Appplication Note

# AN070702

## PCA9698 芯片的应用

Rev 1.0 Date:8/25/2009

产品应用笔记

文件信息

| 类别 | 内容 |
| --- | --- |
| 关键词 | PCA9698、寄存器、编程 |
| 摘要 | 以 LPC2000 系列的 ARM 为例，讲述 PCA9698 芯片的应用解决方案 |

修订历史

| 版本 | 日期 | 原因 |
|---|---|---|
| Rev 1.0 | 2007/04/17 | 创建 |
| | | |
| | | |
| | | |

# 产品制造商与技术支持

公司：广州致远电子有限公司嵌入式系统事业部
地址：广州市天河区车陂路黄洲工业区三栋二楼（售后服务）
邮编：510660
电话：(020) 22644249　22644399（销售服务）
(020) 28872347　（技术支持）
传真：(020) 38601859
E–mail：　lpc2103@ zlgmcu.com　（技术支持）
网址：www.embedtools.com

# 销售与服务网络

广州周立功单片机发展有限公司
地址：广州市天河北路 689 号光大银行大厦 15 楼 F1　邮编：510630
电话：(020)38730916　38730917　38730976　38730977
传真：(020)38730925
网址：http://www.zlgmcu.com

广州专卖店
地址：广州市天河区新赛格电子城 203-204 室
电话：(020)87578634　87569914
传真：(020)87578842

南京周立功
地址：南京市珠江路 280 号珠江大厦 2006 室
电话：(025)83613221　83613271　83603500
传真：(025)83613271

北京周立功
地址：北京市海淀区知春路 113 号银网中心 715 室（中发电子市场斜对面）
电话：(010)62536178　62536179　82628073
传真：(010)82614433

重庆周立功
地址：重庆市石桥铺科园一路二号大西洋国际大厦（赛格电子市场）1115 室
电话：(023)68796438　68796439
传真：(023)68796439

杭州周立功
地址：杭州市登云路 428 号浙江时代电子市场 205 号
电话：(0571)88009205　88009932　88009933
传真：(0571)88009204

成都周立功
地址：成都市一环路南一段 57 号金城大厦 612 室
电话：(028)85399320　85437446
传真：(028)85439505

深圳周立功
地址：深圳市深南中路 2070 号电子科技大厦 A 座 24 楼 2403 室
电话：(0755)83781768　83781788　83782922
传真：(0755)83793285

武汉周立功
地址：武汉市洪山区广埠屯珞瑜路 158 号 12128 室（华中电脑数码市场）
电话：(027)87168497　87168297　87168397
传真：(027)87163755

上海周立功
地址：上海市北京东路 668 号科技京城东座 7E 室
电话：(021)53083452　53083453　53083496
传真：(021)53083491

西安办事处
地址：西安市长安北路 54 号太平洋大厦 1201 室
电话：(029)87881296　83063000　85399492
传真：(029)87880865

# 目录

# 1. 适用范围

此应用笔记适用于 LPC2000 系列 ARM 控制 NXP 公司生产的 PCA9698 芯片。

# 2. PCA9698 概述

## 2.1 特性

- 1MHz 的高速 $I^2C$ 总线接口模式；
- 支持高速（400K）和标准（100K）$I^2C$ 总线速率；
- 工作电源电压 2.3V~5.5V，I/O 口可承受 5.5V 电压；
- 40 个 I/O 口，复位后默认为输入；
- 低电平有效中断输出；
- 内部上电复位；
- 具有唯一的 ID 号；
- 小于 3uA 的低待机电流；
- 温度操作范围为-40℃～+85℃；
- 具有 ESD 保护（超出 JESD22-A114 2000V HBM ，JESD22-A115 200V MM 和 JESD22-C101 1000V CDMA）；
- 根据 JESDEC 标准 JESD78 所做的锁定测试超过 100mA；
- 提供两种封装:TSSOP56 和 HVQFN56。

## 2.2 描述

PCA9698 是 56 脚的 CMOS 器件，能够实现 $I^2C$/SMBus 应用中 40 位通用 GPIO 的扩展。改进的特性包括 4000pF 的驱动能力、5V I/O 口、工作电流低于 1mA、单独的 I/O 口配置、400kHz $I^2C$ 总线时钟频率和更小的封装形式。当应用中需要额外的 I/O 口来连接 ACPI（“高级配置与电源接口“这是英特尔、微软和东芝共同开发的一种电源管理标准）电源开关、传感器、按钮、LED、 风扇等时，可使用 I/O 扩展器件实现简单的解决方案。

PCA9698 包含五个 I/O 口设置寄存器和三个控制寄存器，系统主控器通过写 I/O 口相应的配置位来激活端口的输入或输出。当任何输入口状态发生改变时，激活 PCA9698 中断，中断引脚输出低电平，也就是说，该中断用来指示输入引脚的电平状态发生了改变。上电复位，将所有寄存器设置成默认值并初始化器件内部的状态机。在器件响应“GPIO ALL CALL”命令的模式下，可同时编程多个不同 $I^2C$ 总线地址的器件。PCA9698 有 3 个地址管脚 AD0 、AD1、AD2，可以实现不同的 $I^2C$ 地址，最多允许 64 个器件同时挂接在 $I^2C$/SMBus 总线上。

## 2.3 管脚配置

TSSOP 封装如图 2.1 所示。

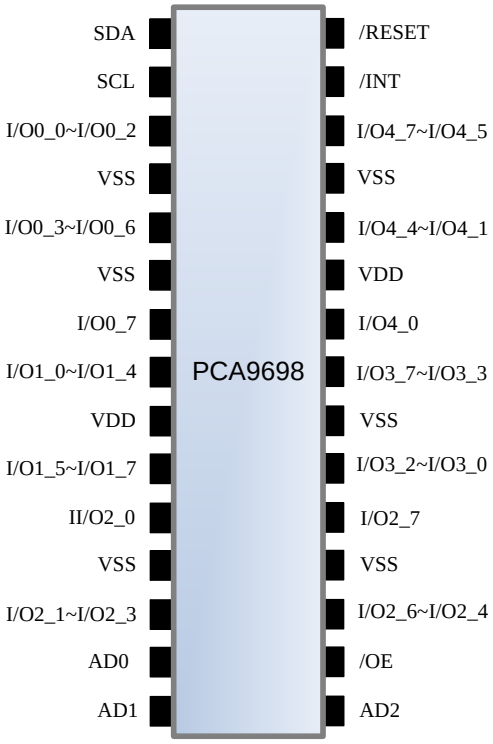


图 2.1 PCA9698 的 TSSOP 封装引脚排列图

## 2.4 管脚描述

PCA9698 的管脚描述如表 2.1 所示。

表 2.1 引脚功能说明

| 符号 | 引脚 | 类型 | 描述 |
|---|---|---|---|
| | TSSOP | | |
| SDA | 1 | 输入/输出 | 串行数据线 |
| SCL | 2 | 输入 | 串行时钟线 |
| IO0_0 to IO0_7 | 3,4,5,7,8,9,10,12 | 输入/输出 | 输入/输出端口 0 |
| IO1_0 to IO1_7 | 13,14,15,16,17,19,20,21 | 输入/输出 | 输入/输出端口 1 |
| IO2_0 to IO2_7 | 22,24,25,26,31,32,33,35 | 输入/输出 | 输入/输出端口 2 |
| IO3_0 to IO3_7 | 36,37,38,40,41,42,43,44 | 输入/输出 | 输入/输出端口 3 |
| IO4_0 to IO4_7 | 45,47,48,49,50,52,53,54 | 输入/输出 | 输入/输出端口 4 |
| VSS | 6,11,23,34,39,51 | 供给电源 | 提供地 |
| VDD | 18,46 | 供给电源 | 提供电压 |
| AD0 | 27 | 输入 | 地址输入 0 |
| AD1 | 28 | 输入 | 地址输入 1 |
| AD2 | 29 | 输入 | 地址输入 2 |
| \OE | 30 | 输入 | 输出使能端，低电平有效 |
| \INT/\SMBALERT | 55 | 输出 | 中断/报警输出，低电平有效 |
| \RESET | 56 | 输入 | 复位输入，低电平有效 |

