

物联网 IoT 经验分享小站

三傻大闹宝莱坞——追求卓越

目录视图摘要视图RSS 订阅

个人资料



xukai871105

关注发私信

访问：614267次

积分：8454

等级：BLDC E

排名：第834名

原创：180篇 转载：2篇

译文：4篇 评论：880条

联系方式

E-mail：xukai19871105@126.com

所在地：江苏无锡

TEL：13812055074

文章搜索

Q

文章分类

- 树莓派 (37)
- 物联网学习笔记 (35)
- Contiki (11)
- Linux学习笔记 (22)
- PHP/Python/前端 (21)
- 嵌入式ARM (24)
- 单片机 (27)
- C/C++/C# (4)
- 个人计划/个人随笔 (5)
- opencv (1)

文章存档

- 2015年04月 (1)
- 2015年02月 (1)
- 2015年01月 (3)
- 2014年11月 (2)
- 2014年10月 (1)

展开

阅读排行

- 树莓派学习笔记——GPIO功能... (26475)
- 树莓派学习笔记——wiringPi... (13124)
- 树莓派学习笔记——wiringPi... (12612)

微信公众平台开发入门Markdown编辑器轻松写博文欧考，你过了吗？天天发答题 一大波C币向你袭来读文章说感想送好书

树莓派学习笔记——wiringPi GPIO使用详解

分类：树莓派2014-01-05 12:1012636人阅读评论(30)收藏举报

树莓派wiringPiGPIO

目录(?)

[+]

1.前言

最近认真学习了树莓派，从浅到深认真分析了wiringPi实现代码，借助树莓派学习linux收获颇丰。深入学习linux一段时间后发现它非常有魅力，一个简单的IO口输出操作尽有那么多的“玩法”。wiringPi是一个简单易用的函数库，通过wiringPi可以扩展SPI和I2C等芯片，关于wiringPi的介绍和安装请参考我的另一篇【博文】。

本篇博文将通过一个简单的例子呈现wiringPi的使用，虽然例子简单但会深入分析wiringPi内部实现代码。
【树莓派学习笔记——索引博文】

2.BCM2835 GPIO相关寄存器

(该部分表述可能有误，正在确认修改中)

树莓派平台的GPIO驱动，例如RPi.GPIO和WiringPi均采用直接操作GPIO寄存器的方式，树莓派的CPU采用博通的BCM2835，想要更好的了解树莓派的GPIO驱动实现就必须阅读BCM2835的数据手册。在BCM2835数据手册中需要认真关注两个内容：

外设寄存器物理地址和外设虚拟地址的映射关系。在linux操作系统中，借助ARM内部的MMU，CPU外设物理地址映射成了虚拟地址，外设的物理起始地址为0x7E00 0000，被MMU虚拟之后的起始地址为0x2000 0000。以此类推，GPIO外设物理起始地址为0x7E20 0000 = 0x7E00 0000+0x0020 0000，被MMU虚拟之后的GPIO外设地址为0x2000 0000+0x0020 0000。那么对于Linux系统而言，GPIO相关操作的起始地址为0x2020 0000。BCM2835的内部映射关系如下图所示。

