIO.27	16	27
25	26	GPIO.25	37	38	GPIO.28	20	28
		GND	39	40	GPIO.29	21	29

表格由树莓派实验室绘制 <http://shumeipai.nxez.com>

wiringPi库API大全

在使用wiringPi库时，你需要包含头文件 `#include<wiringPi.h>`。凡是写wiringPi的程序，都包含这个头文件。

硬件初始化函数

使用wiringPi时，你必须在执行任何操作前初始化树莓派，否则程序不能正常工作。

可以调用下表函数之一进行初始化，它们都会返回一个int，返回 -1 表示初始化失败。

<code>int wiringPiSetup (void)</code>	返回执行状态，-1表示失败	当使用这个函数初始化树莓派引脚时，程序使用的是wiringPi 引脚编号表。引脚的编号为 0~1 需要root权限
<code>int wiringPiSetupGpio (void)</code>	返回执行状态，-1表示失败	当使用这个函数初始化树莓派引脚时，程序中使用的是BCM GPIO 引脚编号表。 需要root权限
<code>wiringPiSetupPhys(void)</code>	不常用，不做介绍	/
<code>wiringPiSetupSys (void) ;</code>	不常用，不做介绍	/

通用GPIO控制函数

<code>void pinMode (int pin, int mode)</code>	<code>pin</code> : 配置的引脚 <code>mode</code> :指定引脚的IO模式 可取的值: INPUT、OUTPUT、PWM_OUTPUT , GPIO_CLOCK	作用：配置引脚的IO模式 注意： 只有wiringPi 引脚编号下的1脚（BCM下的18
---	--	--

		只有wiringPi编号下的7（BCM下的4号）支持
void digitalWrite (int pin, int value)	pin：控制的引脚 value：引脚输出的电平值。 可取的值：HIGH，LOW分别代表高低电平	让对一个已近配置为输出模式的 引脚 输出指
int digitalRead (int pin)	pin：读取的引脚 返回：引脚上的电平，可以是LOW HIGH 之一	读取一个引脚的电平值 LOW HIGH ，返回
void analogWrite(int pin, int value)	pin:引脚 value：输出的模拟量	模拟量输出 树莓派的引脚本身是不支持AD转换的，也就 需要增加另外的模块
int analogRead (int pin)	pin：引脚 返回：引脚上读取的模拟量	模拟量输入 树莓派的引脚本身是不支持AD转换的，也就 需要增加另外的模块
void pwmWrite (int pin, int value)	pin：引脚 value：写入到PWM寄存器的值，范围在0~1024之间。	输出一个值到PWM寄存器，控制PWM输出。 pin只能是wiringPi 引脚编号下的1脚（ BCM T
void pullUpDnControl (int pin, int pud)	pin：引脚 pud：拉电阻模式 可取的值：PUD-OFF 关闭拉电阻 PUD_DOWN 引脚电平拉到3.3v PUD_UP 引脚电平拉到0v 接地	对一个设置IO模式为 INPUT 的输入引脚设置 与Arduino不同的是，树莓派支持的拉电阻模 树莓派内部的拉电阻达50K欧姆

LED闪烁程序

```
#include<iostream>
#include<cstdlib>
#include<wiringPi.h>

const int LEDpin = 1;

int main()
{
    if(-1==wiringPiSetup())
    {
        cerr<<"setup error\n";
        exit(-1);
    }
    pinMode(LEDpin,OUTPUT);

    for(size_t i=0;i<10;++i)
    {
        digitalWrite(LEDpin,HIGH);
        delay(600);
        digitalWrite(LEDpin,LOW);
        delay(600);
    }

    cout<<"-----bye-----"<<endl;
    return 0;
}
```

