项目四　LPC2000 基本输入输出控制

项目三学会了搭建嵌入式最小系统，而如何让 LPC2000 芯片通过 GPIO 接口完成一个简单的智能控制，需要了解芯片上 GPIO 功能的引脚，通过引脚如何输出高、低电平，以及 GPIO 接口的基本操作。本项目分 4 个任务详细介绍这部分内容。

任务一　选择 LPC2000 引脚功能为 GPIO

任务描述

掌握 LPC2000 的基本输入输出控制方式，并能够正确设置 LPC2000 的引脚功能为 GPIO 功能，结合 MagicARM2200 教学实验开发平台进行单灯闪烁的控制实验来有效地说明如何选择引脚为 GPIO 功能。

相关知识点

要完成这个任务，需要了解芯片上的引脚，以及如何选择引脚功能为 GPIO。

一、LPC2000 系列 GPIO 概述

1. 什么是 GPIO

GPIO 的英文全称 general-purpose input/output ports，中文意思是通用 I/O 端口。

LPC2000 系列 GPIO 具备以下特性。

(1)每个 I/O 口线可单独设置为输入/输出模式。LPC2000 系列的绝大多数 GPIO 为真正的全双向 I/O 口，可以独立控制每一根 I/O 口线的状态是输入还是输出。

(2)单独控制 I/O 口输出的置位或清零。绝大多数 GPIO 的输出为推挽输出，可以独立控制每一根 I/O 口的输出状态。

(3)所有 I/O 口在复位后默认为输入。虽然 LPC2000 系列的 I/O 电压为 3.3 V，GPIO 的输出最高为 I/O 口电源电压，但绝大多数 GPIO 能够承受 5 V 电压的输入，绝大多数

GPIO 作为输入时是处于高阻状态。

2. GPIO 的应用

GPIO 具有非常广泛的应用，主要应用在以下场合。

(1)通用 I/O 口。

(2)驱动 LED 或其他指示器。

(3)控制片外器件。

(4)检测数字输入，如键盘或开关信号。

因为 LPC2000 系列的 GPIO 具有以上特性，所以可以通过程序达到控制相应器件的目的。

LPC2000 系列 GPIO 作为输出，最常见的应用有控制 LED。

LED 在嵌入式系统常用作信号灯，指示系统当前的某些状态。LED 的控制很简单，只需在阳极与阴极间提供一个 1.7 V 的正向电压，并使流经 LED 的电流为 5～10 mA，即可较理想地点亮 LED。如图 4-1 所示，设置 GPIO 引脚为输出方式，使 GPIO 引脚 P0.10 输出低电平时，VDD3.3 与 GPIO 引脚即有 3.3 V 的电压差，这时 LED1 即被点亮；使 GPIO 引脚 P0.10 输出高电平时，VDD3.3 与 GPIO 引脚电压差为 0，这时 LED1 不能被点亮；电阻 R1～R4 用于限流。如果使用 GPIO 控制较多的 LED，需要使用三极管驱动。

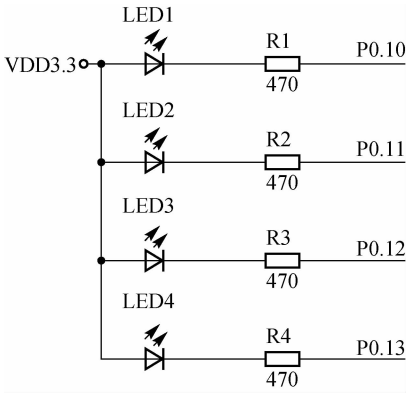

图 4-1　GPIO 直接驱动 LED

GPIO 引脚描述见表 4-1，这是 P0 和 P1 端口的 GPIO 引脚。除 P0 和 P1 外，LPC2210/2212/2214 还包含另外两个端口——P2 和 P3，这两个端口与外部存储器总线复用，只有在不用作外部存储器总线时(通过配置 PINSEL2 寄存器实现)，才可以作为 GPIO 使用，如配置为 16 位总线接口时，D16～D31 可作为 GPIO。

表 4-1　GPIO 引脚描述

引脚名称	类　型	描　述
P0.0～P0.31 P1.16～P1.31	输入/输出	通用 I/O 口，实际可用的 GPIO 数量取决于可选功能的使用，即引脚连接模块的设置

二、设置引脚选择寄存器(PINSELx)

LPC2000 的基本输入输出功能是通过软件配置寄存器来实现的,GPIO 相关寄存器结构如图 4-2 所示。

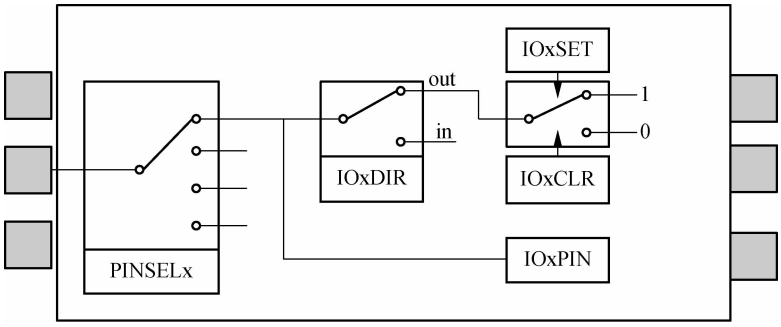

图 4-2　GPIO 相关寄存器结构

其中,引脚连接模块使一个引脚可以具有多种功能,即引脚复用,这是通过配置相关寄存器控制多路开关来连接引脚与片内外设,如图 4-3 所示。

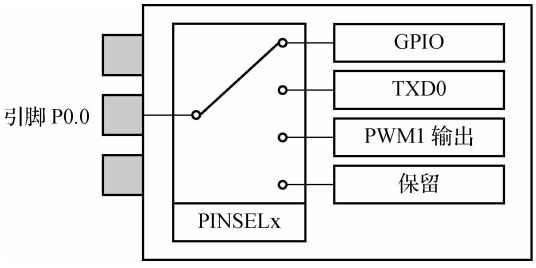

图 4-3　引脚连接模块示意图

引脚连接模块具有 3 个相关寄存器,分别是 PINSEL0、PINSEL1 和 PINSEL2,见表 4-2。

表 4-2　引脚连接模块寄存器映射

名　称	描　述	访　问
PINSEL0	引脚功能选择寄存器 0	读/写
PINSEL1	引脚功能选择寄存器 1	读/写
PINSEL2	引脚功能选择寄存器 2	读/写

1. 引脚功能选择寄存器 0(PINSEL0)

PINSEL0 寄存器按照表 4-3 中的设定来控制引脚的功能。IODIR 寄存器中的方向控制位只有在引脚选择 GPIO 功能时才有效。对于其他功能,方向是自动控制的。

表 4-3　引脚功能选择寄存器 0

PINSEL0	引脚名称	00	01	10	11
1:0	P0.0	GPIO P0.0	TXD(UART0)	PWM1	保留
3:2	P0.1	GPIO P0.1	RXD(UART0)	PWM3	EINT0
5:4	P0.2	GPIO P0.2	SCL(I^2C)	捕获 0.0(TIMER0)	保留
7:6	P0.3	GPIO P0.3	SDA(I^2C)	匹配 0.0(TIMER0)	EINT1
9:8	P0.4	GPIO P0.4	SCK(SPI0)	捕获 0.1(TIMER0)	保留
11:10	P0.5	GPIO P0.5	MISO(SPI0)	匹配 0.1(TIMER0)	保留
13:12	P0.6	GPIO P0.6	MOSI(SPI0)	捕获 0.2(TIMER0)	保留
15:14	P0.7	GPIO P0.7	SSEL(SPI0)	PWM2	EINT2
17:16	P0.8	GPIO P0.8	TXD(UART1)	PWM4	保留
19:18	P0.9	GPIO P0.9	RXD(UART1)	PWM6	EINT3
21:20	P0.10	GPIO P0.10	RTS(UART1)	捕获 1.0(TIMER1)	保留
23:22	P0.11	GPIO P0.11	CTS(UART1)	匹配 1.0(TIMER1)	保留
25:24	P0.12	GPIO P0.12	DSR(UART1)	捕获 1.1(TIMER1)	保留
27:26	P0.13	GPIO P0.13	DTR(UART1)	匹配 1.1(TIMER1)	保留
29:28	P0.14	GPIO P0.14	CD(UART1)	EINT1	保留
31:30	P0.15	GPIO P0.15	RI(UART1)	EINT2	保留

2. 引脚功能选择寄存器 1(PINSEL1,0xE002C004)

PINSEL1 寄存器按照表 4-4 中的设定来控制引脚的功能。IODIR 寄存器中的方向控制位只有在引脚选择 GPIO 功能时才有效。对于其他功能,方向是自动控制的。