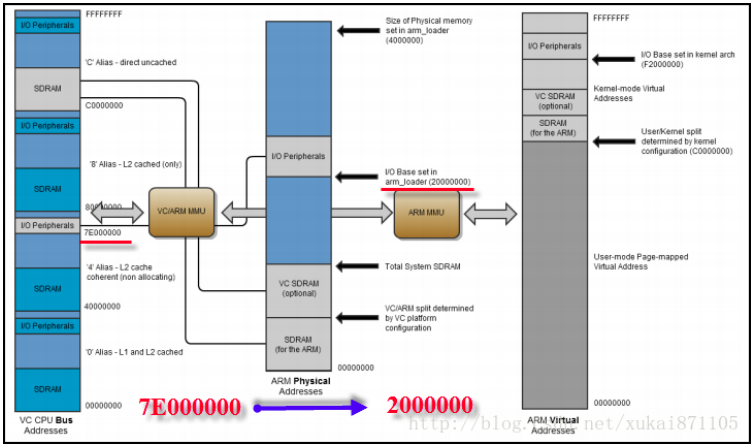


图1 BCM2835 物理地址和虚拟地址映射关系

GPFSelx、GPSETx、GPCLRx和GPLEVn寄存器。简单来说，GPFSelx为IO口方向或复用寄存器，负责IO口方向例如输入或输出；GPSETx为IO口输出寄存器，负责IO口输出逻辑高电平；GPCLRx寄存器同为IO口输出寄存器，不过和GPSETx相反，负责输出逻辑低电平。GPLEVx为IO口输入寄存器，负责IO口输入状态。

(亲爱的网友们，如果您不理解这些寄存器也不理解MMU机制，也不会影响您使用wiringPi。请放心大胆地使用wiringPi，它已经帮您完成了很多基础性的工作)

- Instant Contiki 安装笔记——

(11159)
- FreeRTOS STM32移植笔记

(10565)
- CC2530 RF部分使用 ——实现…

(10229)
- 树莓派学前班——设置屏幕分…

(8581)
- 树莓派学习笔记——索引博文

(8280)
- YeeLink Http请求格式分析

(8034)
- CoAP协议学习——CoAP基础

(7797)

最新评论

- AVR Studio 6设置技巧

yy5334407 ： 真赞！
- 树莓派学习笔记——webiopi安装与入门

xukai871105 ： @qq369251823:这两个东西应该不是一个层面的东西！
- 树莓派学习笔记——webiopi安装与入门

qq369251823 ： 楼主，webiopi可以和zigbee配合使用吗？我想搭建一个物联网
- STM32NET学习笔记——TCP部分

xukai871105 ： @u013164357:当然是买块开发板要方便的多了！
- STM32NET学习笔记——TCP部分

u013164357 ： 楼主是通过开发板学习的吗？
- 树莓派学习笔记——获取树莓派CPU温度

xukai871105 ： @keygle:好像是写错了，我删除这行吧！
- 树莓派学习笔记——获取树莓派CPU温度

keygle ： python 代码段的首行 !#/usr/bin/python 写错了 应该是 #!/usr/b...
- uIP学习笔记

xukai871105 ： @lionwes:其实可以用网卡直接接到PC机，但是要用另外一种网线。我建议还是连接到路由器上，这...
- uIP学习笔记

lionwes ： 楼主，在做ping动作的时候，STM32 需要连接到路由器吗？还是STM32直接连接PC的网口，就可...
- contiki STM32移植

lionwes ： @xukai871105:OK谢谢

3. 简单测试代码

下面通过一个简单的代码实现树莓派流水灯，在树莓派（树莓派版本2）中可以直接利用的IO口共有8个，在wiringPi中的编号为GPIO0到GPIO7，对于BCM2835而言编号分别为17，18，27，22，23，24，25，4。具体对应关系见下图。

P1: The Main GPIO connector						
WiringPi Pin	BCM GPIO	Name	Header	Name	BCM GPIO	WiringPi Pin
		3.3v	1 2	5v		
8	Rv1:0 - Rv2:2	SDA	3 4	5v		
9	Rv1:1 - Rv2:3	SCL	5 6	0v		
7	4	GPIO7	7 8	TxD	14	15
		0v	9 10	RxD	15	16
0	17	GPIO0	11 12	GPIO1	18	1
2	Rv1:21 - Rv2:27	GPIO2	13 14	0v		
3	22	GPIO3	15 16	GPIO4	23	4
		3.3v	17 18	GPIO5	24	5
12	10	MOSI	19 20	0v		
13	9	MISO	21 22	GPIO6	25	6
14	11	SCLK	23 24	CE0	8	10
		0v	25 26	CE1	7	11
WiringPi Pin	BCM GPIO	Name	Header	Name	BCM GPIO	WiringPi Pin