PWM输出控制LED呼吸灯的例子

```
#include<iostream>
#include<wiringPi.h>
#include<cstdlib>
using namespace std;

const int PWMpin = 1; //只有wiringPi编号下的1脚（BCM标号下的18脚）支持
void setup();

int main()
{
    setup();
    int val = 0;
    int step = 2;
    while(true)
    {
        if(val>1024)
        {
            step = -step;
            val = 1024;
        }
        else if(val<0)
        {
            step = -step;
            val = 0;
        }

        pwmWrite(PWMpin,val);
        val+=step;
        delay(10);
    }

    return 0;
}

void setup()
{
    if(-1==wiringPiSetup())
    {
        cerr<<"setup error\n";
        exit(-1);
    }
    pinMode(PWMpin,PWM_OUTPUT);
}
```

时间控制函数

unsigned int millis (void)	这个函数返回 一个 从你的程序执行 wiringPiSetup 初始化函数（或者wiringPiSetupGpio ）到 当前时间 返回类型是unsigned int，最大可记录 大约49天的毫秒时长。
unsigned int micros (void)	这个函数返回 一个 从你的程序执行 wiringPiSetup 初始化函数（或者wiringPiSetupGpio ）到 当前时间 返回类型是unsigned int，最大可记录 大约71分钟的时长。
void delay (unsigned int howLong)	将当前执行流暂停 指定的毫秒数。因为Linux本身是多线程的，所以实际暂停时间可能会长一些。参数最大可达49天
void delayMicroseconds (unsigned int howLong)	将执行流暂停 指定的微秒数（1000微秒 = 1毫秒 = 0.001秒）。因为Linux本身是多线程的，所以实际暂停时间可能会长一些。参数是unsigned int 类型，最大延时时间

中断

wiringPi提供了一个中断处理注册函数，它只是一个注册函数，并不处理中断。他无需root权限。

int wiringPiISR (int pin, int edgeType, void (*function) (void))	返回值：返回负数则代表注册失败 pin：接受中断信号的引脚 edgeType：触发的方式。	注册的函数会在中断发生时执行 和51单片机不同的是：这个注册的中断处理函数会和main函数并发执行（同时执
--	---	--

	<div><div>INT_EDGE_FALLING：下降沿触发</div><div>INT_EDGE_RISING：上升沿触发</div><div>INT_EDGE_BOTH：上下降都会触发</div><div>INT_EDGE_SETUP：编程时用不到。</div></div> <div><div>function：中断处理函数的指针，它是一个无返回值，无参数的函数。</div></div>	<div>当本次中断函数还未执行完毕，这个时候树莓派又触发了一个中断，那么这个后续行。但是wiringPi最多可以跟踪并记录后来的仅仅1个中断，如果不止1个，则他们</div>
--	---	--

通过1脚检测 因为按键按下引发的 下降沿，触发中断，反转11控制的LED

```
#include<iostream>
#include<wiringPi.h>
#include<cstdlib>
using namespace std;

void ButtonPressed(void);
void setup();

/*****/
const int LEDPin = 11;
const int ButtonPin = 1;
/*****/

int main()
{
    setup();

    //注册中断处理函数
    if(0>wiringPiISR(ButtonPin,INT_EDGE_FALLING,ButtonPressed))
    {
        cerr<<"interrupt function register failure"<<endl;
        exit(-1);
    }

    while(1)
    ;

    return 0;
}

void setup()
{
    if(-1==wiringPiSetup())
    {
        cerr<<"wiringPi setup error"<<endl;
        exit(-1);
    }

    pinMode(LEDPin,OUTPUT);    //配置11脚为控制LED的输出模式
    digitalWrite(LEDPin,LOW); //初始化为低电平

    pinMode(ButtonPin,INPUT);    //配置1脚为输入
    pullUpDnControl(ButtonPin,PUD_UP); //将1脚上拉到3.3v
}

//中断处理函数：反转LED的电平
void ButtonPressed(void)
{
    digitalWrite(LEDPin, (HIGH==digitalRead(LEDPin))?LOW:HIGH );
}