表 4-4　引脚功能选择寄存器 1

PINSEL1	引脚名称	00	01	10	11
1:0	P0.16	GPIO P0.16	EINT0	保留	保留
3:2	P0.17	GPIO P0.17	捕获 1.2(TIMER1)	SCK(SPI1)	匹配 1.2(TIMER1)
5:4	P0.18	GPIO P0.18	捕获 1.3(TIMER1)	MISO(SPI1)	匹配 1.3(TIMER1)
7:6	P0.19	GPIO P0.19	匹配 1.2(TIMER1)	MOSI(SPI1)	匹配 1.3(TIMER1)
9:8	P0.20	GPIO P0.20	匹配 1.3(TIMER1)	SSEL(SPI1)	EINT3
11:10	P0.21	GPIO P0.21	PWM5	保留	捕获 1.3(TIMER1)
13:12	P0.22	GPIO P0.22	保留	捕获 0.0(TIMER0)	匹配 0.0(TIMER0)
15:14	P0.23	GPIO P0.23	保留	保留	保留
17:16	P0.24	GPIO P0.24	保留	保留	保留
19:18	P0.25	GPIO P0.25	保留	保留	保留
21:20	P0.26	保留			
23:22	P0.27	GPIO P0.27	AIN0(A/D 转换器)	捕获 0.1(TIMER0)	匹配 0.1(TIMER0)
25:24	P0.28	GPIO P0.28	AIN1(A/D 转换器)	捕获 0.2(TIMER0)	匹配 0.2(TIMER0)
27:26	P0.29	GPIO P0.29	AIN2(A/D 转换器)	捕获 0.3(TIMER0)	匹配 0.3(TIMER0)
29:28	P0.30	GPIO P0.30	AIN3(A/D 转换器)	EINT3	捕获 0.0(TIMER0)
31:30	P0.31	保留			

3. 引脚功能选择寄存器 2(PINSEL2,0xE002C014)

PINSEL2 寄存器按照表 4-5、表 4-6 中的设定来控制引脚的功能。IODIR 寄存器中的方向控制位只有在引脚选择 GPIO 功能时才有效。对于其他功能,方向是自动控制的。

表 4-6 中的"复位值"这一栏表示微控制器复位时对应位的值;对于 PINSEL2 的 bit2、bit3,复位时由 P1.26、P1.20 引脚的电平决定,若引脚接有上拉电阻(如 10 kΩ 上拉电阻),相应位的值被设置为 0;若引脚接有下拉电阻(如 4.7 kΩ 下拉电阻),相应位的值被设置为 1。对于 PINSEL2 的 bit23、bit24、bit25～bit27,复位时由 BOOT1 和 BOOT0 引脚的电平决定。

警告:使用读—修改—写的方法来访问 PINSEL2 寄存器。

例:PINSEL2 =(PINSEL2&0xFFFFFFCF) | (2<<4)。

对 bit0～bit2 和(或)bit3 的意外写操作会造成调试和(或)跟踪功能的丢失。

表 4-5 LPC2114/2124 引脚功能选择寄存器 2

PINSEL2	描　述
1:0	保留
2	该位为 0 时,P1.31:26 用作 GPIO; 该位为 1 时,P1.31:26 用作调试端口
3	该位为 0 时,P1.25:16 用作 GPIO; 该位为 1 时,P1.25:16 用作跟踪端口
31:4	保留

表 4-6 LPC2210/2212/2214 引脚功能选择寄存器 2

PINSEL2	描　述			复位值
1:0	保留			00
2	该位为 0 时,P1.31:26 用作 GPIO;该位为 1 时,P1.31:26 用作一个调试端口			$\overline{\text{P1.26/RTCK}}$
3	该位为 0 时,P1.26:16 用作 GPIO;该位为 1 时,P1.26:16 用作一个调试端口			$\overline{\text{P1.20/TRACESYNC}}$
5:4	控制数据总线和选通引脚的使用: 引脚 P2.7:0	11=GPIO	0x 或 10=D7:0	BOOT1:0 (如 BOOT1:0=01, 该域的复位值就为 01)
	引脚 P1.0	11=GPIO	0x 或 10=CS0	
	引脚 P1.1	11=GPIO	0x 或 10=OE	
	引脚 P3.31	11=GPIO	0x 或 10=BLS0	
	引脚 P2.15:8	00 或 11=GPIO	01 或 10=D15:8	
	引脚 P3.30	00 或 11=GPIO	01 或 10=BLS1	
	引脚 P2.27:16	0x 或 11=GPIO	10=D27:16	
	引脚 P2.29:28	0x 或 11=GPIO	10=D29:28	
	引脚 P2.31:30	0x 或 11=GPIO 或 AIN6:7	10=D31:30	
	引脚 P3.29:28	0x 或 11=GPIO 或 AIN6:7	10=BLS2:3	

续表

PINSEL2	描　　述	复 位 值
6	如果位 5:4 不为 10,由该位控制 P3.29 脚的使用:为 0 时使能 P3.29,为 1 时使能 AIN6	1
7	如果位 5:4 不为 10,由该位控制 P3.28 脚的使用:为 0 时使能 P3.28,为 1 时使能 AIN7	1
8	该位控制 P3.27 脚的使用:为 0 时使能 P3.27,为 1 时使能 WE	0
10:9	保留	
11	该位控制 P3.26 脚的使用:为 0 时使能 P3.26,为 1 时使能 CS1	0
12	保留	
13	如果位 25:23 不为 111,由该位控制 P3.27/A23/XCLK 脚的使用:为 0 时使能 P3.23,为 1 时使能 XCLK	0
15:14	控制 P3.25 脚的使用:00 使能 P3.25,01 使能 CS2,10 和 11 保留	00
17:16	控制 P3.24 脚的使用:00 使能 P3.24,01 使能 CS2,10 和 11 保留	00
19:18	保留	
20	如果位 5:4 不为 10,由该位控制 P2.29:28 的使用:0 使能 P2.29:28,1 保留	0
21	如果位 5:4 不为 10,由该位控制 P2.30 的使用:0 使能 P2.30,1 使能 AIN4	1
22	如果位 5:4 不为 10,由该位控制 P2.31 的使用:0 使能 P2.31,1 使能 AIN4	1
23	控制 P3.0/A0 用作端口引脚(0)或地址线(1)	如果$\overline{\text{RESET}}$=0 时 BOOT1:0=00,该位的复位值为 1,反之为 0
24	控制 P3.1/A1 用作端口引脚(0)或地址线(1)	如果复位时 BOOT1=0,该位的复位值为 1,反之为 0
27:25	控制 P3.23/A23/XCLK 和 P3.22:2/A2.22:2 中地址线的选择: 000=无地址线　　100=A11:2 为地址线 001=A3:2 为地址线　　101=A15:2 为地址线 010=A5:2 为地址线　　110=A19:2 为地址线 011=A7:2 为地址线　　111=A23:2 为地址线	如果复位时 BOOT:0=11,该位的复位值为 000,反之为 111
31:28	保留	

例 4.1　将 P0.8、P0.9 设置为 TxD1、RxD1 功能。

```
PINSEL0 = 0x00050000;
```

或

```
PINSEL0 = 0x05<<16;
```

PINSEL0、PINSEL1 和 PINSEL2 寄存器是可读可写的,为了不更改原来的引脚功能设置,可以先读取寄存器值,然后进行逻辑“与”“或”操作,再回写到此寄存器。

```
PINSEL0 = ( PINSEL0 & 0xFFF0FFFF ) | (0x05<<16);
```

三、嵌入式 C 语言程序的基本结构

一个被开发的软件往往由许多功能组成，包含的程序语句很多。从组成来看，各个功能模块之间彼此有一定的联系，但是在功能上是相对独立的。如何将不同的功能模块连接在一起形成一个程序，这就需要进行模块化程序设计。模块化程序设计是指将一个大的程序自上向下进行功能分解，将其分成若干个子模块，每个模块对应了一个功能，完成独立的功能。而支持这种设计方法的语言被称为模块化程序设计语言，C 语言就具备这种模块化程序设计的功能，实现这种功能的工具就是函数，函数是一个可以反复使用的、具有特定功能的且相对独立的程序段。使用函数可以使程序变得更简短而清晰，有利于程序的修改和维护，可以提高程序开发的效率，提高代码的重用性。

因此，C 语言是一种函数式语言，它的一个函数实际上就是一个功能模块，C 程序的基本组成就是函数。换句话说，C 程序是由一个主函数 main 和其他若干函数构成的，其结构如下。

```
#include  "config.h"              //包含头文件
#define   LEDCON  0x00000040      //宏定义
void DelayNS(uint32 dly);         //函数声明
/*主函数*/
int  main(void)
{
    PINSEL0 = 0x00000000;         //设置引脚连接 GPIO
    IO0DIR = LEDCON;              //设置 I/O 为输出
    while(1)
    {
        IO0SET = LEDCON;
        DelayNS(15);
        IO0CLR = LEDCON;
        DelayNS(15);
    }
    return(0);
}
/*延时函数*/
void  DelayNS(uint32  dly)
{  uint32  i;
    for(; dly>0; dly--)
    {
        for(i=0; i<5000; i++);
    }
}
```