图2 wiringPi GPIO 和 BCM2835 GPIO映射关系

```
[cpp]
01. #include <wiringPi.h>
02. int main( )
03. {
04.     // 初始化wiringPi
05.     wiringPiSetup();
06.
07.     int i = 0;
08.     // 设置IO口全部为输出状态
09.     for( i = 0 ; i < 8 ; i++ )
10.         pinMode(i, OUTPUT);
11.
12.     for (;;)
13.     {
14.         for( i = 0 ; i < 8 ; i++ )
15.         {
16.             // 点亮500ms 熄灭500ms
17.             digitalWrite(i, HIGH); delay(500);
18.             digitalWrite(i, LOW); delay(500);
19.         }
20.     }
21.
22.     return 0;
23. }
```

为了方便生成可执行文件，可编写以下makefile文件，CD进入该目录之后直接make即可。

```
[cpp]
01. blink:blink.o
02. gcc blink.c -o blink -lwiringPi
03. clean:
04. rm -f blink blink.o
```

图3 代码运行结果

4. 代码详解

上面的代码非常简单，可以分为四个部分——wiringPiSetup初始化、pinMode设置IO为输出方向、digitalWrite输出高电平或低电平和delay系统延时函数。

4.1 wiringPiSetup

```
[cpp]
01. int wiringPiSetup (void)
02. {
03.     int fd ;
04.     int boardRev ;
05.     // 第一步，获得树莓派的版本编号，并根据版本编号映射IO口
06.     boardRev = piBoardRev () ;
07.     if (boardRev == 1)
08.     {
09.         pinToGpio = pinToGpioR1 ;
10.         physToGpio = physToGpioR1 ;
11.     }
12.     else
```

```
13.     {
14.         pinToGpio = pinToGpioR2 ;
15.         physToGpio = physToGpioR2 ;
16.     }
17.
18.     // 第二步，打开/dev/mem设备，使得在用户空间可以直接操作内存地址
19.     if ((fd = open ("/dev/mem", O_RDWR | O_SYNC | O_CLOEXEC)) < 0)
20.         return wiringPiFailure (WPI_ALMOST, "wiringPiSetup: Unable to open /dev/mem: %s\n", strerror (errno)) ;
21.
22.     gpio = (uint32_t *)mmap(0, BLOCK_SIZE, PROT_READ|PROT_WRITE, MAP_SHARED, fd, GPIO_BASE) ;
23.     if ((int32_t)gpio == -1)
24.         return wiringPiFailure (WPI_ALMOST, "wiringPiSetup: mmap (GPIO) failed: %s\n", strerror (errno)) ;
25.
26.     // 第三步，设定wiringPi GPIO外设的操作模式
27.     wiringPiMode = WPI_MODE_PINS ;
28.     return 0 ;
29. }
```

该部分代码的实现可以分为三步（注意该部分并不是wiringPiSetup的完整代码，为了说明问题对代码进行简化）

第一步，获得树莓派的版本编号，并根据版本编号映射IO口。pinToGpioR2为树莓派版本2的GPIO映射关系，不但包括（括SPI、I2C和UART等。此处physToGpioRx存在疑问。

第二步，打开/dev/mem设备，使得在用户空间可以直接操作内存地址。/dev/mem是物理内存的全映像，可以用来访问物理内存（能够访问物理内存当然也包括MCU外设），一般用法是open("/dev/mem",O_RDWR|O_SYNC)，接着可以用mmap的地址来访问物理内存（此处为GPIO_BASE），这是实现用户空间驱动的一种方法【[参考博文](#)】。（该部分需要深入，请关注后期博文）