```

多线程

wiringPi提供了简单的Linux系统下的通用的 Posix threads线程库接口来支持并发。

<div>int piThreadCreate(name)</div>	<div>name:被包装的线程执行函数</div> <div>返回：状态码。返回0表示成功启动，反之失败。</div> <div>源代码：<pre>int piThreadCreate (void *(*fn)(void *)) { pthread_t myThread ; return pthread_create (&myThread, NULL, fn, NULL) ; }</pre></div>	<div>包装一个用PI_THREAD定义的函数为一个线程，并启动这个线程。</div> <div>首先你需要通过以下方式创建一个特特殊的函数，这个函数中的代码就是在新线程中执行的代码，已线程的名字，可自定义。</div> <div><pre>PI_THREAD (myThread) { //在这里面写上的代码会和主线程并发执行。 }</pre></div> <div>在wiringPi.h中，我发现这样一个宏定义：#define PI_THREAD(X) void *X (void *) { ... } 那么，被预处理后我们写的线程函数会变成下面这个样子，请注意返回值，难怪我总说，以后注意返回NULL，或者 (void*)0</div> <div><pre>void *myThread (void *dummy) { //在这里面写上的代码会和主线程并发执行。 }</pre></div>
<div>piLock(int keyNum)</div>	<div>keyNum:0-3的值，每一个值代表一把锁</div>	<div>使能同步锁。wiringPi只提供了4把锁，也就是keyNum只能取0~3的值，官方认为有4把锁足够了。</div> <div>keyNum : 0,1,2,3 每一个数字就代表一把锁。</div> <div>源代码：</div> <div><pre>void piLock (int keyNum) { pthread_mutex_lock (&piMutexes [keyNum]); }</pre></div>
<div>piUnlock(int keyNum)</div>	<div>keyNum:0-3的值，每一个值代表一把锁</div>	<div>解锁，或者说让出锁。</div> <div>源代码：</div> <div><pre>void piUnlock (int key) { pthread_mutex_unlock (&piMutexes [key]); }</pre></div>
<div>int piHiPri (int priority)</div>	<div>priority：优先级指数，0~99</div> <div>返回值：0，成功</div> <div>-1：失败</div>	<div>设定线程的优先级，设定线程的优先级变高，不会使程序运行加快，但会使这个线程的优先级更高。</div> <div>比如你的程序只用到了主线程，和另一个线程A，主线程设定优先级为1，A线程设定为2，那也代表A比main线程优先级高。</div>

凡是涉及到多线程编程，就会涉及到线程安全的问题，多线程访问同一个数据，需要使用同步锁来保障数据操作正确性和符合预期。

当A线程锁上 锁S 后，其他共用这个锁的竞争线程，只能等到锁被释放，才能继续执行。

成功执行了piLock 函数的线程将拥有这把锁。其他线程想要拥有这把锁必须等到这个线程释放锁，也就是这个线程执行piUnlock后。

同时要扩展的知识是：volatile 这个C/C++中的关键字，它请求编译器不缓存这个变量的数据，而是每次都从内存中读取。特别是在多线程下对变量的操作，使用volatile声明才是保险的。

softPwm，软件实现的PWM

树莓派硬件上支持的PWM输出的引脚有限，为了突破这个限制，wiringPi提供了软件实现的PWM输出API。

需要包含头文件：`#include <softPwm.h>`

编译时需要添pthread库链接 `-lpthread`

<code>int softPwmCreate (int pin, int initialValue, int pwmRange)</code>	<p>pin：用来作为软件PWM输出的引脚</p> <p>initialValue：引脚输出的初始值</p> <p>pwmRange：PWM值的范围上限</p> <p>建议使用100.</p> <p>返回：0表示成功。</p>	使用一个指定的pin引脚创建一个模拟的PWM输出引脚
<code>void softPwmWrite (int pin, int value)</code>	<p>pin：通过softPwmCreate创建的引脚</p> <p>value：PWM引脚输出的值</p>	更新引脚输出的PWM值