## 2.5 方框图

PCA9698 的内部结构方框图如图 2.2 所示。

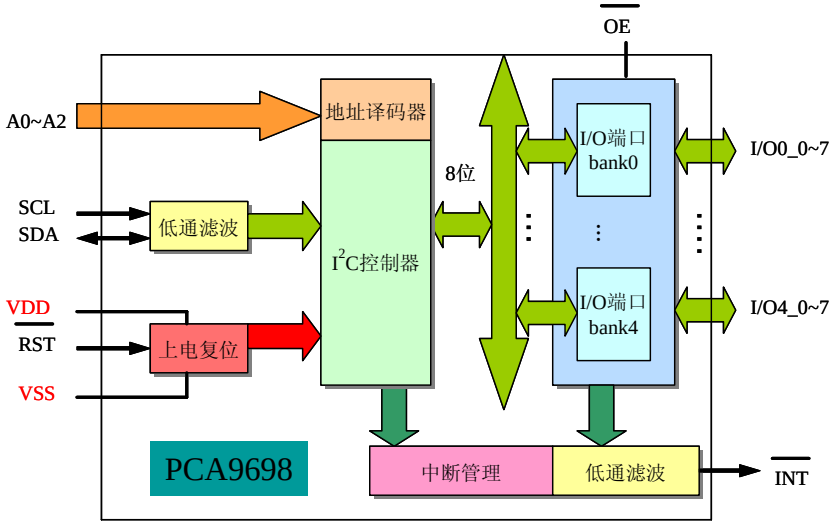


图 2.2 PCA9698 的内部结构图

## 2.6 器件从地址

在起始条件之后，主发送器必须发送它所访问的从器件地址和读/写操作位。PCA9698 的地址如图 2.3 所示，器件地址引脚 AD2、AD1、AD0 可以配置 64 个从地址，AD2、AD1、AD0 引脚为高阻模式，使用时不能悬空。如图 2.4 所示，地址引脚全部上拉，根据表 2.2，此时的从机地址为：0x4E，有关从机地址的详细配置信息请参考“PAC9698 数据手册”。

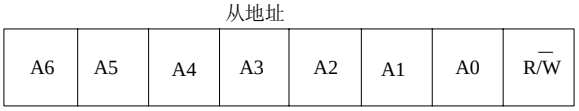


图 2.3 PCA9698 地址

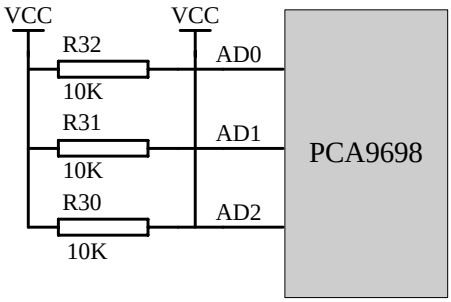


图 2.4 地址选择示意图

表 2.2 引脚的设置及对应地址

| AD2 | AD1 | AD0 | A6 | A5 | A4 | A3 | A2 | A1 | A0 | 地址 |
|---|---|---|---|---|---|---|---|---|---|---|
| VSS | VSS | VSS | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 40H |
| VDD | VDD | VDD | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 4EH |

**注意:** 若选择地址 40H，则需用杜绑线将图 2.4 中的 AD0、AD1、AD2 引脚与地连接;
若选择地址 4EH，则需用杜绑线将图 2.4 中的 AD0、AD1、AD2 引脚与电源连接。

## 2.7 “GPIO ALL CALL“地址

写这个地址允许几个高级 GPIO 器件同时编程，这个地址在执行写操作时经常被用到，地址示意图如图 2.5 所示。

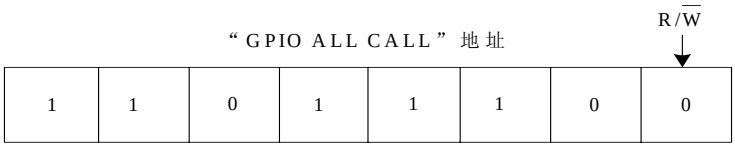


图 2.5 “GPIO ALL CALL“地址

## 2.8 寄存器

### 2.8.1 寄存器说明

PCA9698 芯片有五组 I/O 设置相关的寄存器和三个控制寄存器， 对寄存器的描述有如下 8 点，在描述中‘x’指的是某个寄存器（0~4），‘y’指的是寄存器中的某一位（0~7）：

1. I/O 口设置寄存器设置引脚的方向。
   - Cx[y]=0:相应的端口引脚为输出，如：若 $C_4$[7]=0，则是把 I/O4 口的第 8 个引脚设置为输出如图 2.6A 所示；
   - Cx[y]=1:相应的端口引脚为输入，如：若 $C_3$[7]=1，则是把 I/O3 口的第 8 个引脚设置为输入如图 2.6B 所示。

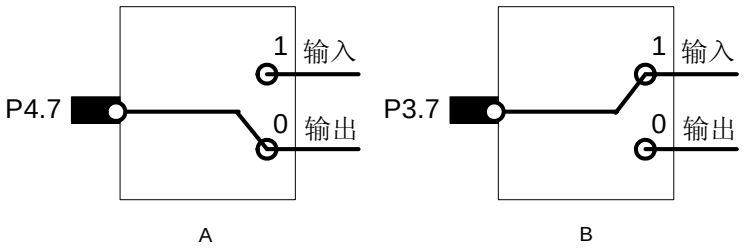


图 2.6 引脚功能设置

2. 输入端口寄存器为只读寄存器，不管引脚被定义为输入或输出，寄存器都会反应输入逻辑电平的状态。
3. 输出设置寄存器，引脚被设置为输出时，这些寄存器反应输出的逻辑电平。若引脚被设置为输入，则这些寄存器中的位不会反应引脚的状态。
4. 中断屏蔽寄存器，在 I/O 口被设置为输入且引脚状态发生改变时，这些寄存器屏蔽中断的产生。
   - Mx[y]=0：如果 I/O 口被设置为输入，当 I/O 口的状态发生改变时将会产生中断；如：若 $C_4$[7]=0，则 I/O4 口的第 8 个引脚的输入状态改变时将会产生中断如图 2.7A 所示；
   - Mx[y]=1：如果 I/O 口被设置为输入，当 I/O 口的状态发生改变时将不会产生中断。如：若 $C_4$[7]=1，则 I/O4 口的第 8 个引脚的输入状态改变时将会屏蔽中断如图 2.7B 所示。