第三步，设定wiringPi GPIO外设的操作模式。此处也存在若干疑惑，默认情况便是使用WPI_MODE_PINS 模式，wiringPi的IO管脚编号和BCM IO管脚编号存在一个固定映射关系，但是wiringPi其他代码中还存在wiringPiSetupSys函数，该函数操作GPIO端口时通过/sys/class/gpio中的驱动文件实现，这也是实现树莓派GPIO操作的另一个途径。这种方法便是应用Sysfs——Sysfs 是 Linux 2.6 所提供的一种虚拟文件系统，这个文件系统不仅可以把设备（devices）和驱动程序(drivers)的信息从内核输出到 用户空间，也可以用来对设备和驱动程序做设置【[wiki百科](#)】。（该部分需要深入，请关注后期博文）。

```
[cpp] C P
01. int wiringPiSetupSys (void)
02. {
03.     int boardRev ;
04.     int pin ;
05.     char fName [128] ;
06.     // 获得树莓派版本编号，版本1或者版本2
07.     boardRev = piBoardRev () ;
08.     if (boardRev == 1)
09.     {
10.         pinToGpio = pinToGpioR1 ;
11.         physToGpio = physToGpioR1 ;
12.     }
13.     else
14.     {
15.         pinToGpio = pinToGpioR2 ;
16.         physToGpio = physToGpioR2 ;
17.     }
18.     // 查找/sys/class/gpio，并记录GPIOx操作文件fd
19.     for (pin = 0 ; pin < 64 ; ++pin)
20.     {
21.         sprintf (fName, "/sys/class/gpio/gpio%d/value", pin) ;
22.         sysFds [pin] = open (fName, O_RDWR) ;
23.     }
24.     // 设置操作模式为 sysfs模式 文件方式驱动GPIO而非寄存器方式
25.     wiringPiMode = WPI_MODE_GPIO_SYS ;
26.     return 0 ;
27. }
```

4.2 pinMode

```
[cpp] C P
01. void pinMode (int pin, int mode)
02. {
03.     int fSel, shift, alt ;
04.     struct wiringPiNodeStruct *node = wiringPiNodes ;
05.
06.     // 树莓派 板载GPIO设置，板载GPIO的管脚编号必须小于64
07.     if ((pin & PI_GPIO_MASK) == 0)
08.     {
09.         // 第一步 确定BCM GPIO引脚编号
10.         if (wiringPiMode == WPI_MODE_PINS)
11.             pin = pinToGpio [pin] ;
12.         // 第二步，确定该管脚对应的fSel寄存器
13.         fSel = gpioToGPFSEL [pin] ;
14.         shift = gpioToShift [pin] ;
15.
16.         // 第三步，根据输入和输出状态设置fSel寄存器
17.         if (mode == INPUT)
```

```
18.         *(gpio + fSel) = (*(gpio + fSel) & ~(7 << shift));
19.     else if (mode == OUTPUT)
20.         *(gpio + fSel) = (*(gpio + fSel) & ~(7 << shift)) | (1 << shift);
21.     }
22.
23.     // 树莓派 外扩GPIO设置
24.     else
25.     {
26.         if ((node = wiringPiFindNode (pin)) != NULL)
27.             node->pinMode (node, pin, mode);
28.         return;
29.     }
30. }
```

该部分代码的实现可以分为三步（注意该部分并不是pinMode 的完整代码，为了说明问题对代码进行简化）

【注意】在wiringPi中，pin编号小于64认为是板载GPIO，如果编号大于64则认为是外扩GPIO，例如使用外扩的MCP23017或者PCF8574，同时外扩的AD和DA芯片的相应pin也应该大于64。

第一步，确定BCM GPIO引脚编号。如果是树莓派2版本，那么映射关系由数组pinToGpioR2决定

```
(cpp)
01. static int pinToGpioR2 [64] =
02. {
03.     17, 18, 27, 22, 23, 24, 25, 4, // GPIO 0 through 7: wpi 0 - 7
04.     2, 3, // I2C - SDA0, SCL0 wpi 8 - 9
05.     8, 7, // SPI - CE1, CE0 wpi 10 - 11
06.     10, 9, 11, // SPI - MOSI, MISO, SCLK wpi 12 - 14
07.     14, 15, // UART - Tx, Rx wpi 15 - 16
08.     28, 29, 30, 31, // New GPIOs 8 though 11 wpi 17 - 20
09. };
```