串口通信

使用时需要包含头文件：`#include <wiringSerial.h>`

<code>int serialOpen (char *device, int baud)</code>	<p>device:串口的地址，在Linux中就是设备所在的目录。</p> <p>默认一般是"/dev/ttyAMA0",我的是这样的。</p> <p>baud：波特率</p> <p>返回：正常返回文件描述符，否则返回-1失败。</p>	打开并初始串口
<code>void serialClose (int fd)</code>	fd：文件描述符	关闭fd关联的串口
<code>void serialPutchar (int fd, unsigned char c)</code>	<p>fd:文件描述符</p> <p>c:要发送的数据</p>	发送一个字节的数据到串口
<code>void serialPuts (int fd, char *s)</code>	<p>fd：文件描述符</p> <p>s：发送的字符串，字符串要以'\0'结尾</p>	发送一个字符串到串口
<code>void serialPrintf (int fd, char *message, ...)</code>	<p>fd：文件描述符</p> <p>message：格式化的字符串</p>	像使用C语言中的printf一样发送数据到串口
<code>int serialDataAvail (int fd)</code>	<p>fd：文件描述符</p> <p>返回：串口缓存中已经接收的，可读取的字节数，-1代表错误</p>	获取串口缓存中可用的字节数。
<code>int serialGetchar (int fd)</code>	<p>fd：文件描述符</p> <p>返回：读取到的字符</p>	<p>从串口读取一个字节数据返回。</p> <p>如果串口缓存中没有可用的数据，则会返回-1。</p> <p>所以，在读取前，做好通过serialDataAvail检查。</p>
<code>void serialFlush (int fd)</code>	fd：文件描述符	刷新，清空串口缓冲中的所有可用的数据
<code>ssize_t write (int fd, const void * buf, size_t count)</code>	<p>fd：文件描述符</p> <p>buf：需要发送的数据缓存数组</p> <p>count:发送buf中的前count个字节数据</p> <p>返回：实际写入的字符数，错误返回-1</p>	<p>这个是Linux下的标准IO库函数，需要包含unistd.h</p> <p>当要发送到的数据量过大时，wiringPi建议分多次发送。</p>
<code>ssize_t read (int fd, void * buf, size_t count);</code>	<p>fd：文件描述符</p> <p>buf：接受的数据缓存的数组</p> <p>count:接收的字节数.</p> <p>返回：实际读取的字符数。</p>	<p>这个是Linux下的标准IO库函数，需要包含unistd.h</p> <p>当要接收的数据量过大时，wiringPi建议分多次接收。</p>

初次使用树莓派串口编程，需要配置。我开始搞了很久，以为是程序写错了 还一直在调试。。。 (~_~)~

```

/* 修改 cmdline.txt文件 */
>cd /boot/
>sudo vim cmdline.txt

删除 【】 之间的部分
dwc_otg.lpm_enable=0 【console=ttyAMA0,115200】 kgdboc=ttyAMA0,115200 console=tty1 root=/dev/mmcblk0p2 rootfstype=
rootwait

/*修改 inittab文件 */
>cd /etc/
>sudo vim inittab

注释掉最后一行内容：，在前面加上 # 号
#T0:23:respawn:/sbin/getty -L ttyAMA0 115200 vt100

sudo reboot 重启

```

下面是双机通信的一个例子

C51代码，作为串口通信的接发送。serial库请看另一篇文章

```

#include<reg52.h>
#include"serial.h"

/*****function*****/
bit isOpenPressed(void);
bit isClosePressed(void);
void delay(unsigned int t);
/*****/

sbit closeButton = P2^0; //与关闭按钮相连的引脚
sbit openButton = P2^1; //与打开按钮相连的引脚

void main(void)
{
    closeButton = 1; //拉高
    openButton = 1; //拉高

    EA =1; //打开总中断
    serial_init(9600); //初始化51串口

    while(1)
    {
        if(isClosePressed()) //如果关闭按钮按下
        {
            serial_write(0); //发送数据 0给树莓派
            delay(10);
        }

        else if(isOpenPressed()) //如果打开按钮按下
        {
            serial_write(1); //发送数据 1给树莓派
            delay(10);
        }
    }
}

bit isOpenPressed(void)
{
    bit press =0;

    if(0==openButton)
    {
        delay(5);
        if(0==openButton)
        {
            while(!openButton)
            ;
            press = 1;
        }
    }
}