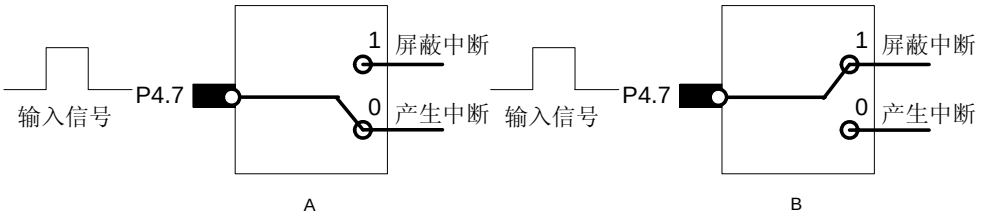


图 2.7 屏蔽中断

5. 极性反转寄存器，这些寄存器允许相应输入端口寄存器极性的反转。
   - Px[y]=0:相应输入端口寄存器数据的极性被保持；如：若 $C_4$[7]=0，则 I/O4 口的第 8 个引脚的输入状态保持如图 2.8A 所示；
   - Px[y]=1：相应输入端口寄存器数据的极性被反转。如：若 $C_4$[7]=1，则 I/O4 口的第 8 个引脚的输入状态被反转如图 2.8B 所示。

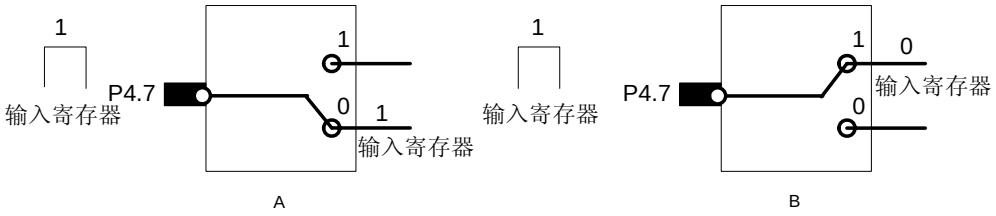


图 2.8 极性反转设置

6. 模式选择寄存器，允许编程 PCA9698 的模式，寄存器的描述如
7. 表 2.3 所示。

表 2.3 模式选择寄存器

| 位 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| 符号 | X | X | X | SMBA | IOAC | X | OCH | OEPOL |
| 默认设置 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |

- OEPOL 位控制/OE 引脚的极性；
  - OEPOL=0：/OE 引脚低电平有效；
  - OEPOL=1：/OE 引脚高电平有效。
- OCH 位决定输出端口在什么状态下改变时选择 $I^2C$ 总线事件；
  - OCH=0：在停止命令时输出发生改变；
  - OCH=1：在应答时输出发生改变。
- IOAC 位控制器件响应“GPIO ALL CALL”命令的能力，允许同时编程多于一个的器件；
  - IOAC=0：这个器件不会响应“GPIO ALL CALL”命令；
  - IOAC=1：这个器件会响应“GPIO ALL CALL”命令。
- SMBA 控制 PCA9698 响应 SMBAlert 命令的能力；
  - SMBA=0：PCA9698 不会响应警报反应的地址。
  - SMBA=1：PCA9698 响应警报反应的地址，5、6、7 位被保留并编程为 0；

8. “ALLBANK“寄存器，允许所有的 I/O 口设置为输出时输出相同的逻辑电平值，这种设置适应于所有的组或所选的组如表 2.4 所示。

表 2.4 控制全组寄存器

| 位 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| 符号 | BSEL | X | X | B4 | B3 | B2 | B1 | B0 |

| 默认设置 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
|---|---|---|---|---|---|---|---|---|

- 若 BSEL=0：
  - Bx=0:被设置为输出的相对应于 x 组的所有 I/O 端口被编程为 0；
  - Bx=1:被设置为输出的相对应于 x 组的所有 I/O 端口被编程为相对应寄存器的真实的值。
- 若 BSEL=1：
  - Bx=0: 被设置为输出的相对应于 x 组的所有 I/O 端口被编程为相对应寄存器的真实的值；
  - Bx=1:被设置为输出的相对应于 x 组的所有 I/O 端口被编程为 1。

9. 输出结构配置寄存器，控制输出端口的配置为开漏输出或图腾柱输出。

### 2.8.2 寄存器对应的地址

在编程时，对各个寄存器的操作是通过相应地址的操作来实现的，各个寄存器的地址如表 2.5 所示。

表 2.5 寄存器对应的地址

| 寄存器类 | 地址 | 名称 | 描述 |
|---|---|---|---|
| I/O 口设置寄存器 | 00h~04h | 输入端口寄存器 0~4 | 只读寄存器，反应输入逻辑电平的状态 |
| | 08h~0Ch | 输出端口寄存器 0~4 | 设置相应端口的输出状态 |
| | 10h~14h | 极性反转寄存器 0~4 | 使相应的输出寄存器的极性发生反转 |
| | 18h~1Ch | I/O 口配置寄存 0~ 4 | 设置 I/O 口为输出或输入 |
| | 20h~24h | 中断屏蔽寄存器 0~ 4 | 设置相应输入端口响应中断 |
| 控制寄存器 | 28h | 输出结构配置寄存器 | 控制输出端口的配置为开漏输出或图腾输出 |
| | 29h | 控制全组寄存器 | 所有的 I/O 口设置为输出时输出相同的逻辑电平值 |
| | 2Ah | 模式选择寄存器 | 有 16 种模式可供选择 |

# 3. LPC2000 系列 ARM 的 I²C 接口

由于 LPC2000 系列 ARM 的 I²C 是硬件 I²C，需要操作的寄存器比较多，并且要按照严格的 I²C 时序进行，因此需要一个 I²C 软件包。串行传输总线以两根连线实现了完善的全双工同步数据传送，其传输协议和软件包如下。

## 3.1 I²C 传输协议

主机产生起始信号后，发送一个寻址字节，收到应答后紧跟着的就是数据传输，数据传输一般由主机产生的停止位终止。但是，如果主机仍希望在总线上通讯，它可以产生重复起始信号和寻址另一个从机，而不是首先产生一个停止信号。在这种传输中，可能有不同的读/写格式结合。可能的数据传输格式有：

- **主机发送数据到从机。**寻址字节的读或写位为 0，数据传输的方向不改变，主机发送从地址—数据流②，之后从机发送应答信号—数据流③，然后主机每发送一字节数据—数据流④、⑥，从机发送应答信号—数据流⑤、⑦，直到主机发送停止信号—数据流⑧，结束数据通信，如图 3.1 所示。