第二步，根据输入和输出状态设置fSel寄存器。作者采用简单明了的查表法，在一个FSEL寄存器中共可设置10个GPIO管脚。具体的含义可查看数据手册和gpioToGPFSEL、gpioToShift的具体定义

```
(cpp)
01. static uint8_t gpioToGPFSEL [] =
02. {
03.     0,0,0,0,0,0,0,0,0,0,
04.     1,1,1,1,1,1,1,1,1,1,
05.     2,2,2,2,2,2,2,2,2,2,
06.     3,3,3,3,3,3,3,3,3,3,
07.     4,4,4,4,4,4,4,4,4,4,
08.     5,5,5,5,5,5,5,5,5,5,
09. };
10. static uint8_t gpioToShift [] =
11. {
12.     0,3,6,9,12,15,18,21,24,27,
13.     0,3,6,9,12,15,18,21,24,27,
14.     0,3,6,9,12,15,18,21,24,27,
15.     0,3,6,9,12,15,18,21,24,27,
16.     0,3,6,9,12,15,18,21,24,27,
17. };
```

第三步，根据输入和输出状态设置FSEL寄存器。结合第二步便可发现其中的设置技巧。例如操作wiringPi的GPIO0对应BCM GPIO17；那么查找gpioToGPFSEL表，应操作第1个（从0开始计数）FSEL1寄存器；*(gpio + fSel)中gpio指GPIO外设的虚拟起始地址，此处为0x2200000，第二个FSEL寄存器在此基础上地址偏移1。shift决定置1或者置0的具体位，例如此时的GPIO17，对应该fSel寄存器的21位；如果是输入状态21-23位全部为0，如果是输出状态21位为1，具体代码如下：

```
*(gpio + fSel) = (*(gpio + fSel) & ~(7 << shift));           ——设置为输入IO
*(gpio + fSel) = (*(gpio + fSel) & ~(7 << shift)) | (1 << shift); ——设置为输出IO
```

Bit(s)	Field Name	Description	Type	Reset
31-30	---	Reserved	R	0
29-27	FSEL19	FSEL19 - Function Select 19 000 = GPIO Pin 19 is an input 001 = GPIO Pin 19 is an output 100 = GPIO Pin 19 takes alternate function 0 101 = GPIO Pin 19 takes alternate function 1 110 = GPIO Pin 19 takes alternate function 2 111 = GPIO Pin 19 takes alternate function 3 011 = GPIO Pin 19 takes alternate function 4 010 = GPIO Pin 19 takes alternate function 5	R/W	0
26-24	FSEL18	FSEL18 - Function Select 18	R/W	0
23-21	FSEL17	FSEL17 - Function Select 17	R/W	0
20-18	FSEL16	FSEL16 - Function Select 16	R/W	0

图4 BCM2835 FSEL寄存器含义

4.3 digitalWrite

```
[cpp]
01. void digitalWrite (int pin, int value)
02. {
03.     struct wiringPiNodeStruct *node = wiringPiNodes ;
04.     // 树莓派 板载GPIO设置，板载GPIO的引脚编号必须小于64
05.     if ((pin & PI_GPIO_MASK) == 0)
06.     {
07.         // 第一步 确定BCM GPIO引脚编号
08.         if (wiringPiMode == WPI_MODE_PINS)
09.             pin = pinToGpio [pin] ;
10.
11.         // 第二步 设置高电平和低电平
12.         if (value == LOW)
13.             *(gpio + gpioToGPCLR [pin]) = 1 << (pin & 31) ;
14.         else
15.             *(gpio + gpioToGPSET [pin]) = 1 << (pin & 31) ;
16.     }
17.     else
18.     {
19.         if ((node = wiringPiFindNode (pin)) != NULL)
20.             node->digitalWrite (node, pin, value) ;
21.     }
22. }
```

该部分代码的实现可以分为两步（注意该部分并不是digitalWrite 的完整代码，为了说明问题对代码进行简化）

第一步，确定BCM GPIO引脚编号。

第二步，设置高电平和低电平。该步骤用于设置GPCLR寄存器和GPSET寄存器。BCM GPIO0到GPIO31 位于GPIO Output Set Register 0，相对于GPIO_BASE的偏移量为7，而BCM GPIO32到GPIO53 位于GPIO Output Set Register 1，相对于GPIO_BASE的偏移量为8。例如操作wiringPi的GPIO0，对应BCM GPIO17；查找gpioToGPSET表可获得GPIO17位于GPIO Output Set Register 0寄存器，该寄存器的偏移量（相对于GPIO_BASE）为7。通过*(gpio + gpioToGPSET [pin]) = 1 << (pin & 31)，便可设置GPIO17为输出高电平。

