第3章　I/O 端口的 C 语言程序设计

3.1 IO 端口的结构

输入/输出接口（也可简称为 I/O 口）是单片机与外设交换信息的通道。输入端口负责从外界接收检测信号、键盘信号等各种开关量信号。输出端口负责向外界输送由内部电路产生的处理结果、显示信息、控制命令、驱动信号等。unSP 内有并行和串行两种方式的 I/O 口。SPCE061A 有两个 16 位通用的并行 I/O 口：A 口和 B 口。这两个口的每一位都可通过编程单独定义成输入或输出口。

A 口的 IOA0~IOA7 用作输入口时具有唤醒功能，即具有输入电平变化引起 CPU 中断功能。在那些用电池供电、追求低能耗的应用场合，可以应用 CPU 的睡眠模式（通过软件设置）以降低功耗，需要时以按键来唤醒 CPU，使其进入工作状态。例如：手持遥控器、电子字典、PDA、计算器、移动电话等。

图 3.1 是 SPCE061A 的 I/O 端口结构图。与其它的单片机相比，它除了每个 I/O 端口可以单独定义其状态外，每个对应状态下的 I/O 端口性质电路都是内置的，在实际的电路中不需要再次外接。例：设端口 A 口为带下拉电阻的输入口，在连接硬件时无需在片外接下拉电路。

■

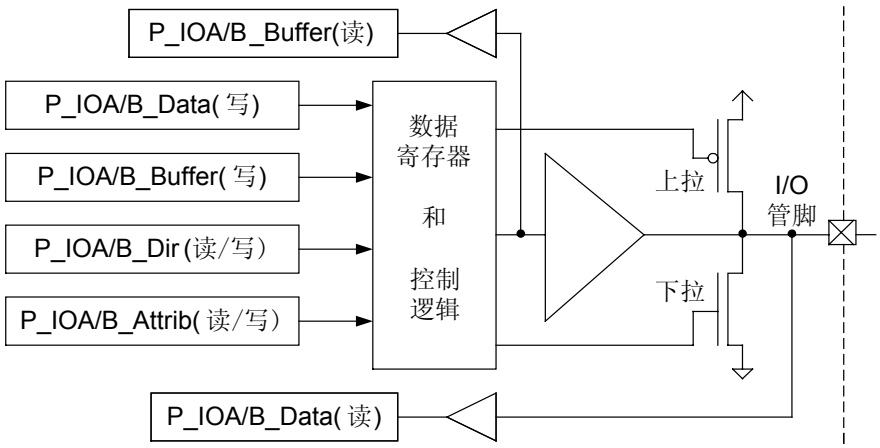

图3.2 I/O 端口结构图

3.2 IO 端口设置的寄存器

SPCE061A 提供了位控制结构的 I/O 端口，每一位都可以被单独定义用于输入或输出数据。通常，对某一位的设定包括以下 3 个基本项：数据向量 Data、属性向量 Attribution 和方向控制向量 Direction。3 个端口内每个对应的位组合在一起，形成一个控制字，用来定义相应 I/O 口位的输入输出状态和方式。例如，假设需要 IOA0 是下拉输入管脚，则相应的 Data、Attribution 和

Direction 的值均被置为“0”。如果需要 IOA1 是带唤醒功能的悬浮式输入管脚，则 Data、Attribution 和 Direction 的值被置为“010”。

A 口和 B 口的 Data、Attribution 和 Direction 的设定值均在不同的寄存器里，用户在进行 I/O 口设置时要特别注意这一点。

3.2.1 A 口相应的寄存器

P_IOA_Data(读/写)($7000H)

A 口的数据单元，用于向 A 口写入或从 A 口读出数据。当 A 口处于输入状态时，读出是读 A 口管脚电平状态； 写入是将数据写入 A 口的数据寄存器。当 A 口处于输出状态时，写入输出数据到 A 口的数据寄存器。

P_IOA_Buffer (读/写) ($7001H)

A 口的数据向量单元，用于向数据向量寄存器写入或从该寄存器读出数据。当 A 口处于输入状态时，写入是将 A 口的数据向量写入 A 口的数据寄存器；读出则是从 A 口数据寄存器内读其数值。当 A 口处于输出状态时，写入输出数据到 A 口的数据寄存器。

P_IOA_Dir(读/写)($7002H)

A 口的方向向量单元，用于用来设置 A 口是输入还是输出，该方向控制向量寄存器可以写入或从该寄存器内读出方向控制向量。Dir 位决定了口位的输入/输出方向：即‘0’为输入，‘1’为输出。

P_IOA_Attrib(读/写)($7003H)

A 口的属性向量单元，用于 A 口属性向量的设置。

P_IOA_Latch(读)($7004H)

读该单元以锁存 A 口上的输入数据，用于进入睡眠状态前的触键唤醒功能的启动。

3.2.2 并行 I/O 口的组合控制

方向向量 Dir、属性向量 Attrib 和数据向量 Data 分别代表三个控制口。这三个口中每个对应的位组合在一起，形成一个控制字，来定义相应 I/O 口位的输入/输出状态和方式。

表 3.1 具体表示了如何通过对 I/O 口位的方向向量位 Dir、属性向量位 Attrib 以及数据向量位 Data 进行编程，来设定口位的输入/输出状态和方式。