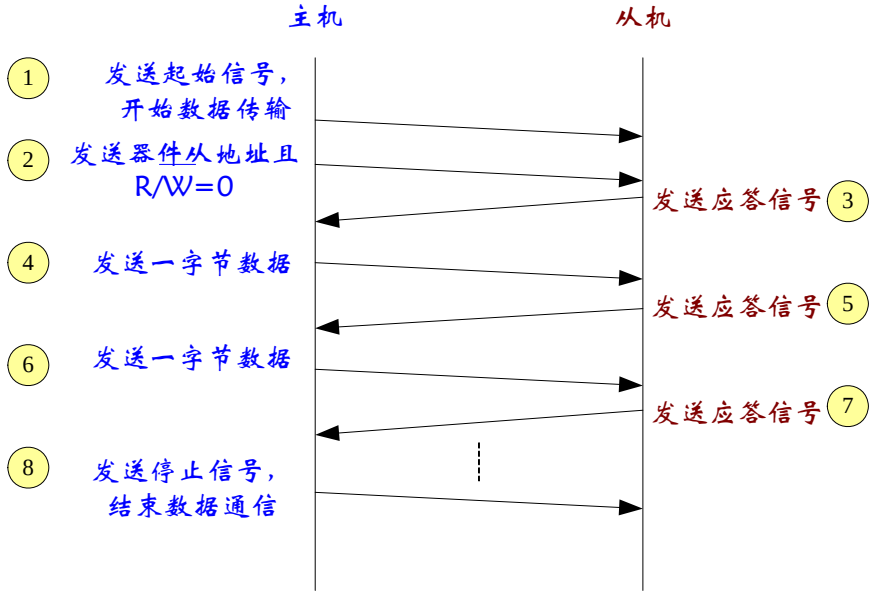


图 3.1 主机发送数据到从机

- **主机读取从机中的数据。**主机发送起始信号—数据流①，主机发送完寻址字节—数据流②后，主机立即读取从机中的数据。如图 3.2 所示，寻址字节的读或写位为 1，在从机产生的响应—数据流③后，主机发送器变成主机接收器，从机接收器变成从机发送器。之后，从机发送数据—数据流④、⑥，主机发送应答信号—数据流⑤、⑦，直到主机发送非应答信号—数据流⑧，接着发送停止信号—数据流⑨，则终止本次传输。

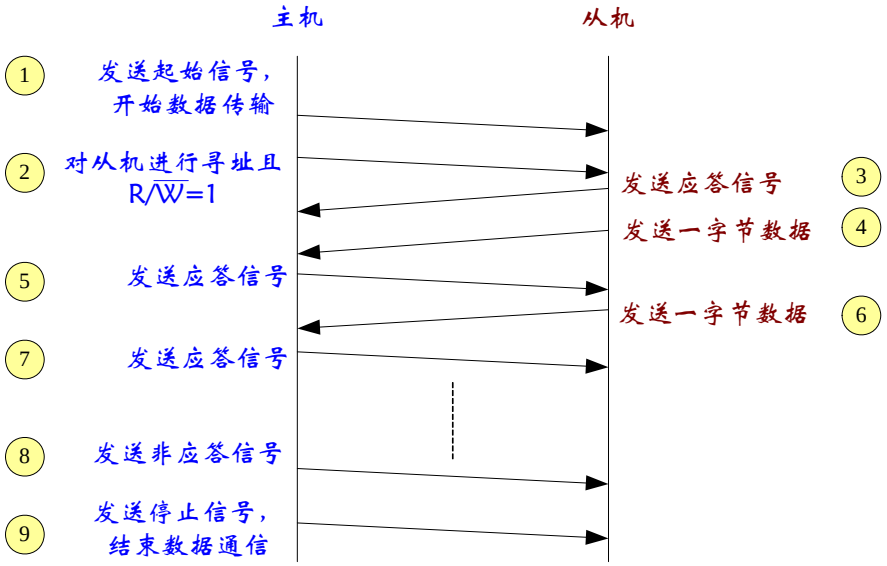


图 3.2 主机读取从机中的数据

## 3.2 I²C 软件包

应用 I²C 软件包，只须简单地调用 API 函数即可对 I²C 总线进行访问，无需细察底层寄存器和中断操作。

I²C 软件包提供给用户调用的 API 函数如表 3.1 所示。

表 3.1 I²C 软件包 API 函数

| 函数名称 | 函数功能描述 |
|---|---|
| I2C_Init | I²C 接口初始化 |
| I2C_SendByte | 向无子地址器件写入单字节数据 |
| I2C_RcvByte | 向有子地址器件读取单字节数据 |
| I2C_WriteNByte | 向有子地址器件写入 N 字节数据 |
| I2C_ReadNByte | 从有子地址器件任意地开始读取 N 字节数据 |

对总线进行读或写操作，只需调用软件包中的函数即可，软件包中所要使用的函数说明如

程序清单 3.1 所示。但在此之前需要在工程中添加“I2C.c”和“I2C.h”文件如图 3.3 所示，并在主程序中添加语句“include I2C.h”。

程序清单 3.1 软件包中各函数说明

```
/*********************************************************************************************
** 函数名称：  void  I2C_Init(uint8 n,uint8 MODE,uint32 Fi2c,uint8 Adr,uint8 slot)
** 函数功能：  初始化 I2C 接口
** 入口参数：  n       ：I2C 接口号,0—I2C0，1—I2C1
**             MODE    ：工作模式，0—从模式，1—主模式
**             Fi2c    ：I2C 通信速率,0~400K,如果超过 400K,则会强制设置为 400KHz,如果设置为从
**                       机,该参数无效
**             Adr     ：当设置为从模式时,Adr 表示从地址,在主模式下,该参数是无效的,可以任意设置
**             slot    ：由于 I2C 采用 IRQ 中断方式,所以需要指定对应的通道,0~15
** 出口参数：  1—接口初始化成功,0—接口初始化失败
** 说    明：  初始化函数会将 I2Cn 的中断设置为 IRQ 中断,并分为 slot0
```

```
*********************************************************************************************/
uint8 I2C_Init(uint8 n,uint8 MODE,uint32 Fi2c,uint8 Adr,uint8 slot)
{
    .........
}
/*********************************************************************************************
** 函数名称:  uint8 I2C_WriteNByte(uint8 n,uint8 sla,uint32 suba_type,uint32 suba,uint8 *s,uint32 num)
** 函数功能:  向有子地址器件写入 N 字节数据
** 入口参数:  n          : 接口号 0,1
**            sla        : 器件从地址
**            suba_type  : 子地址结构  1－单字节地址，2－双字节地址，3－8＋X 结构
**            suba       : 器件子地址
**            *s         : 数据发送缓冲区指针
**            num        : 要写入的数据的个数
** 出口参数:  1          :    操作成功
**            0          :    操作失败
** 说    明:  程序死等待操作完成
*********************************************************************************************/
uint8 I2C_WriteNByte(uint8 n,uint8 sla,uint8 suba_type,uint32 suba,uint8 *s,uint32 num)
{
    .........
}
/*********************************************************************************************
** 函数名称:  uint8 I2C_ReadNByte(uint8 n,uint8 sla,uint32 suba_type,uint32 suba,uint8 *s,uint32 num)
** 函数功能:  从有子地址器件任意地址开始读取 N 字节数据
** 入口参数:  n              接口号 0,1
**            sla            器件从地址
**            suba_type      子地址结构     1－单字节地址，2－双字节地址，3－8＋X 结构
**            suba           器件子地址
**            *s             数据接收缓冲区指针
**            num            读取的个数
** 出口参数:  1              操作成功
**            0              操作失败
** 说    明:  程序死等待操作完成
*********************************************************************************************/
uint8 I2C_ReadNByte(uint8 n,uint8 sla,uint32 suba_type,uint32 suba,uint8 *s,uint32 num)
{
    .........
}
```