Bit(s)	Field Name	Description	Type	Reset
31-0	SETn (n=0..31)	0 = No effect 1 = Set GPIO pin <i>n</i>	R/W	0

Table 6-8 – GPIO Output Set Register 0

Bit(s)	Field Name	Description	Type	Reset
31-22	-	Reserved	R	0
21-0	SETn (n=32..53)	0 = No effect 1 = Set GPIO pin <i>n</i> .	R/W	0

Table 6-9 – GPIO Output Set Register 1

图5 BCM2835 SET寄存器含义

4.4 delay

```
[cpp]
01. void delay (unsigned int howLong)
02. {
03.     struct timespec sleeper, dummy ;
04.     sleeper.tv_sec = (time_t)(howLong / 1000) ;
05.     sleeper.tv_nsec = (long)(howLong % 1000) * 1000000 ;
06.     nanosleep (&sleeper, &dummy) ;
}
```

```
07. } }
```

delay是wiringPi提供的一个毫秒级别的延时函数，该函数通过nanosleep实现。nanosleep的声明如下：

```
[cpp] C P
01. #include <time.h>
02. int nanosleep(const struct timespec *req, struct timespec *rem);
```

调用nanosleep使得进程挂起，直到req所设定的时间耗尽。在该函数中，req至进程最终休眠的时间而rem只剩余的休眠时间，struct timespec结构体的定义如下，

```
[cpp] C P
01. struct timespec {
02.     time_t tv_sec; /* 秒 */
03.     long tv_nsec; /* 纳秒 */
04. };
```

从结构体的成员来说，nanosleep似乎可以实现纳秒级别的延时，但是受到linux时钟精度的影响无法实现纳秒级别的延时，但是微妙级别的延时也可以让人接受。

5.总结

深入分析wiringPi之后收获颇丰。wiringPi可通过两种方式实现GPIO的驱动——第一，在用户空间直接操作寄存器（RAM），在用户空间操作寄存器（RAM）需要借助 /dev/mem；第二，利用/sys/class/gpio，通过操作文件的方式控制GPIO。在wiringPi中pin编号小于64为板载设备，例如GPIO0到GPIO7为板载设备，pin编号大于64为扩展设备，例如扩展的AD和DA芯片。最后可以使用nanosleep实现定时休眠。

未来将利用wiringPi实现SPI和I2C设备。

6.参考资料和博文链接

- 6.1 【[elinux 树莓派外设驱动开发指南](#)】
- 6.2 【[树莓派 GPIO入门指南](#)】

- [上一篇](#) CoAP学习笔记——服务器端繁忙时的处理请求流程
- [下一篇](#) 树莓派学习笔记——wiringPi I2C设备使用详解

顶

2

踩


0

主题推荐 文件系统 驱动开发 操作系统 内存 扩展

猜你在找

- 【精品课程】C语言基础视频教程
- 【精品课程】Apple Watch开发入门
- 【精品课程】VRP系统使用、维护...
- 【精品课程】iOS即时通讯(IM)开...
- 【精品课程】Cocos2d-Lua进阶篇...

查看评论

JillLOL


13楼 2015-03-20 10:58发表

博主，非常喜欢你叙述事情的方式，清晰有条理。请问你有没有更详细的关于WiringPi SPI的详解呢？我试图用Raspberry Pi连接Timing Controller，但是不成功，只能接收到一连串的0。另外，我用的pi用BCM2708。这个和上文提到的有什么区别，在运用是需要注意些什么吗？谢谢。