嵌入式 C 语言开发环境不同于标准 C 语言开发环境，标准 C 语言的程序设计使用的是

Turbo C，而嵌入式 C 语言程序设计不能使用 Turbo C，必须使用专门的开发环境和开发工具，如 ADS、Keil uV3、Keil uV4、IAR、LPCXpresso IDE 等开发环境，以及 EasyJTAG-H、JLink 等编程器。

任务解析

控制 P2.16 连接的 LED1 处于闪烁状态。P2.16 引脚作为输出，结合以上相关知识点，应该将 P2.16 引脚设置为 GPIO 功能，使用 PINSEL2 寄存器来设置引脚的功能，通过查表 4-6，输入下列语句即可实现。

```
PINSEL2 = 0x00000000;
```

单 LED 闪烁程序流程图如图 4-4 所示。

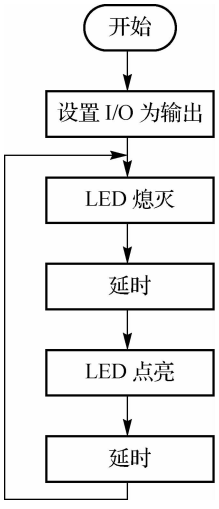

图 4-4　单 LED 闪烁程序流程图

```
/*************************C 语言源程序代码************************/
#include  "config.h"
#define  LED1   0x00010000                    //P2.16 引脚控制 LED1
/*****************************************************************
**名      称:DelayNS
**功      能:长软件延时
**入口参数:dly     延时参数,值越大,延时越久
**返 回 值:无
*****************************************************************/
void  DelayNS(uint32  dly)
{   uint32  i;
     for(; dly>0; dly--)
     {
for(i=0; i<5000; i++);
```

```
    }
}
/*************************************************************
* * 名    称:main
* * 功    能:对 LED1 进行控制,采用软件延时方法
* * 入口参数:无
* * 返 回 值:0
*************************************************************/
int  main(void)
{
/* PINSEL2 使用启动代码的默认配置,切勿任意配置 PINSEL2,否则总线会受到干扰,
因此这里不用设置 PINSEL2 */
IO2DIR = LED1;                          //设置 P2.16 引脚为输出
       while(1)
       {
            IO2CLR = LED1;              //LED1 亮
            DelayNS(15);
            IO2SET = LED1;              //LED1 灭
            DelayNS(15);
       }
    return(0);
}
```

任务实施

具体操作步骤如下(相关图示可参考本书附录 B)。

(1)连接 EasyJTAG 仿真器和 MagicARM2200 教学实验开发平台。将 EasyJTAG 仿真器的 25 针接口通过并口延长线与 PC 的并口连接,将 EasyJTAG 仿真器的 20 针接口通过 20 PIN 连接电缆接到 MagicARM2200 教学实验开发平台的 J3 上,将实验箱上的跳线 JP23 连接,JP15 断开,最后打开实验箱电源。

(2)安装 EasyJTAG 仿真器的驱动软件 H-JTAG(若已经安装过,此步省略)。

(3)打开 H-JTAG 软件,配置 H-JTAG。

①单击任务栏的 H 提示图标,将看到 H-JTAG 的主窗口。单击放大镜按钮后应能看到 ARM7 处理器。

②执行 Flasher→Auto Download 菜单命令,选择自动下载项。

③启动 H-Flasher,并加载配置文件。单击任务栏的 F 提示图标或执行 Flasher→Start H-Flasher 菜单命令,将会弹出 H-Flasher 的主窗口。单击 Load 菜单装载目标板配置文件 LPC2200.hfc(文件路径为 C:\Program Files\ H-JTAG V0.3.2\ LPC2200.hfc),配置 H-Flasher 如图 4-5 所示。

图 4-5　装载配置文件

④验证并启用配置。验证调试代理配置是否正确，打开 H-Flasher 对话框的 Programming 选项单击 Check 按钮，如果正常，可看到所使用的 Flash 芯片的型号。单击 Check 按钮时，H-Flasher 就会启用当前新的配置值。到此配置完成。

(4)启动 ADS 1.2，使用 ARM Executable Image for MagicARM2200 工程模板建立一个工程 GPIO_C.mcp(也可以直接复制一个完整的工程文件将其作为模板进行修改)。

(5)在 user 组的 main.c 中编写实验程序代码(详见后面的实验参考程序)。

(6)选用 DebugInExram 生成目标，如图 4-6 所示，然后编译连接工程。

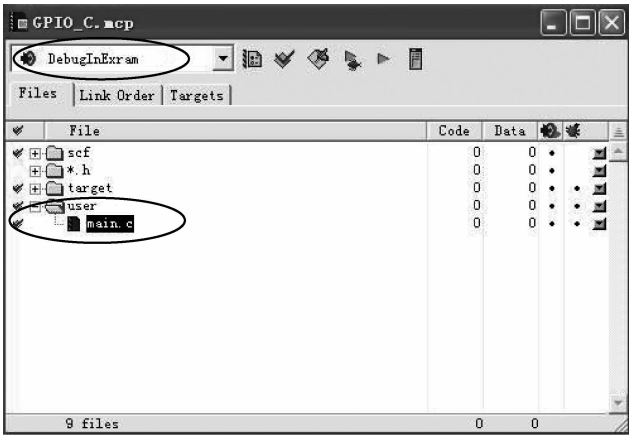

图 4-6　选择生成目标

(7)将 MagicARM2200 教学实验开发平台上的跳线器 JP23 短接，JP15 断开。

(8)执行 Project→Debug 菜单命令，启动 AXD 进行 JTAG 仿真调试(需要正确设置仿真器，如图 4-7 和图 4-8 所示，也可以参考附录中 ADS 集成开发环境及仿真器应用中的内容)。

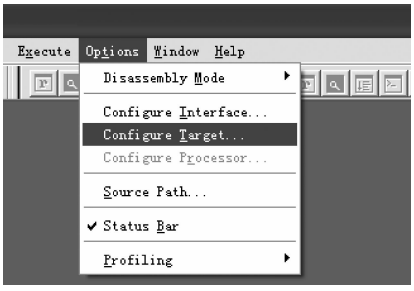

图 4-7　Configure Target

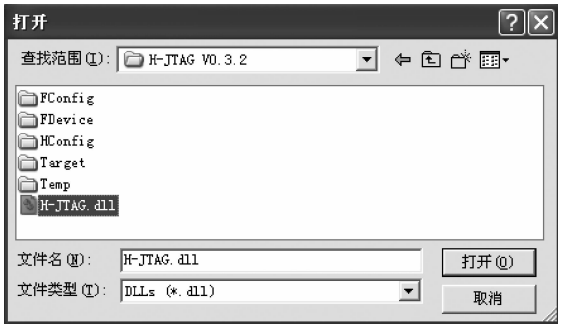

图 4-8　打开 H-JTAG.dll 文件

注意:使用 DebugInExram 生成目标时,使用片外 RAM 进行仿真调试,建议 AXD 设置 Halt Mode 选择 Halt program,Aux Option 选择 Erase Flash when need。片外 RAM 调试的仿真器设置参考如图 4-9 所示。