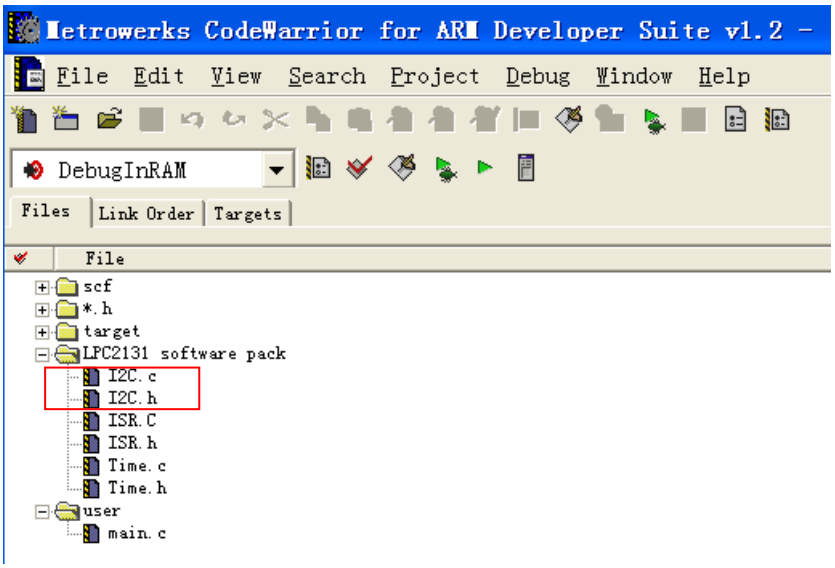


图 3.3 添加文件的工程

# 4. PCA9698 的应用解决方案

## 4.1 系统概述

本示例选用 LPC2000 系列 ARM 的 LPC2131 控制 PCA9698 芯片，整个系统可分为三部分：LPC2000 系列 ARM、PCA9698（NXP 的 I/O 扩展芯片）以及输入、输出电路，其系统整体框图如图 4.1 所示。

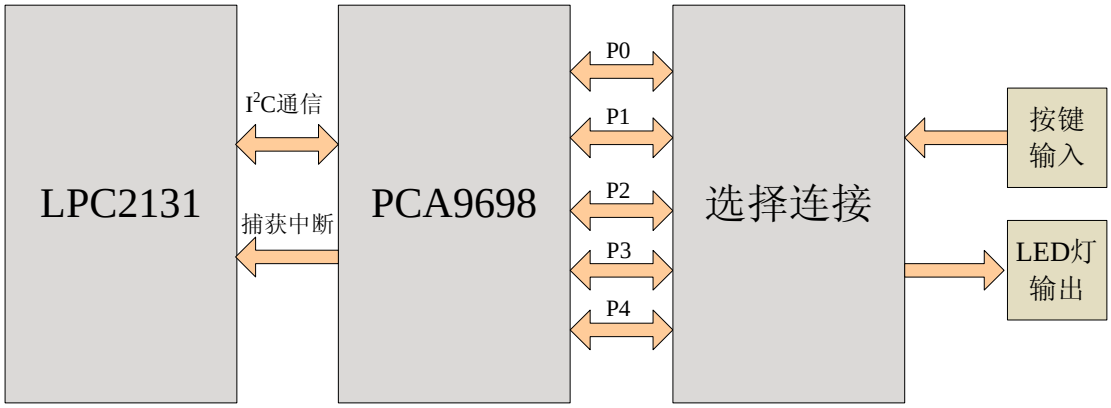


图 4.1 系统整体框图

**注意**：只需把 JP1、JP2 利用短路帽短接即可实现此案例硬件电路的连接。

## 4.2 硬件设计

### 4.2.1 PACK 板简介

电路板的实物如图 4.2 所示。

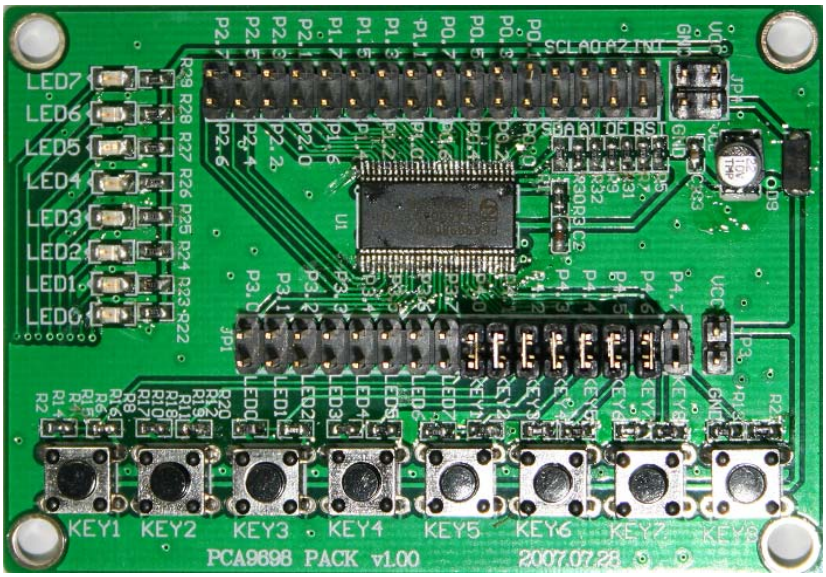


图 4.2 Demo 板实物图

### 4.2.2 输入、输出电路

在 PCA9698PACK 上，设计了相关的输出、输入电路。

- 电路采用了 I/O 口灌电流的驱动方式来驱动 LED，这样做主要是因为 I/O 口能提供的灌电流 20mA 大于其拉电流 8mA，保证了 LED 的显示亮度，电路如图 4.3 所示。

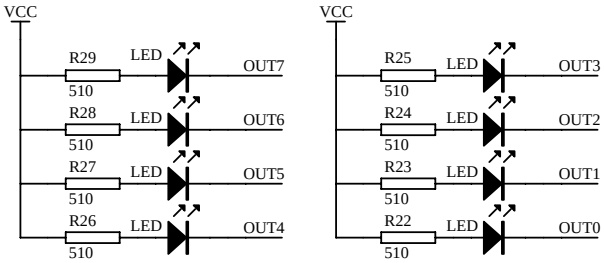


图 4.3 输出电路图

- I/O 口作为输入时为高阻模式，通常需要外部上拉，如图 4.4 所示。

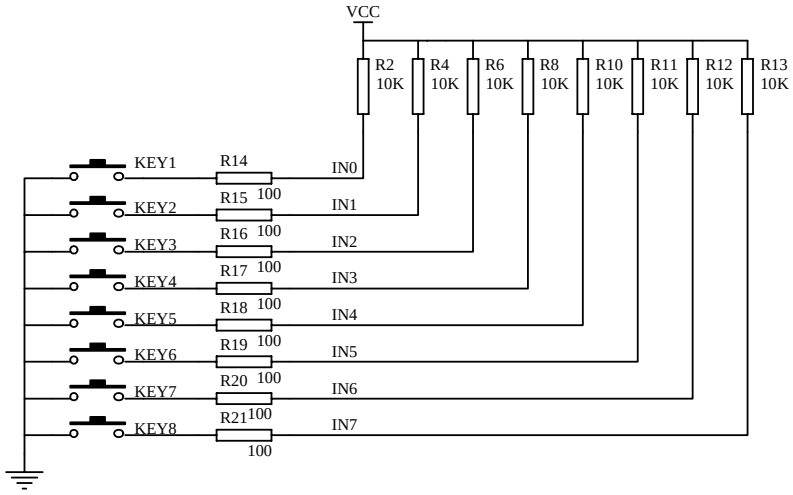


图 4.4 输入电路图

### 4.2.3 控制信号电路

- $I^2C$ 接口为开漏输出，用于 $I^2C$ 通信时需要加 1~10K 不等的上拉电阻，阻值大小视系统需要的传输速率而定。一般而言，所需的 $I^2C$ 通信速率越高，阻值应该越小。当使用标准 $I^2C$ 总线时，典型值为 3~5.1K，在此电路中设置为 3K 如图 4.5 所示。
- 输入监测信号、复位信号、输出使能信号都为低电平有效，通常需要外部上拉，如图 4.5 所示，PCA9698 PACK 中外接 10K 的上拉电阻。