回复JHLLLOL：注意CS端口连接，使用CE0或者CE1

Re: 2015-03-20 17:25发表



sinat_26309553

12楼 2015-03-04 13:16发表


楼主，我认真研究了一下。百度了一下bcm2835的英文原版说明，上面那个映射到虚拟地址应该是错误的。Peripherals (at physical address 0x20000000 on) are mapped into the kernel virtual address space starting at address 0xF2000000. Thus a peripheral advertised here at bus address0x7Ennnnnn is available in the ARM kenel at virtual address 0xF2nnnnnn. ²外设（物理地址0x20000000）被映射到内核的虚拟地址空间的起始地址0xF2000000。因此，外设物理地址0x7E000000映射到虚拟地址应该是0xF2000000。 ³不知道我理解的对不对



xukai871105

Re: 2015-03-05 09:05发表


回复sinat_26309553：非常感谢，我再认真阅读BCM2835的手册和wiringPi的源代码。发现问题，及时修正博文中的错误。



RainSmell7

11楼 2015-01-19 23:57发表


地址映射有点不对好像，我试过Addr = (uint32_t)ioremap(BCM2835_GPIO_BASE, 16)返回值是0xF2200000，其中BCM2835_GPIO_BASE在官方bcm2835.h中定义为0x22000000，也就是说GPIO映射后的地址是0xF2200000而不是0x7E200000。



number007cool

10楼 2014-12-01 15:36发表


你好 请教下
在树莓派下进行linux驱动开发的卡发环境怎样搭建？
不需要进行交叉编译等操作么
程序在电脑上编译好后，怎样传到板子上执行呢



qq_23181573

9楼 2014-11-21 19:40发表


看到了博主给的教程，非常不错。学了不少东西；但是有个问题是wiringPi 库是用到指令行中的，如何写个窗口程序来控制GPIO口呢？ 第二我我现在在学校QT4编程，可以做窗口的软件编程；如果要是用qt4写软件的话，能否调用wiringpi库呢？ 我的email： zl_diy@163.com 我很少用微博，也不会用，今天看到了，也说不好那找不到我给你的留言。谢谢。



xukai871105

Re: 2014-11-27 11:10发表


回复qq_23181573：我并没有使用过QT4，如果有时间使用的话我研究一下。我建议你使用网页，因为可以达到使用QT一样的效果，开发也更灵活些。



qq_16707531

8楼 2014-07-21 15:21发表

博主你好啊，像你的这些代码都是在什么上面写的？是在RPI上就有一个环境还是在PC上写的



xukai871105

Re: 2014-07-21 17:29发表


回复qq_16707531：可以这样，在运行windows的PC上写好代码，然后FTP到树莓派中，在树莓派上编译。wiringpi比较适合这种方法。或者在运行linux上的电脑上写好代码，然后交叉编译，把可执行文件FTP到树莓派中。wiringPi是否合适，我还要尝试一下，主要是正确安装wiringPi的库，然后交叉编译时可以找到。你看看我的博客吧，有交叉编译和makefile的文章