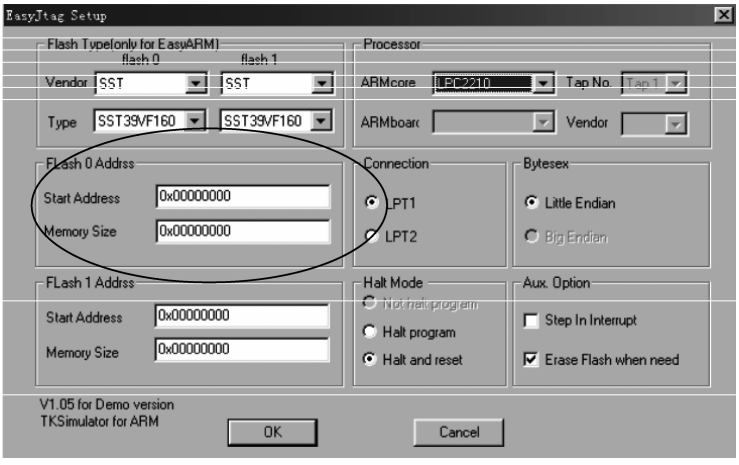

图 4-9　片外 RAM 调试的仿真器设置参考

(9)若 JTAG 连接出错,或 AXD 主窗口没有显示源程序 main.c,按本书 ADS 集成开发环境及仿真器应用中介绍的方法进行处理。

(10)全速运行程序,程序会在 main.c 的主函数中停止(因为调试选项设置第一次启动运行停在 main 处)。

(11)可以单步运行程序,也可以设置/取消断点,或者全速运行程序。停止程序运行,观察变量的值,判断 LED 控制是否正确。

任务二　GPIO 输出控制蜂鸣器蜂鸣

任务描述

利用 LPC2000 的 GPIO 输出功能控制 MagicARM2200 教学实验开发平台上 P0.7 引脚

连接的蜂鸣器蜂鸣，从而实现报警控制。

相关知识点

在本章涉及的知识点中，需要掌握 GPIO 输出功能的设置和蜂鸣器的工作原理。

一、蜂鸣器和驱动电路

1. 蜂鸣器

LPC2000 系列芯片驱动的输出设备除了 LED，常见的还有蜂鸣器。

在嵌入式系统中常用的蜂鸣器有直流型和交流型两种。

直流型蜂鸣器只需提供额定电压就可以控制蜂鸣器蜂鸣；交流型蜂鸣器则需提供一定频率的交流信号，才可以使蜂鸣器鸣响。

直流型蜂鸣器的蜂鸣频率是固定不能更改的，而交流型蜂鸣器则可以通过更改驱动电流的频率来调整蜂鸣频率。

2. 蜂鸣器驱动电路

以上两种类型的蜂鸣器都可以使用相同的控制电路，只是控制方式有所不同。GPIO 提供的输出电流不能直接驱动蜂鸣器，需经过三极管驱动，如图 4-10 所示。三极管 8550 是一种常用的普通三极管，它是一种低电压、大电流、小信号、PNP 型、硅材料的三极管。8550 三极管采用 TO-92 封装。

8550 三极管贴片采用 SOT-23 封装，如图 4-11 所示。

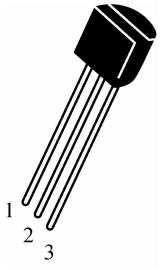

图 4-10　8550 TO-92 封装引脚图

1—发射极；2—基极；3—集电极

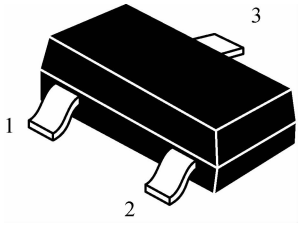

图 4-11　8550 SOT-23 封装引脚图

1—基极；2—发射极；3—集电极

8550 三极管的主要参数如下。

(1)最大集电极电流：0.5 A。

(2)直流电增益：10～60。

(3)集电极-发射极击穿电压：25 V。

8550 三极管主要用于开关和射频放大。

蜂鸣器三极管驱动电路如图 4-12 所示。当 P0.7 为低电平，三极管基极为低电平，三极管饱和导通，集电极为高电平，即蜂鸣器被控制端为高电平，另一端接地，所以蜂鸣器响。反之，当 P0.7 为高电平，三极管基极为高电平，三极管截止，蜂鸣器不响。

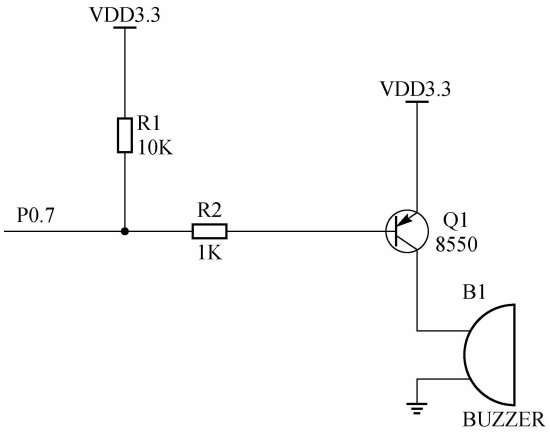

图 4-12　蜂鸣器三极管驱动电路

二、GPIO 方向寄存器(IOxDIR,x=0,1,2,3)

LPC2114/2124 有 2 个 32 位的通用 I/O 口。P0 口使用了 30 个引脚(P0.0～P0.25,P0.27～P0.30)作为 GPIO 功能,P1 口有 16 个引脚可用作 GPIO 功能(P1.16～P1.31)。P0 口和 P1 口都分别有 4 个寄存器控制,见表 4-7。对于 LPC2210/2212/2214,P2 口和 P3 口各寄存器的功能与 P0 和 P1 的寄存器是一致的。P2 口的寄存器起始地址为 0xE0028020,P3 口的寄存器起始地址为 0xE0028030。不同 GPIO 接口的 4 个寄存器命名就是在 IO 后面加 0、1、2、3(如 P3 口的引脚值寄存器名称为 IO3PIN)来区分不同的 IO 口。P0、P1、P2、P3 接口的功能描述和设置方法相同。

表 4-7　GPIO 寄存器映射——P0 和 P1

P0 口寄存器名称	P1 口寄存器名称	描　　述	访　　问
IO0PIN	IO1PIN	GPIO 引脚值寄存器	只读
IO0SET	IO1SET	GPIO 输出置位寄存器。写入 1 使对应引脚输出高电平,输入 0 无效	读/置位
IO0DIR	IO1DIR	GPIO 方向控制寄存器。该寄存器单独控制每个 I/O 口的方向	读/写
IO0CLR	IO1CLR	GPIO 输出清零寄存器。写入 1 使对应引脚输出低电平,写入 0 无效	只清零

LPC2000 的 GPIO 功能是通过软件配置寄存器来实现的,GPIO 相关寄存器结构如图 4-2 所示。

当引脚配置为 GPIO 模式时,可使用 GPIO 方向寄存器控制引脚的方向。任意引脚的方向位的设置必须与引脚功能一致,如某引脚用作输出功能,IODIR 寄存器的相应位必须设置为 1。IODIR 寄存器描述见表 4-8。

表 4-8　GPIO 方向寄存器描述

IODIR	描　　述
31:0	方向控制位。 0:输入;1:输出。 IO0DIR 的位 0 控制 P0.0,……,位 31 控制 P0.31。 IO1DIR 的位 0 控制 P1.0,……,位 31 控制 P1.31。 IO2DIR 的位 0 控制 P2.0,……,位 31 控制 P2.31。 IO3DIR 的位 0 控制 P3.0,……,位 31 控制 P3.31

例 4.2　设置 P0.0 口为输出模式。

```
/*程序清单 设置 P0.0 为输出模式*/
PINSEL0 = 0x00000000;
IO0DIR = 0x00000001;
```

或

```
PINSEL0 &= 0xFFFFFFFC;
IO0DIR |= 0x01;
```

任务解析

1. 电路分析

MagicARM2200 实验箱采用三极管 8550 驱动蜂鸣器，如图 4-13 所示，P0.7 控制三极管 8550，当 P0.7 为低电平，三极管基极为低电平，三极管饱和导通，集电极为高电平，即蜂鸣器被控制端为高电平，另一端接地，所以蜂鸣器响。反之，当 P0.7 为低电平，三极管基极为低电平，三极管截止，蜂鸣器不响。