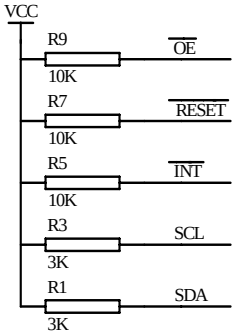

图 4.5 控制信号电路图

### 4.2.4 PCA9698 PACK 与 LPC2000 系列 ARM 硬件连接

在此次操作中，选用了 LPC2000 系列 ARM 做为 I²C 总线主机，PCA9698 做为从机，硬件连接电路参考图 4.6。

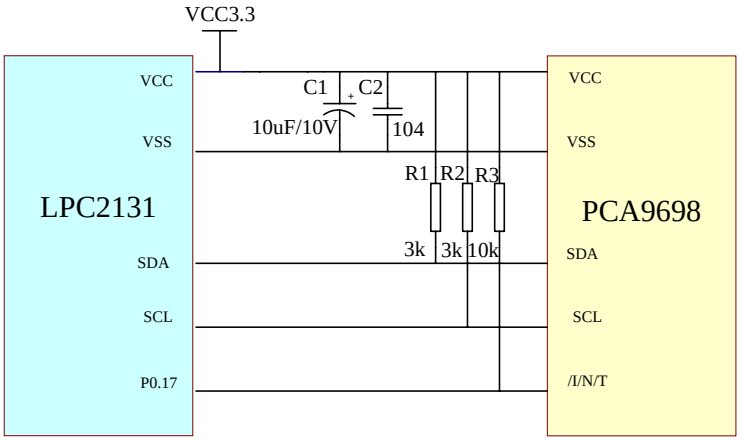


图 4.6 主从机连接图

## 4.3 软件设计

### 4.3.1 对芯片读、写操作说明

对芯片的读或写操作完全符合 I²C 软件包的时序，所以用户可以直接调用 I²C 软件包即可完成对芯片寄存器的读或写操作。

- 主机向芯片写入数据：寻址字节的读或写位为 0，数据传输的方向不改变，写入数据须按照数据流①→②→③→④→⑤→⑥→⑦→⑧进行操作，如图 4.7 所示；

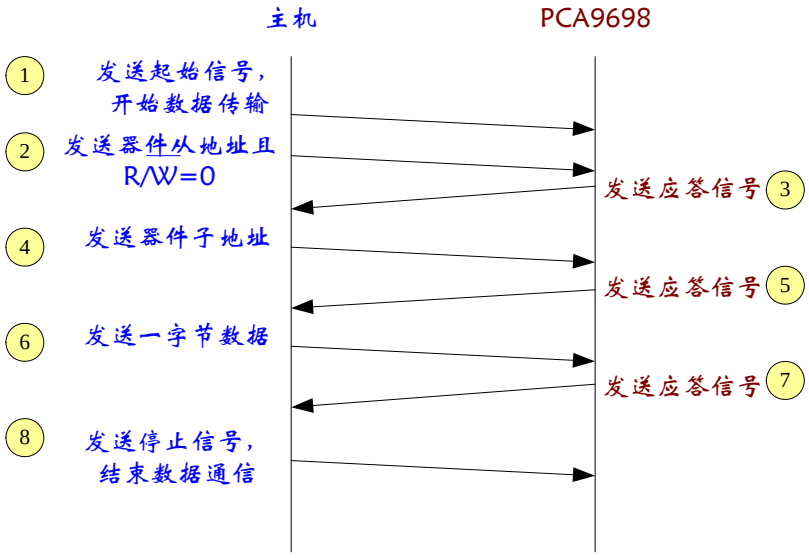


图 4.7 主机操作 PCA9698 寄存器

- 主机从芯片读取数据：主机发送起始信号—数据流①，主机发送完寻址字节—数据流②，从机发送应答信号—数据流③，之后主机发送器件子地址—数据流④，从机

发送应答信号—数据流⑤，接着主机立即读取从机中的数据如图 4.8 所示。读取一个字节后主机发送非应答信号—数据流⑧，接着发送停止信号—数据流⑨，则终止本次传输。

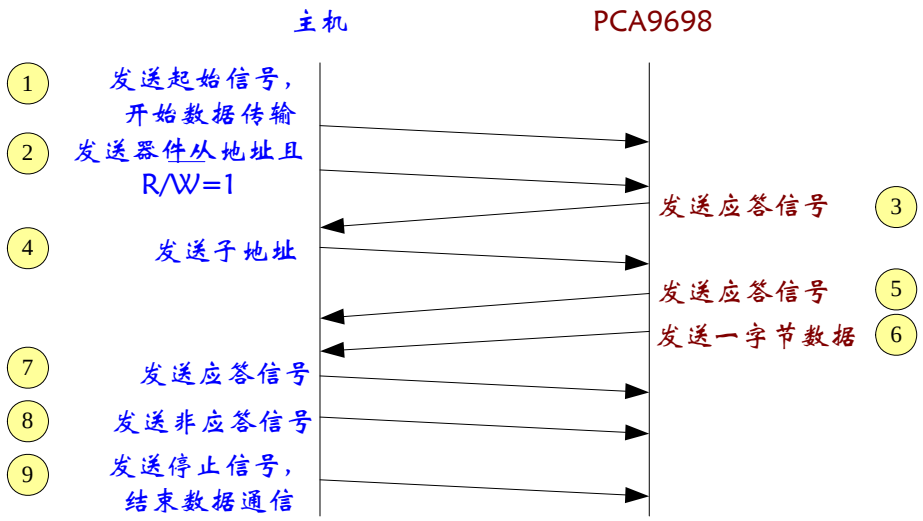


图 4.8 主机读取 PCA9698 寄存器中的数据

### 4.3.2 对 I/O 口设置相关的寄存器的读写应用

PCA9698 是 NXP 公司生产的 40 位通用 GPIO 的扩展，当应用中需要额外的 I/O 口来连接 ACPI（“高级配置与电源接口“这是英特尔、微软和东芝共同开发的一种电源管理标准）电源开关、传感器、按钮、LED、 风扇等时，可使用此器件实现简单的解决方案。

芯片 I/O 口使用的基本方法如下：

- 设置 I/O 口配置寄存器使能相应的 I/O 口为输出或输入；
- 设置为输出的 I/O 口，可以通过输出寄存器控制 I/O 口的电平状态；
- 设置为输入的 I/O 口，可以设置极性反转寄存器控制 I/O 的输入电平是否反转，并且可以通过中断屏蔽寄存器，设置 I/O 口的输入电平状态发生改变时是否产生中断。