zuoyioo7

7楼 2014-02-26 21:18发表


真是太佩服博主了！



xukai871105

Re: 2014-02-27 08:36发表


回复zuoyioo7：为什么这么说呢！



zuoyioo7

Re: 2014-03-03 21:41发表

回复xukai871105：呵呵...我看您在博客中介绍自己是一个学机械的，现在能掌握这些真是很厉害啊！



xukai871105

Re: 2014-03-04 08:54发表

回复zuoyioo7：见笑了，你可以比我做的更好的，期待你的博客了！



tianruoqiwo














6楼 2014-01-14 12:36发表

很漂亮，加油！



xukai871105

Re: 2014-01-14 12:37发表

回复tianruoqiwo：后面还会持续跟进树莓派！		
	<div><div>xukai871105</div><div>这个肯定不是裸奔，至少我用了nanosleep，这个是内核提供的定时函数。树莓派并不存在裸奔的概念，只是wiringPi的作者为了实现GPIO驱动，使用了在用户空间操作寄存器的方法；如果使用wiringPiSetupSys，那么就是操作文件的方式控制GPIO，而这个GPIO驱动是树莓派操作系统自带的。</div></div>	5楼 2014-01-06 08:59发表
	<div><div>cortex1990</div><div>看完了，这是在裸奔？</div></div>	4楼 2014-01-05 23:57发表
	<div><div>cortex1990</div><div>被你搞晕了</div></div>	3楼 2014-01-05 23:48发表
	<div><div>xukai871105</div><div>回复cortex1990：哪里被搞糊涂了，如果内容不合适或者章节有问题，我很愿意修改。要写就好好写，写的大家都明白，如果看了不明白，我想一定是我的问题！</div></div>	Re: 2014-01-06 09:00发表
	<div><div>cortex1990</div><div>博主，第一张图片上的地址标错了吧</div></div>	2楼 2014-01-05 23:48发表
	<div><div>xukai871105</div><div>回复cortex1990：图片中的错误已修改</div></div>	Re: 2014-01-07 20:17发表
	<div><div>xukai871105</div><div>回复cortex1990：的确，我少写了一个零，这个要到今天晚上才可以修好了！有问题，我立刻更正。</div></div>	Re: 2014-01-06 08:56发表
	<div><div>cortex1990</div><div>回复xukai871105：0x20000000 图片上的也错了吧 嘿嘿</div></div>	Re: 2014-01-07 23:19发表
	<div><div>cortex1990</div><div>回复cortex1990：博主 我觉得GPIO相关操作的起始地址应该是0xF2200000啊，最右边的那个灰色的映射图才是 Linux 使用的虚拟地址吧，上面IO外设的基址是0xF2000000。有点迷糊。</div></div>	Re: 2014-01-08 00:52发表
	<div><div>xukai871105</div><div>回复cortex1990：外设的物理起始地址为0x7E000000，被MMU虚拟之后的起始地址为0x20000000。这里的外设包括很多，例如SPI UART GPIO等等，GPIO的物理起始地址在这个基础上（0x7E000000）偏移0x200000，物理地址映射到虚拟地址，GPIO外设在这个基础上（0x2000000）偏移0x200000。偏移量是不变的。</div></div>	Re: 2014-01-08 08:52发表
	<div><div>啊树</div><div>回复xukai871105：Software directly accessing peripherals must translate these addresses into physical or virtual addresses, as described above. Software accessing peripherals using the DMA engines must use bus address</div><div>外设的物理起始地址为0x20000000，而0x7E000000是VideoCore的MMU映射后的Bus地址</div></div>	Re: 2014-11-19 17:09发表
	<div><div>啊树</div><div>Physical addresses range from 0x20000000 to 0x20FFFFFF for peripherals. The bus addresses for peripherals are set up to map onto the peripheral bus address range starting at 0x7E000000. Thus a peripheral advertised here at bus address 0x7Ennnnnn is available at physical address 0x20nnnnnn</div></div>	Re: 2014-11-19 16:59发表
	<div><div>cortex1990</div><div>博主，阿莫论坛有我给你留的消息，请务必指点一下我哈。多谢</div></div>	1楼 2014-01-05 18:48发表



回复cortex1990：树莓派主板+转接板，初次之外还可以购买 USB wifi设备，如果你需要一些扩展设备的话，可以在这家淘宝店里面找找——<http://dblsm.tmall.com/>。我在这家店里面购买了，PCF8574 IO扩展板，但是MC23017 MCP3208我都自己制作！

Re: 2014-01-05 20:39发表

您还没有登录,请[登录](#)或[注册](#)

* 以上用户言论只代表其个人观点，不代表CSDN网站的观点或立场

核心技术类目

全部主题	Hadoop	AWS	移动游戏	Java	Android	iOS	Swift	智能硬件	Docker	OpenStack	VPN	Spark	ERP
IE10	Eclipse	CRM	JavaScript	数据库	Ubuntu	NFC	WAP	jQuery	BI	HTML5	Spring	Apache	.NET
HTML	SDK	IIS	Fedora	XML	LBS	Unity	Splashtop	UML	components	Windows Mobile	Rails	QEMU	KDI
Cassandra	CloudStack	FTC	coremail	OPhone	CouchBase	云计算	iOS6	Rackspace	Web App	SpringSide	Maas		
Compuware	大数据	aptech	Perl	Tornado	Ruby	Hibernate	ThinkPHP	HBase	Pure	Soir	Angular	Cloud Foundry	
Redis	Scala	Django	Bootstrap										