注意：将 JP22 短接，JP20 断开。

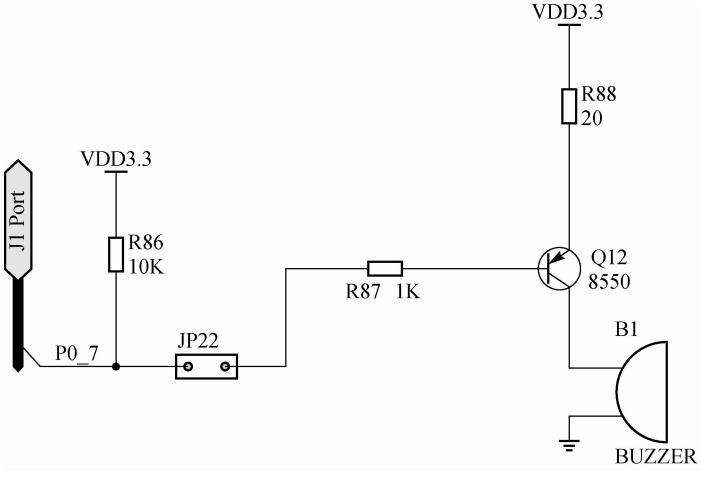

图 4-13　蜂鸣器接线原理图

2. 程序设计

本任务采用 LPC2290 芯片的 P0.7 引脚控制蜂鸣器。蜂鸣器控制程序流程图如图 4-14 所示。

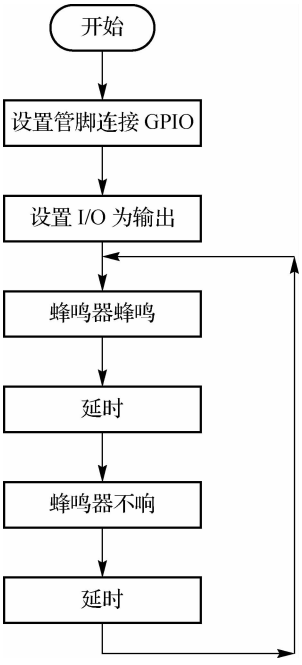

图 4-14　蜂鸣器控制程序流程图

```
/*************************C 语言源程序代码************************/
#include  "config.h"
#define BEEPCON   0x00000080              //P0.7 引脚控制 B1,低电平蜂鸣
/*****************************************************************
**名    称:DelayNS
**功    能:长软件延时
**入口参数:dly  延时参数,值越大,延时越久
**返 回 值:无
*****************************************************************/
void  DelayNS(uint32  dly)
{   uint32  i;
    for(; dly>0; dly--)
    {
        for(i=0; i<5000; i++);
    }
}
/*****************************************************************
**名    称:main
```

```
**  功      能:控制蜂鸣器蜂鸣
**  入口参数:无
**  返 回 值:0
*********************************************************************/
int  main(void)
{
    PINSEL0 = 0x00000000;                    //设置 P0.7 引脚连接 GPIO
    IO0DIR = BEEPCON;                        //设置 I/O 为输出
    while(1)
    {
      IO0CLR = BEEPCON;                      //蜂鸣器响
      DelayNS(15);
      IO0SET= BEEPCON;                       //蜂鸣器不响
      DelayNS(15);
    }
    return(0);
}
```

任务实施

GPIO 输出控制蜂鸣器蜂鸣实验的操作步骤如下。

(1)连接 EasyJTAG 仿真器和 MagicARM2200 教学实验开发平台。将 EasyJTAG 仿真器的 25 针接口通过并口延长线与 PC 的并口连接,将 EasyJTAG 仿真器的 20 针接口通过 20 PIN 连接电缆接到 MagicARM2200 教学实验开发平台的 J3 上,将 MagicARM2200 教学实验开发平台上的跳线器 JP22 短接,JP20 断开。最后打开实验箱电源。

(2)安装 EasyJTAG 仿真器的驱动程序(若已经安装过,此步省略)。

(3)打开 H-JTAG 软件,设置 H-JTAG 的配置(操作方法详见任务 4.1)。

(4)启动 ADS 1.2,使用 ARM Executable Image for MagicARM2200 工程模板建立一个工程 BeepCon_C(也可以直接复制一个完整的工程文件将其作为模板进行修改)。

(5)在 user 组的 main.c 中编写实验程序代码(详见前面的任务解析)。

(6)选用 DebugInExram 生成目标,然后编译链接工程。

(7)单击 Debug 按钮,启动 AXD 进行 JTAG 仿真调试(需要正确设置仿真器,参考附录 B)。若 JTAG 连接出错,或 AXD 主窗口没有显示 Startup.S 源程序,按本书 ADS 集成开发环境及仿真器应用中介绍的方法进行处理。

(8)全速运行程序,程序将会在 main.c 的主函数中停止(因为调试选项设置第一次启动运行停在 main 处)。

(9)全速运行程序,观察蜂鸣器的状态。

任务三　用按键控制蜂鸣器

任务描述

根据 LPC2000 的 GPIO 输入输出功能，利用 MagicARM2200 教学实验开发平台上的一个按键输入来控制蜂鸣器的输出状态。

相关知识点

一、GPIO 引脚值寄存器(IOxPIN，x＝0，1，2，3)

LPC2000 的 GPIO 功能是通过软件配置寄存器来实现的，GPIO 相关寄存器结构如图 4-2 所示，其中 IOxPIN 为引脚值寄存器。

引脚值寄存器提供 GPIO 引脚的值，它反映了外部环境对引脚的影响。寄存器的每一位的值和引脚值一一对应。IOPIN 寄存器描述见表 4-9。

表 4-9　IOPIN 寄存器描述

IOPIN	描　　述
31:0	GPIO 引脚值。 IO0PIN 的位 0 对应于 P0.0，……，位 31 对应于 P0.31。 IO1PIN 的位 0 对应于 P1.0，……，位 31 对应于 P1.31。 IO2PIN 的位 0 对应于 P2.0，……，位 31 对应于 P2.31。 IO3PIN 的位 0 对应于 P3.0，……，位 31 对应于 P3.31

例 4.3　读取 P0.0 引脚状态，如图 4-2 所示。

```
/*程序清单 读 P0.0 引脚状态*/
uint32  PinStat;
PINSEL0 = 0x00000000;
IO0DIR = 0x00000000;          //设置 P0.0 口方向，设置为输入
PinStat = IO0PIN;             //从 IO0PIN 读取引脚状态，存放到变量 PinStat
```

二、按键

按键是嵌入式系统最为常见的输入设备，绝大多数需要人机交互的嵌入式系统都离不开按键。基于 LPC2000 系列微控制器中，使用 GPIO 部件实现按键功能是最简单且低成本的方法。使用 GPIO 部件实现按键功能通常有两种方法：独立式按键输入和行列式键盘输入。

独立式按键输入编程简单，每个按键都占用一个 GPIO 引脚，如图 4-15 所示。使用时，定义 GPIO 为输入方式，由于每个 GPIO 引脚都接有上拉电阻，所以当没有键按下时，读取

GPIO 状态都为高电平，当有键被按下，读取 GPIO 状态时，被按下的 GPIO 引脚为低电平。通过判断 GPIO 引脚电平状态确定按键是否被按下。

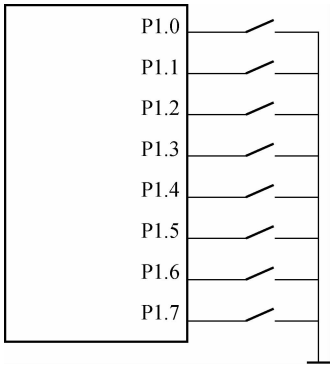

图 4-15　独立式按键输入

如果需要的按键数目较多，而 GPIO 引脚不够时，可以考虑使用行列式键盘输入方式。行列式键盘输入方式使用较少的 GPIO 引脚，可支持较多的按键，其缺点是编程较复杂。如图 4-16 所示，P1.0～P1.3 设置为 GPIO 输出引脚，P3.0～P3.3 设置为 GPIO 输入引脚。P1.0～P1.3 引脚以一定的顺序及频率在同一时间仅使其中一个引脚输出低电平。控制器快速查询 P3.0～P3.3 引脚的电平状态，如果有 1 个键被按下，那么在一定时间内 P3.0～P3.3 中将有一个脚为低电平，就可以很容易地判断出哪个按键被按下。

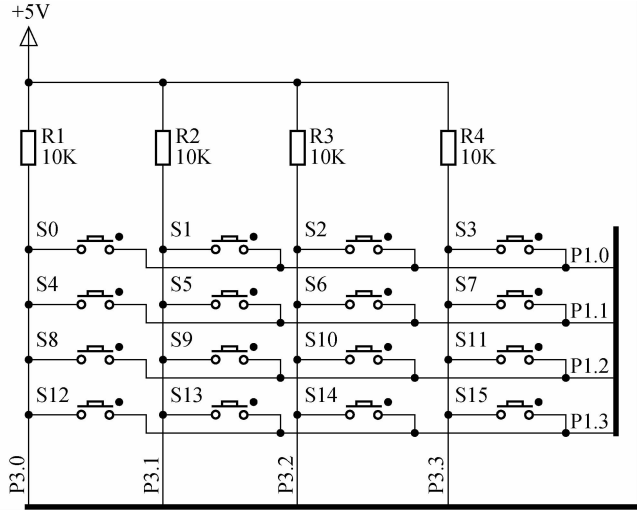

图 4-16　行列式键盘输入

任务解析

1. 电路分析

P0.20 口设置为输入模式时，口线内部无上拉电阻。当与按键或跳线器连接使用时需

要外接上拉电阻，防止口线悬空。GPIO 输入电路原理图如图 4-17 所示(将 JP9 短接)。

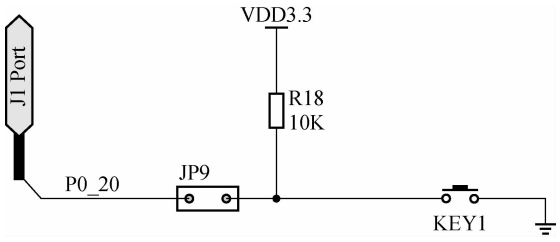

图 4-17　GPIO 输入电路原理图

MagicARM2200 实验箱采用三极管 8550 驱动蜂鸣器，电路如图 4-13 所示(将 JP22 短接，JP20 断开)。

2. 软件分析

查询按键所连接的 P0.20 是否为低电平，若为低电平，则有按键按下，控制 P0.7 输出低电平，蜂鸣器响；否则控制 P0.7 输出高电平，蜂鸣器不响。按键输入控制蜂鸣器程序流程图如图 4-18 所示。