```



```

    }
}

return press;
}

bit isClosePressed(void)
{
    bit press = 0;

    if(0==closeButton)
    {
        delay(5);
        if(0==closeButton)
        {
            while(!closeButton)
            ;
            press = 1;
        }
    }

    return press;
}

void delay(unsigned int t)
{
    unsigned int i ;
    unsigned char j;
    for(i = t;i>0;i--)
        for(j=120;j>0;j--)
            ;
}

```

树莓派代码，作为串口通信的接收方

```

#include<iostream>
#include<cstdlib>
#include<wiringPi.h>
#include<wiringSerial.h>
using namespace std;

void setup();
const int LEDPin = 11;

int main()
{
    setup();

    int fd; //Linux 的思想是：将一切IO设备，都看做 文件，fd就是代表串口抽象出来的文件

    if((fd = serialOpen("/dev/ttyAMA0",9600))== -1)    //初始化串口，波特率9600
    {

        cerr<<"serial open error"<<endl;
        exit(-1);
    }

    while(true)
    {

        if(serialDataAvail(fd) >= 1)    //如果串口缓存中有数据
        {
            int data = serialGetchar(fd);

            if(data==0)    //接受到51发送的 数据 0
            {
                // close led
                digitalWrite(LEDPin,LOW);
            }

            else if(data==1)    //接受到51发送的 数据 1
            {
                //open led
                digitalWrite(LEDPin,HIGH);
            }
        }
    }
}

```

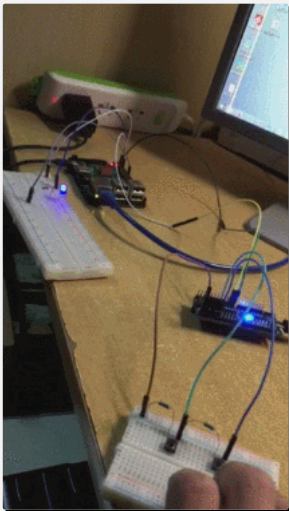
```
    }

    return 0;
}

void setup()
{
    if(-1==wiringPiSetup())
    {
        cerr<<"set up error"<<endl;
        exit(-1);
    }

    pinMode(LEDpin, OUTPUT);
    digitalWrite(LEDpin, HIGH);
}

```



shift移位寄存器芯片API

需要包含头文件 #include <wiringShift.h>

void shiftOut (uint8_t dPin, uint8_t cPin, uint8_t order, uint8_t val)	dPin : 移位芯片的串行数据入口引脚, 比如74HC595的SER脚	将val串化, 通过芯片 如常见的74HC595
	cPin : 移位芯片的时钟引脚. 如74HC595的11脚	
	order :	
	LSBFIRST 先发送数据的低位	
	MSBFIRST先发送数据的高位	
	val : 要发送的8位数据	
uint8_t shiftIn (uint8_t dPin, uint8_t cPin, uint8_t order)	同上。	将并行数据, 通过芯

用过595的都知道还有一个引脚: 12脚, Rpin, 用于把移位寄存器中的数据更新到存储寄存器中, 然后wiringPi的API中没有使用这个引脚。我

```

#include<iostream>
#include<wiringPi.h>
#include <wiringShift.h>
#include<cstdlib>
using namespace std;

const int SERpin = 1; //serial data input
const int SCKpin = 2; //shift register clock
const int RCKpin = 3; // storage register clock

```

```

/*****/
void setup();

/*****/

int main()
{
    setup();

    for(int i=0;i<8;++i)
    {
        digitalWrite(RCKpin,LOW);

        shiftOut(SERpin,SCKpin,LSBFIRST,1<<i);
        digitalWrite(RCKpin,HIGH);

        delay(800);
    }
    return 0;
}

void setup()
{
    if(-1==wiringPiSetup())
    {
        cerr<<"setup error\n";
        exit(-1);
    }

    pinMode(SERpin,OUTPUT);
    pinMode(RCKpin,OUTPUT);
    pinMode(SCKpin,OUTPUT);
}