**（1）I/O 口输出**

利用 I/O3 口的输出功能，控制 PACK 板上相应 LED 灯的亮灭。在 PACK 板上需用短路帽，将 I/O3 口与 LED 灯连接。根据本章中 4.2 节的硬件设计，I/O3 口采用灌电流的方式驱动 LED。本示例实现通过写 I/O3 口的输出寄存器，控制 LED 灯亮灭的功能如程序清单 4.1 所示。指令⑴的有关设置详见 2.6 节内容，指令⑵、⑶参照表 2.5，指令⑷～⑹详见 2.8.1 小节中的内容，指令⑺～⑽按照 4.3.1 小节中 $I^2C$ 的读、写进行操作。

程序清单 4.1 I/O 口输出控制

```
#include “config.h”
#include “I2C.h”
#define    PCA9698   0x4E          /*  PCA9698 器件从地址      */   (1)
#define    IOC3      0x1B          /*  I/O3 口设置寄存器地址   */   (2)
#define    OP3       0x0B          /*  I/O3 口输出寄存器地址   */   (3)

uint8 Gudata_IOC3[]  = {0x00};     /*  低电平为输出            */   (4)
uint8 Gudata _LEDC[] = {0Xff};     /*  高电平熄灭灯            */   (5)
```

```
uint8 Gudata _LED[]  = {0x55};                          /*  低电平点亮          */                (6)
/*********************************************************************************************
**  函数名称：main ()
**  函数功能：通过控制 I/O3 口的输出寄存器，点亮 1、3、5、7 灯
*********************************************************************************************/
int main (void)
{
    /*  I2C 初始化,总线速率 400Kb/s   */
    I2C_Init(0,1,400000,0,0);                                                              (7)
    /*  设置 I03 口为输出*/
    I2C_WriteNByte(0,PCA9698,ONE_BYTE_SUBA,IOC3, Gudata _IOC3,1);                          (8)
    /*  熄灭所有 LED 灯*/
    I2C_WriteNByte(0,PCA9698,ONE_BYTE_SUBA,OP3, Gudata _LEDC,1);                           (9)
    /* 1、3、5、7 灯亮   */
    I2C_WriteNByte(0,PCA9698,ONE_BYTE_SUBA,OP3, Gudata _LED,1);                            (10)
    while(1);
    return 0;
}
```

**（2）I/O 口输入**

利用 I/O4 口的输入，实现 PCA9698 的输入、中断和极性反转功能。在 PACK 板上需用短路帽，将 I/O3 口与 LED 灯连接，I/O4 口与按键连接。本示例是当按键 7 按下时产生中断，主机捕获到中断后，设置 I/O4 口的输入电平状态发生反转，利用反转后的值控制 LED 灯亮灭，其程序流程图如图 4.9 所示，程序清单 4.2 中指令⑴～⑶的设置详见 2.8.1 小节的内容。

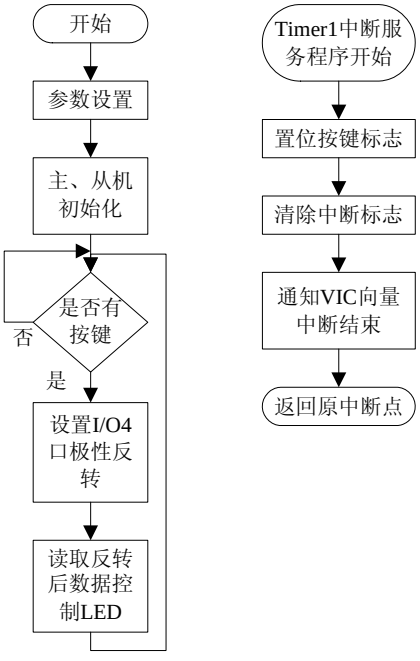


图 4.9　I/O 口输入流程图

程序清单 4.2　I/O 口输入控制

```
#include   “config.h”
#include   “I2C.h”
```

```
volatile uint8 flag; /*   键值变量   */
/*********************************************************************************************
** 函数名称：Timer1_CapInit()
** 函数功能：定时器 1 捕获中断服务程序
** 入口参数：无
** 出口参数：无
*********************************************************************************************/
void __irq Timer1_CapInit(void)
{
      Flag   =   1;                              /*   有输入变化置位按键标志   */
      T1IR   =   1<<6;                           /*   清除 CAP1.2 中断标志  */
      VICVectAddr   =   0x00;                    /*   向量中断处理结束   */
}
/*********************************************************************************************
** 函数名称：main()
** 函数功能：往 PCA9698 写入数据,按键控制灯亮灭
*********************************************************************************************/
int main (void)
{
      uint8 udata_IOC3[]   =   {0x00};           /*   设置 I/O 口为输出的配置寄存器 3 的数据    */
      uint8 uLED_Close[]   =   {0Xff};           /*   设置熄灭所有 LED 灯的数据                 */
      uint8 udata_IOC4[]   =   {0Xff};           /*   设置 I/O4 口为输入的数据                  */
      uint8 udata_MSK4[]   =   {0x00};           /*   设置 I/O4 口中断屏蔽寄存器输入的数据      */
      uint8 uRead_IP4[2]   =   {0x00,0x00};      /*   读取 PI4 端口数据的缓冲区                 */
      uint8 udata _ PI1 []   =   {0xFF};         /*   设置极性反转寄存器的值                    */
      uint8 uoutdata _ PI1 [1]   =   {0x00};     /*   读取极性反转后寄存器值的缓冲区            */

      PINSEL1   =   1<<2;                        /*   P0.17 连接捕获 1.2                        */
      I2C_Init(0,1,400000,0,0);                  /*   I2C 初始化,总线速率 400Kb/s               */
      T1PR   =   99;                             /*   分频系数                                  */
      T1CCR   =   (1<<7)|                        /*   设置 CAP1.2 下降沿捕获                    */
                  (1<<8);                        /*   允许产生中断                              */
      T1TC   =   0;
      T1TCR   =   0x01;                          /*   启动定时器                                */
      IRQEnable();                               /*   打开中断                                  */
      VICIntSelect   =   0x00000000;             /*   设置所有通道为 IRQ 中断                   */
      VICVectCntl1   =   (0x20|0x09);            /*   I2C 通道分配为 IRQ SLOT0，最高优先级      */
      VICVectAddr1   =   (uint32)IRQ_I2C;        /*   设置 I2C 中断向量                         */
      VICVectCntl0   =   (0x20|5);               /*   Timer1 通道分配为 IRQ SLOT1               */
      VICVectAddr0   =   (uint32)Timer1_CapInit; /*   设置 Timer1 中断向量                      */
      VICIntEnable   =   (1<<5|1<<9);            /*   使能 Timer1 中断、I2C 中断   */

      I2C_WriteNByte(0, 0x4E,ONE_BYTE_SUBA, 0x1B,udata_IOC3,1);      /*   设置 I03 口为输出      */
      I2C_WriteNByte(0, 0x4E,ONE_BYTE_SUBA, 0x0B,uLED_Close,1);      /*   熄灭所有 LED 灯        */
```

```
    I2C_WriteNByte(0, 0x4E,ONE_BYTE_SUBA,0x1C,udata_IOC4,1);        /*  设置 I04 口为输入  */ (1)
    /*  允许 I04 口的输入发生改变时产生中断  */
    I2C_WriteNByte(0, 0x4E,ONE_BYTE_SUBA, 0x24,udata_MSK4,1);                              (2)
    while(1) {
        if(flag) {                                                          /* 判断按键按下与否  */
            I2C_ReadNByte(0, 0x4E,ONE_BYTE_SUBA, 0x04,uRead_IP4,1);     /* 读取键值     */
            switch(uRead_IP4[0]) {
                case  0xBF:
                    /*  设置输入 I/O 口 4 极性发生反转  */
                    I2C_WriteNByte(0, 0x4E,ONE_BYTE_SUBA, 0x14, udata _ PI1,1);          (3)
                    /*  读取反转后寄存器的值  */
                    I2C_ReadNByte(0, 0x4E,ONE_BYTE_SUBA, 0x04, uoutdata _ PI1,1);
                    /*  读取的值去控制输出端口  */
                    I2C_WriteNByte(0, 0x4E,ONE_BYTE_SUBA, 0x0B, uoutdata _ PI1,1);
                    break;
            default:
                    break;
            }
        }
    }
    return 0;
}
```