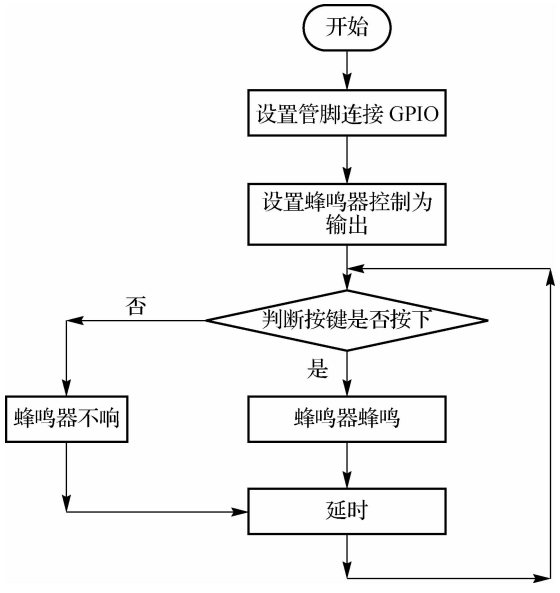

图 4-18　按键输入控制蜂鸣器程序流程图

```
/************************ C 语言程序清单 ************************/
#include  "config.h"
#define   BEEPCON  0x00000080          // P0.7 引脚控制 B1,低电平蜂鸣
#define   KEY      0x00100000          // 定义按键
/*************************************************************
* * 名     称:main
* * 功     能:读取 P0.20 口的值,并输出控制蜂鸣器 B1
* * 输入参数:无
```

```
**返 回 值:0
*************************************************************/
int  main(void)
{  uint32  i;
   PINSEL0 = 0x00000000;                    //设置管脚连接 GPIO
   IO0DIR = BEEPCON;                        //设置 B1 控制口为输出
   while(1)
   {
      if( (IO0PIN&KEY)! =0 )                //读按键值,若无键按下,则蜂鸣器不响
        {
          IO0SET = BEEPCON;
        }
     else                                   //读按键值,若有键按下,则蜂鸣器响
     {
       IO0CLR = BEEPCON;
     }
     for(i=0; i<1000; i++);
  }
return(0);
}
```

任务实施

用一个按键来控制蜂鸣器输出状态实验的操作步骤如下。

(1)连接 EasyJTAG 仿真器和 MagicARM2200 教学实验开发平台。将 EasyJTAG 仿真器的 25 针接口通过并口延长线与 PC 的并口连接,将 EasyJTAG 仿真器的 20 针接口通过 20 PIN 连接电缆接到 MagicARM2200 教学实验开发平台的 J3 上,将 MagicARM2200 教学实验开发平台上的跳线器 JP22 短接,JP20 断开。最后打开实验箱电源。

(2)安装 EasyJTAG 仿真器的驱动程序(若已经安装过,此步省略)。

(3)打开 H-JTAG 软件,设置 H-JTAG 的配置(操作方法详见任务 4.1)。

(4)启动 ADS 1.2,使用 ARM Executable Image for MagicARM2200 工程模板建立一个工程 ReadPin_C(也可以直接复制一个完整的工程文件将其作为模板进行修改)。

(5)在 user 组的 main.c 中编写实验程序代码(详见前面的任务解析)。

(6)选用 DebugInExram 生成目标,然后编译链接工程。

(7)单击 Debug 按钮,启动 AXD 进行 JTAG 仿真调试(需要正确设置仿真器,参考附录 B)。若 JTAG 连接出错,或 AXD 主窗口没有显示源程序,按本书 ADS 集成开发环境及仿真器应用中介绍的方法进行处理。

(8)全速运行程序,程序将会在 main.c 的主函数中停止(因为调试选项设置第一次启动运行停在 main 处)。

(9)全速运行程序,当有按键按下时,观察蜂鸣器的状态。

任务四　实现 8 个 LED 流水灯状态

任务描述

利用 LPC2000 的 GPIO 输出置位和清零功能控制 MagicARM2200 教学实验开发平台上的 8 个 LED 进行流水灯输出实验。

相关知识点

在本章涉及的知识点中,需要掌握 GPIO 输出低电平的设置、GPIO 输出高电平的设置、流水灯和对嵌入式 C 语言编程的了解。

一、GPIO 输出清零寄存器(IOxCLR,x=0,1,2,3)

LPC2000 的 GPIO 功能是通过软件配置寄存器来实现的,GPIO 相关寄存器结构如图 4-2 所示,其中 IOxCLR 为清零寄存器。

当引脚配置为 GPIO 输出模式时,可使用 IOxCLR 寄存器从引脚输出低电平。写入 1 使对应引脚输出低电平,写入 0 无效。

如果一个引脚被配置为输入或第二功能,写 IOCLR 无效。IOCLR 寄存器描述见表 4-10。

表 4-10　IOCLR 寄存器描述

IOCLR	描　述
31:0	输出清零。 IO0CLR 的位 0 对应于 P0.0,……,位 31 对应于 P0.31。 IO1CLR 的位 1 对应于 P1.0,……,位 31 对应于 P1.31。 IO2CLR 的位 2 对应于 P2.0,……,位 31 对应于 P2.31。 IO3CLR 的位 3 对应于 P3.0,……,位 31 对应于 P3.31

例 4.4　设置 P0.0 输出低电平。

```
/*程序清单 P0.0 输出低电平*/
PINSEL0 = 0x000000000;          //设置引脚连接模块,P0.0 为 GPIO
IO0DIR = 0x00000001;            //设置 P0.0 口方向,设置为输出
IO0CLR = 0x00000001;            //设置 P0.0 口状态,输出低电平
```

二、GPIO 输出置位寄存器(IOxSET,x=0,1,2,3)

LPC2000 的 GPIO 功能是通过软件配置寄存器来实现的,GPIO 相关寄存器结构如图 4-2 所示,其中 IOxSET 为置位寄存器。

当引脚配置为 GPIO 输出模式时,可使用 GPIO 输出置位寄存器从引脚输出高电平,写入 1

使对应引脚输出高电平，写入 0 无效。如果一个引脚被配置为输入或第二功能，写 IOSET 无效。

注意：读 IOSET 寄存器返回 GPIO 输出寄存器中的值，该值由前一次对 IOSET 和 IOCLR（或前面提到的 IOPIN）的写操作决定。该值不反映任何外部环境对引脚的影响。

IOSET 寄存器描述见表 4-11。

表 4-11　IOSET 寄存器描述

IOSET	描　　述
31:0	输出置位。 IO0SET 的位 0 对应于 P0.0，……，位 31 对应于 P0.31。 IO1SET 的位 0 对应于 P1.0，……，位 31 对应于 P1.31。 IO2SET 的位 0 对应于 P2.0，……，位 31 对应于 P2.31。 IO3SET 的位 0 对应于 P3.0，……，位 31 对应于 P3.31

GPIO 使用注意事项：如果指定输出引脚在 GPIO 输出置位寄存器（IOxSET）和 GPIO 输出清零寄存器（IOxCLR）中的对应位都置位，那么引脚的输出电平取决于后写入的寄存器的值。举例如下。

```
IO0SET = 0x00000080;
IO0CLR = 0x00000080;
```

P0.7 输出电平为低，因为写 GPIO 清零寄存器在写置位寄存器之后。

例 4.5　设置 P0.0 输出高电平（见图 4-2）。

```
/* 程序清单 P0.0 输出高电平 */
PINSEL0 = 0xFFFFFFFC;          //设置引脚连接模块，P0.0 为 GPIO
IO0DIR = 0x00000001;           //设置 P0.0 口方向，设置为输出
IO0SET = 0x00000001;           //设置 P0.0 口状态，输出高电平
```

三、流水灯

LED 是嵌入式系统中最为常见的信号指示灯。由多个 LED 组成一个阵列，通过程序控制 LED 的亮和灭，依次逐个点亮的时候就像流水一样，这时通常称之为流水灯。虽然流水灯的设计较为简单，但它的变化形式却多种多样，设计者可以让一组 LED 按照自己的设想来变幻，许多嵌入式工程师都是从流水灯电路开始做起的。

四、嵌入式 C 语言基本知识

嵌入式 C 语言和标准 C 语言在数据类型、程序结构等方面基本相同，也扩充了一些关键字等，但开发环境有所不同。下面主要介绍一些嵌入式 C 语言中的重点、难点和不同于标准 C 语言的重要知识点。

1. 函数

在 C 语言中，就是用函数来实现功能模块的定义，C 程序的功能可以通过函数之间的调用来实现。一个完整的 C 程序可以由多个源程序文件组成，而每一个文件中都可以包含多个函数。