```

树莓派硬件平台特有的API

并没有列全，我只是列出了相对来说有用的，其他的，都基本不会用到。

pwmSetMode (int mode)	mode : PWM运行模式	设置PWM的运行模式。 pwm发生器可以运行在2种模式下，通过参数指定： PWM_MODE_BAL : 树莓派默认的PWM模式 PWM_MODE_MS : 传统的pwm模式，
pwmSetRange (unsigned int range)	range，范围的最大值 0~range	设置pwm发生器的数值范围，默认是1024
pwmSetClock (int divisor)		This sets the divisor for the PWM clock. To understand more about the PWM system, you'll need to read the Broadcom A...
piBoardRev (void)	返回：树莓派板子的版本编号 1或者2	/

就这样，以后会更新。

欢迎转载，请注明出处：www.cnblogs.com/lulipro

为了更好的阅读体验，请访问原博客地址。

限于本人水平，如果文章和代码有表述不当之处，还请不吝赐教。


分类: 树莓派

好文要顶

关注我

收藏该文





代码钢琴家

关注 - 15


粉丝 - 22

+加关注

- « 上一篇：【C51】UART串口通信
- » 下一篇：使用Notepad++代替笨拙的Arduino IDE


posted @ 2016-10-27 21:54 代码钢琴家 View

Post Comment

#1楼 2017-02-12 19:05 | yuepengdaifei 

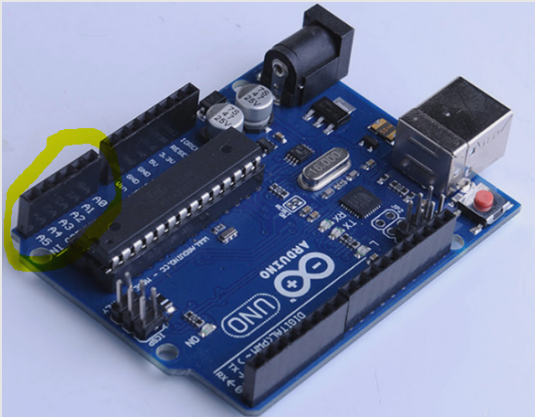
请问如何树莓派才能正常使用analogread？文中说“树莓派的引脚本身是不支持AD转换的，也就是不能使用模拟量的API，需要增加什么模块呢？


我现有一传感器的输出电压要用analogread来读取，然而用analogread读取值一直为0。。。求助。。。O(∩_∩)O谢谢！

#2楼[楼主] 2017-02-12 21:35 | 代码钢琴家 

@ yuepengdaifei

我看了下wiringPi 官方的说明，里面提到了2款扩展板：Gertboard，quick2Wire analog board可以用来支持模拟信号。但是这2款板子价格有点贵。你可以试试用一些AD转换芯片。我的建议是直接使用Arduino板去做，如Uno .价格也很便宜，板子自身支持模拟信号。自带的analogRead(),analogWrite()



 注册用户登录后才能发表评论，请 [登录](#) 或 [注册](#)，[访问网站首页](#)。

【推荐】50万行VC++源码: 大型组态工控、电力仿真CAD与GIS源码库

**最新IT新闻:**

- 都有哪些改变？Windows 10 Creators Update详细上手
 - 微软收购软件容器提供商Deis
 - 外卖决战：“掉队者”百度左右战局？
 - 13张美国首次登月照：从训练到回收再到游行庆祝
 - 索尼推出售价700美元的电子纸平板电脑DPT-RP1
- » 更多新闻...

**最新知识库文章:**

- 如何打好前端游击战
 - 技术文章的阅读姿势
 - 马拉松式学习与技术人员的成长性
 - 程序员的“认知失调”
 - 为什么有的人工作多年还是老样子
- » 更多知识库文章...