程序说明：
在此程序执行后，当按键 7 按下时，LED 灯是低电平点亮那么只有第 7 盏 LED 会亮。但经过极性寄存器的反转后，LED 灯只有第 7 盏不亮。

### 4.3.3 对控制寄存器的读写应用

**（1）对“ALLBNK”寄存器的读写应用**

PCA9698 的“ALLBNK“寄存器，可以控制某一个组或所有组的 I/O 口全部输出 1 或 0。这样可以不必通过输出寄存器，就可以控制输出端口输出相同的电平状态。在 PACK 板上需用短路帽，将 I/O3 口与 LED 灯连接。

对“ALLBNK“寄存器的写操作如程序清单 4.3 所示，指令⑴的设置详见 2.8.1 小节的内容。

程序清单 4.3 对“ALLBNK“寄存器的写应用

```
/*********************************************************************************************
**  函数名称：ALLBNKRegWrite ()
**  函数功能：通过控制 I/O3 口的输出寄存器，点亮所有 LED 灯
*********************************************************************************************/
void ALLBNKRegWrite ()
{
    uint8 udata_IOC3[]  =  {0x00};                          /*  低电平为输出    */
    uint8 udata_LEDC[]  =  {0xFF};                          /*  高电平熄灭灯    */
    uint8 udata_ALLBNK[]  =  {0x00};                        /*  低电平点亮      */
```

```
    I2C_WriteNByte(0,0x4E,ONE_BYTE_SUBA,0x1B,udata_IOC3,1);        /* 设置 I03 口为输出   */
    I2C_WriteNByte(0,0x4E,ONE_BYTE_SUBA,0x0B,udata_LEDC,1);        /* 熄灭所有 LED 灯     */
    I2C_WriteNByte(0,0x4E,ONE_BYTE_SUBA,0x29, udata_ALLBNK,1);  /* 点亮所有灯          */   (1)
    while(1);
     return 0;
}
```

**（2） 对模式寄存器的写应用**

设置两个器件选择响应“GPIO ALL CALL”的模式，只要用一条指令即可控制这两个器件同时动作。在两个 PACK 板上需用短路帽，将 I/O0 口与 LED 灯连接。利用一条指令，实现两个板子上的 LED 灯执行相同的动作。

对模式寄存器的写操作如程序清单 4.4 所示，指令⑴、⑵的设置详见 2.8.1 小节的内容，指令⑶中的 0xDC 地址为 PCA9698 的“GPIO ALL CALL“地址，通过向此地址写入内容就可控制响应”GPIO ALL CALL“命令的 PACK 板详见 2.7 节内容。

程序清单 4.4 写模式寄存器的应用

```
/*****************************************************************************************
** 函数名称：ModeRegWrite ()
** 函数功能：通过控制模式选择寄存器，设置两个 PCA9698PACK 板选择响应“GPIO ALL CALL“命令
**           的模式
*****************************************************************************************/
void ModeRegWrite (void)
{
    uint8 udata_IOC0[]  =  {0x00};          /*  设置 I/O 口为输出的配置寄存器 0 的数据   */
    uint8 udata_MODE[]  =  {0x0A};          /*  设置响应 GPIO ALL CALL 的数据            */
    uint8 udata_GPIO[]  =  {0xAA};          /*  设置响应 GPIO ALL CALL 的数据            */
    uint8 uLED_close[]  =  {0xFF};          /*  设置熄灭所有 LED 灯的数据                */
    /*  设置响应 GPIO ALL CALL    */
    I2C_WriteNByte(0, 0x4E,ONE_BYTE_SUBA, 0x2A,udata_MODE,1);                              (1)
    I2C_WriteNByte(0, 0x40,ONE_BYTE_SUBA, 0x2A,udata_MODE,1);                              (2)
    I2C_WriteNByte(0, 0xDC,ONE_BYTE_SUBA,0x18,udata_IOC0,1);     /*  设置 I00 口为输出  */
    I2C_WriteNByte(0, 0x4E,ONE_BYTE_SUBA, 0x08,uLED_close,1);    /*  熄灭所有灯         */
    I2C_WriteNByte(0, 0x40,ONE_BYTE_SUBA, 0x08,uLED_close,1);    /*  熄灭所有灯         */
    I2C_WriteNByte(0, 0xDC ONE_BYTE_SUBA, 0x08,udata_GPIO,1);    /*  隔一个点亮一个  */  (3)
     while(1);
     return 0;
}
```

节省了代码，
提高了效率

程序说明：
利用 PCA9698 芯片中独有的 GPIO ALL CALL 命令，实现同时控制两个从器件的 LED 灯隔盏亮的功能。

## 4.4 总结

本文对 PCA9698 的介绍是对寄存器进行读写操作的基本应用，PCA9698 可广泛应用于服务器、手机、游戏系统、工业控制、医疗设备和仪器仪表等之中。

# 5. 版权声明

广州致远电子有限公司随附提供的软件或文档资料旨在提供给您(本公司的客户)使用，仅限于且只能在本公司制造或销售的产品上使用。

该软件或文档资料为本公司和/或其供应商所有，并受适用的版权法保护，版权所有。如有违反，将面临相关适用法律的刑事制裁，并承担违背此许可的条款和条件的民事责任。

本公司保留在不通知读者的情况下，修改文档或软件相关内容的权利，对于使用中所出现的任何效果，本公司不承担任何责任。

该软件或文档资料“按现状”提供。不提供保证，无论是明示的、暗示的还是法定的保证。这些保证包括（但不限于）对出于某一特定目的应用此软件的适销性和适用性默示的保证。在任何情况下，公司不会对任何原因造成的特别的、偶然的或间接的损害负责。

公　　司：广州致远电子有限公司 嵌入式系统事业部
地　　址：广州市天河区车陂路黄洲工业区二栋四楼（研发部）
邮　　编：510660
网　　址：www.embedtools.com
销售电话：+86 (020) 2264-4249
技术支持：+86 (020) 2887-2347
传　　真：+86 (020) 3860-1859
E–mail：lpc2103@ zlgmcu.com（技术支持）