C 程序是由一个主函数和其他若干函数构成的，每个函数实现一定的功能，其中主函数

main()是必需的,其他函数被主函数调用或者其他函数之间相互调用。C 语言的函数可以分为三类:主函数 main()、库函数[如 printf()、scanf()等]和用户自定义函数。

1)函数定义

函数定义的格式如下。

```
类型说明符 函数名(数据类型 形式参数 1,数据类型 形式参数 2, ……)
{
  函数体;
}
```

其中,类型说明符和函数名为函数头。类型说明符指明了本函数的类型,函数的类型实际上是函数返回值的类型。该类型说明符与前面介绍的各种说明符相同。函数名是由用户定义的标识符,函数名后有一对圆括号,即使其中无参数,括号也不可少。

{ }中的内容称为函数体。在函数体的声明部分,是对函数体内部所用到的变量的类型说明。

在很多情况下都要求函数有返回值,如果函数没有返回值,那么函数类型说明符写为 void。

在 C 程序中,一个函数的定义可以放在任意位置,既可放在主函数 main()之前,也可放在主函数 main()之后。

C 语言中的 return 语句用于调用函数返回值,语法如下。

```
return (<表达式>);
```

C 语言中的函数只可以返回一个值,不能返回多个值。

返回值的数据类型必须与函数原型中返回值的数据类型相匹配。

当遇到 return 语句时,函数执行将终止。程序控制流将立即返回调用函数。

2)函数分类

(1)主函数。主函数是由系统定义的,其函数结构如下。

```
int  main( )
{
  说明部分程序段(定义数据类型)      //变量定义等,简单程序可能没有此部分
  执行部分程序段(给出操作指令)      //输入、输出、控制、赋值、计算等
  return 0;                          //返回值
}
```

(2)标准函数。标准函数即库函数,它是由系统提供的,用户不必自己定义这些函数,可以直接使用它们。应该说明,不同的 C 系统提供的库函数的数量和功能不同,当然有一些基本的函数是共同的。标准函数需要文件包含,再调用。

(3)自定义函数。自定义函数即用户自己定义的函数,用以解决用户的专门需要。自定义函数先定义,后调用。

从是否含参角度来说,所有函数包括无参函数和有参函数。

(1)无参函数。无参函数定义的一般形式为

```
类型标识符或 void  函数名(void)
{
    说明语句部分;
    可执行语句部分;
```

```
}
```

示例如下。

```
void  Delay()
{
    unsigned  int  i,j;
    for(j=65500; j>0; j--)
    {
        for(i=0; i<5000; i++);
    }
}
```

(2)有参函数。有参函数定义的一般形式为

```
函数类型  函数名(数据类型 参数 1,数据类型 参数 2……)
{
    说明语句部分;
    可执行语句部分;
}
```

有参函数比无参函数多了一个参数表列。调用有参函数时,调用函数将赋予这些参数实际的值。

示例如下。

```
void  DelayMS(unsigned  int  dly)
{
    uint32  i;
    for(; dly>0; dly--)
    {
      for(i=0; i<5000; i++);
    }
}
```

3)函数调用

函数只有在被调用时才能执行。当一个函数调用另一个函数时,程序就从主调函数转移到被调用函数,并且从被调用函数的函数体起始位置开始执行,直到函数体执行结束,返回到主调函数的调用位置继续往下执行。

函数调用的一般形式如下。

```
函数名(实参表列);
```

当实参表列中含有多个参数时,则各参数用逗号间隔。

函数调用方式有 3 种。

(1)把函数调用作为一个语句。

示例:调用延时子程序 void DelayMS(unsigned int dly)

```
DelayMS(10);
```

(2)函数出现在一个表达式中。

示例：unsigned int max(unsigned int i，unsigned int j)函数出现在一个表达式中

```
c=2*max(a,b);
```

(3)函数调用作为一个函数的实参。

示例：unsigned int max(unsigned int i，unsigned int j)函数调用作为 printf 的实参

```
printf("%d",max(a,b));
```

实参和形参必须个数相等、类型一致、顺序对应，进行数据的“值传递”。实参可以是常量、变量、表达式或函数。函数同变量一样，在调用前应该在主调函数中事先说明，即“声明”。

下面两种情况可以不用声明。

①被调用函数的定义的位置在主调函数之前。

②被调用函数是整型 int。

为了避免声明问题，我们在实际应用中几乎都将函数的定义放在主函数的前面。

2. 预处理命令

预处理命令以“#”开头，如包含命令 #include、宏定义命令 #define 等。一般都放在源文件的前面，它们称为预处理部分。

预处理是指在进行编译之前所做的工作。预处理是 C 语言的一个重要功能，它由预处理程序负责完成。当对一个源文件进行编译时，系统将自动引用预处理程序对源程序中的预处理部分做处理，处理完毕自动进入对源程序的编译。C 语言提供了多种预处理功能，如宏定义、文件包含、条件编译等。

下面介绍几种常用的预处理功能。

1)宏定义

define 是 C 语言中的预处理命令，它用于宏定义，可以提高源代码的可读性，为编程提供方便。

在 C 语言源程序中允许用一个标识符来表示一个字符串，称为“宏”。被定义为“宏”的标识符称为“宏名”。

格式：#define 标识符 字符串

其中，“标识符”为所定义的宏名；“字符串”可以是常数、表达式、格式串等。

示例如下。

```
#define  PI  3.14            //指定标识符 PI 来代替常数 3.14
#define  PORT0  0            //指定标识符 PORT0 来代替常数 0
```

在编写源程序时，用到 3.14 的地方都可用 PI 代替，如果修改 3.14 的值，不需要在程序中去修改，只修改宏定义处就可以了。

#undef 指令用来删除事先定义的宏定义，其一般形式为

```
#undef  宏替换名
```

示例如下。

```
#define  TRUE
...
```

```
#undef  TRUE
```

#undef 主要用来使宏替换名只限定在需要使用它们的程序段中。

2)文件包含

文件包含是 C 预处理程序的另一个重要功能。

文件包含命令行的一般形式为

```
#include  "文件名"or<文件名>
```

示例如下。

```
#include  "stdio.h"
```

文件包含命令的功能是把指定的文件插入该命令行位置取代该命令行,从而把指定的文件和当前的源程序文件连成一个源文件。

注意:包含命令中的文件名可以用双引号括起来,也可以用尖括号括起来。这两种形式是有区别的:使用尖括号表示在包含文件目录中去查找(包含目录是由用户在安装开发环境时设置的),而不在源文件目录去查找;使用双引号则表示首先在当前的源文件目录中查找,若未找到才到包含目录中去查找。用户编程时可根据自己文件所在的目录来选择某一种命令形式。

一个 include 命令只能指定一个被包含文件,若有多个文件要包含,则需用多个 include 命令。

3)条件编译

预处理程序提供了条件编译的功能。可以按不同的条件去编译不同的程序部分,因而产生不同的目标代码文件。这对于程序的移植和调试是很有用的。

条件编译有 3 种形式,下面分别介绍。

(1)第一种形式。

```
#ifdef  标识符
程序段 1
#else
程序段 2
#endif
```

它的功能是,如果标识符已被 #define 命令定义过,那么对程序段 1 进行编译,否则对程序段 2 进行编译。如果没有程序段 2(它为空),本格式中的 #else 可以没有,即可以写为

```
#ifdef  标识符
程序段
#endif
```

(2)第二种形式。

```
#ifndef  标识符
程序段 1
#else
程序段 2
#endif
```

与第一种形式的区别是将“ifdef”改为“ifndef”。它的功能是,如果标识符未被 #define

命令定义过，那么对程序段 1 进行编译，否则对程序段 2 进行编译。这与第一种形式的功能正相反。

(3)第三种形式。

```
#if  常量表达式
程序段 1
#else
程序段 2
#endif
```

它的功能是，如果常量表达式的值为真(非 0)，那么对程序段 1 进行编译，否则对程序段 2进行编译。因此，可以使程序在不同条件下，完成不同的功能。

任务解析

1. 电路分析

MagicARM2200 教学实验开发平台采用 P2.16～P2.23 控制 LED，电路如图 4-19 所示(将 JP23 全部短接)。

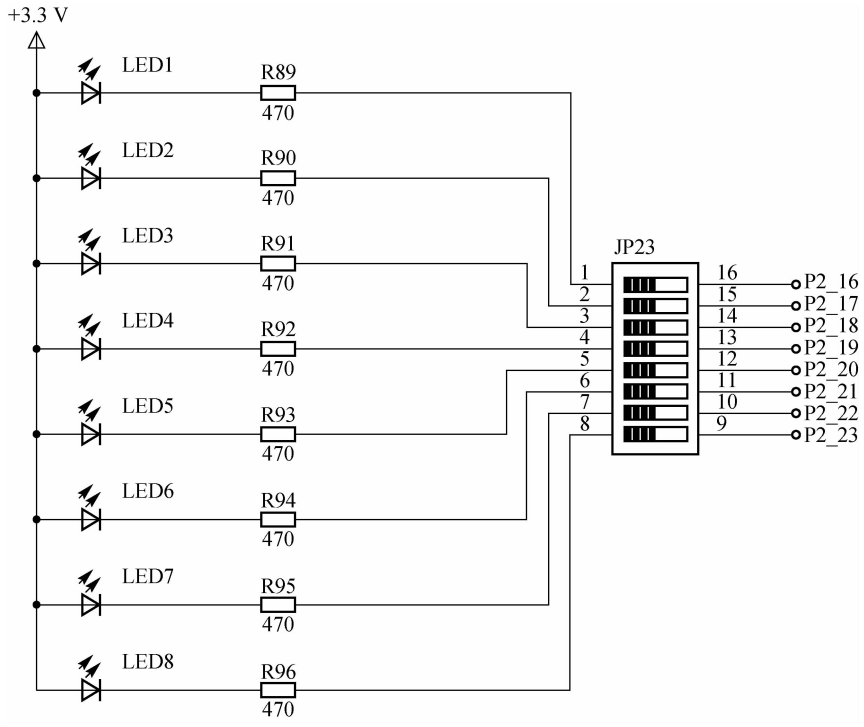

图 4-19　LED 连接电路

2. 软件分析

定义一个数组，将要显示的流水灯数据放在数组中，每隔一段时间取一次，实现流水灯的效果。流水灯控制程序流程图如图 4-20 所示。

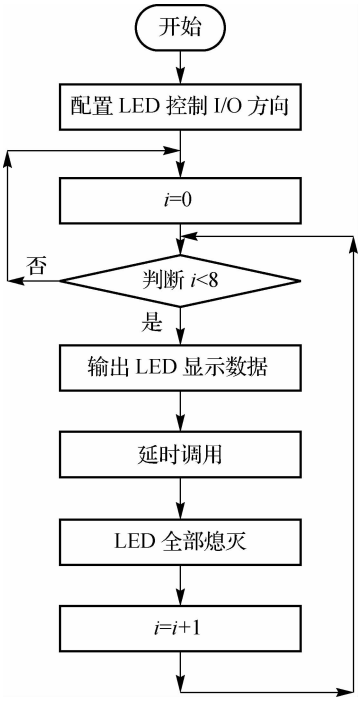

图 4-20　流水灯控制程序流程图

```
/*************************C 语言程序清单************************/
    #include  "config.h"
    #define   LED1   1<<16          //P2.16
    #define   LED2   1<<17          //P2.17
    #define   LED3   1<<18          //P2.18
    #define   LED4   1<<19          //P2.19
    #define   LED5   1<<20          //P2.20
    #define   LED6   1<<21          //P2.21
    #define   LED7   1<<22          //P2.22
    #define   LED8   1<<23          //P2.23
    #define   LEDCON   0x00ff0000
    const uint32   DISP_TAB[8] = { 0xff01ffff, 0xff02ffff, 0xff04ffff, 0xff08ffff,
0xff10ffff, 0xff20ffff, 0xff40ffff, 0xff80ffff};
    /*************************************************************
    **名        称:DelayNS
    **功        能:长软件延时
    **入口参数:dly   延时参数,值越大,延时越久
    **返 回 值:无
    *************************************************************/
    void   DelayNS(uint32   dly)
```

```
{   uint32  i;
    for(; dly>0; dly--)
    {
    for(i=0; i<5000; i++);
    }
}
/*************************************************************
**名        称:main
**功        能:根据表 DISP_TAB 来控制 LED 显示
**入口参数:无
**返 回 值:0
*************************************************************/
int  main(void)
{    uint8  i;
     IO2DIR = LEDCON;                  //配置 LED 控制 I/O 方向
     while(1)
      {
         for(i=0; i<8; i++)
        {
            IO2CLR = DISP_TAB[i];     //依次点亮 LED
            DelayNS(10);              //延时
            IO2SET = 0xffffffff;      //8 个 LED 都灭
        }
      }
      return(0);
}
```

任务实施

LPC2000 控制 8 个 LED 流水灯输出实验的操作步骤如下。

(1)连接 EasyJTAG 仿真器和 MagicARM2200 教学实验开发平台。将 EasyJTAG 仿真器的 25 针接口通过并口延长线与 PC 的并口连接,将 EasyJTAG 仿真器的 20 针接口通过 20 PIN 连接电缆接到 MagicARM2200 教学实验开发平台的 J3 上,将 MagicARM2200 教学实验开发平台上的 JP23 全部短接,JP15 全部断开,最后打开实验箱电源。

(2)打开 H-JTAG 软件,设置 H-JTAG 的配置(操作方法详见任务 4.1)。

(3)启动 ADS 1.2,使用 ARM Executable Image for MagicARM2200 工程模板建立一个工程 LEDCon_C(也可以直接复制一个完整的工程文件将其作为模板进行修改)。

(4)在 user 组的 main.c 中编写实验程序代码(详见前面的任务解析)。

(5)选用 DebugInExram 生成目标,然后编译链接工程。

(6)执行 Project→Debug 菜单命令,启动 AXD 进行 JTAG 仿真调试(需要正确设置仿真器,参考 ADS 集成开发环境及仿真器应用)。

(7)全速运行程序,程序会在 main.c 的主函数中停止(因为调试选项设置第一次启动运行停在 main 处)。

(8)全速运行程序,观察 8 个 LED 流水灯的状态。

(9)当仿真调试通过后关闭 AXD,在 ADS 1.2 集成开发环境中选用 RelOutChip 生成目标,然后编译链接工程。

(10)执行 Project→Debug 菜单命令,启动 AXD 进行 JTAG 仿真调试。此时 EasyJTAG 仿真器将会把程序下载到片外 Flash 上(需要正确设置仿真器,见图 4-21)。

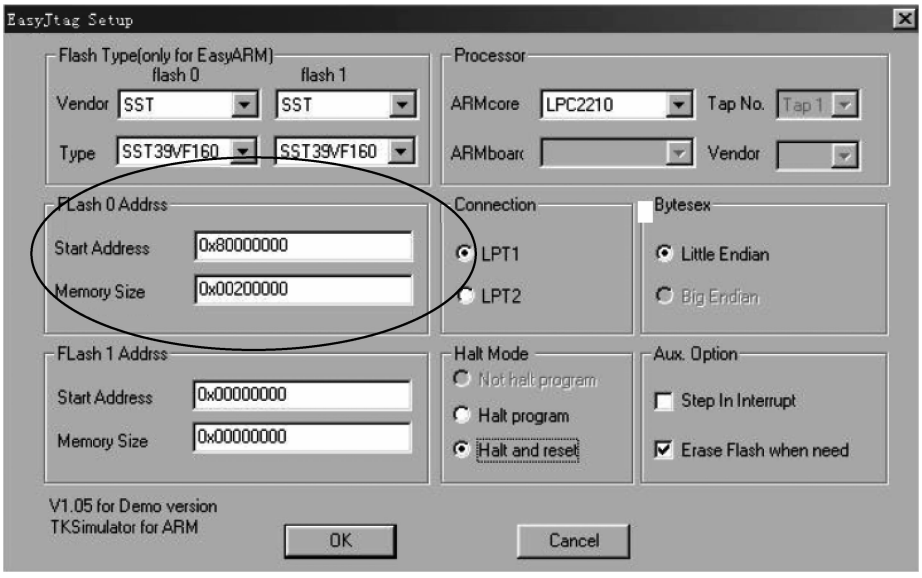

图 4-21 片外 Flash 调试的仿真器设置

(11)按 MagicARM2200 教学实验开发平台上的 RST 复位键,观察程序是否能脱机运行。

(12)实验结束后,在 AXD 中设置仿真器为片外 RAM 调试方式的设置,以便于后面实验的正确操作。

项目小结

LPC2000 的基本输入输出控制方式,设置 LPC2000 的引脚功能为 GPIO 功能,有相应的 4 个寄存器,即 PINSEL0～3。除此之外,还有引脚方向寄存器 IOxDIR、引脚值寄存器 IOxPIN、输出置位寄存器 IOxSET 和输出清零寄存器 IOxCLR。

思考与练习

1. 什么是 GPIO?

2. 如何设置 GPIO 为输入和输出方式?

3. LPC2114 有几组 GPIO 端口? LPC2214 有几组 GPIO 端口?

4. 如何实现利用 LPC2290 控制 LED(连接 P2.20)的闪烁? 并编程实现。

5. 用 LPC2290 控制 8 个 LED(连接 P2.16～P2.23),实现每隔一个亮的状态。