由表 3.1 可以得出以下一些结论：

Dir 位决定了口位的输入/输出方向：即‘0’为输入，‘1’为输出。

Attrib 位决定了在口位的输入状态下是为悬浮式输入还是非悬浮式输入：即‘0’为带上拉或下拉电阻式输入，而‘1’则为悬浮式输入。在口位的输出状态下则决定其输出是反相的还是同相的；‘0’为反相输出，‘1’则为同相输出。

Data 位在口位的输入状态下被写入时，与 Attrib 位组合在一起形成输入方式的控制字‘00’、‘01’、‘10’、‘11’，以决定输入口是带唤醒功能的上拉电阻式、下拉电阻式或悬浮式以及不带唤醒功能的悬浮式输入。Data 位在口位的输出状态下被写入的是输出数据，不过，数据是经过反相器输出还是经过同相缓存器输出要由 Attrib 位来决定。

例如，假设要把 A 口的 Bit0 定义成下拉电阻式的输入口，则 A 口_Dir、_Attrib 和_Data 向量的三个相应的 Bit0 应组合设为‘000’。如果想把 A 口的 Bit1 定义成悬浮式并具有唤醒功能的输入口，只需将 Dir、Attrib 和 Data 向量中相应的 Bit1 组合设置为‘010’即可。

A 口的 IOA0~IOA7 作为唤醒源，常用于键盘输入。要激活 IOA0~IOA7 的唤醒功能，必须读 P_IOA_Latch 单元，以此来锁存 IOA0~IOA7 管脚上的键状态。随后，系统才可通过指令进入

低功耗的睡眠状态。当有键按下时，IOA0~IOA7 的输入状态将不同于其在进入睡眠前被锁存时的状态，从而引起系统的唤醒。

■

表3.1 I/O 端口的组合控制设置

Direction	Attribution	Data	功能	是否带唤唤醒功能	功能描述
0	0	0	下拉*	是**	带下拉电阻的输入管脚
0	0	1	上拉	是**	带上拉电阻的输入管脚
0	1	0	悬浮	是**	悬浮式输入管脚
0	1	1	悬浮	否	悬浮式输入管脚
1	0	0	高电平输出 (带数据反相器)	否	带数据反相器的高电平输出 (当向数据位写入“0” 时输出“1”)
1	0	1	低电平输出 (带数据反相器)	否	带数据反相器的低电平输出 (当向数据位写入“1”时输出“0”)
1	1	0	低电平输出	否	带数据缓存器的低电平输出 (无数据反相功能)
1	1	1	高电平输出	否	带数据缓存器的高电平输出 (无数据反相功能)

注：

*：口位默认为带下拉电阻的输入管脚；

* *：只有当 IOA [7~0]内位的控制字为 000，001 和 010 时， 相应位才具有唤醒的功能。

3.2.3 B 口相应的寄存器

P_IOB_Data(读/写)($7005H)

B 口的数据单元，用于向 B 口写入或从 B 口读出数据。当 B 口处于输入状态时，读出是读 B 口管脚电平状态； 写入是将数据写入 B 口的数据寄存器。当 B 口处于输出状态时，写入输出数据到 B 口的数据寄存器。

P_IOB_Buffer(读/写)($7006H)

B 口的数据向量单元，用于向数据寄存器写入或从该寄存器内读出数据。当 B 口处于输入状态时，写入是将数据写入 B 口的数据寄存器；读出则是从 B 口数据寄存器里读其数值。当 B 口处于输出状态时，写入数据到 B 口的数据寄存器。

P_IOB_Dir(读/写)($7007H)

B 口的方向向量单元，用于设置 IOB 口的状态。‘0’为输入，‘1’为输出。

P_IOB_Attrib(读/写)($7008H)

B 口的属性向量单元，用于设置 IOB 口的属性。

3.2.4 B 口的特殊功能

正如前面提到的，B 口除了具有常规的输入/输出端口功能外，还有一些特殊的功能，如下表 3.2 所示：

表3.2 B 口的特殊功能表

口位	特殊功能	功能描述	备注
IOB0	SCK	串行接口 SIO 的时钟信号	参见串行设备接口 SIO 的功能设置内容
IOB1	SDA	串行接口 SIO 的数据传送信号	参见串行设备接口 SIO 的功能设置内容
IOB2	EXT1	外部中断源(下降沿触发)	IOB2 设为输入状态
	Feedback_Output1	与 IOB4 组成一个 RC 反馈电路，以获得振荡信号，作为外部中断源 EXT1	设置 IOB2 为反相输出方式，见 P_FeedBack(写)($7009H)单元的描述
IOB3	EXT2	外部中断源(下降沿触发)	IOB3 设为输入状态
	Feedback_Output2	与 IOB5 组成一个 RC 反馈电路，以获得一个振荡信号，作为外部中断源 EXT2	设置 IOB3 为反相输出方式
IOB4	Feedback_Input1		
IOB5	Feedback_Input2		
IOB6	---		
IOB7	Rx	通用异步串行数据接收端口	参见通用异步串行接口部分
IOB8	APWMO	TimerA 脉宽调制输出	参见定时器/计数器部分，P_Feedback(单元
IOB9	BPWMO	TimerB 脉宽调制输出	参见定时器/计数器部分
IOB10	Tx	通用异步串行数据发送端口	参见通用异步串行端口 UART 部分

注：

1．口位默认为带下拉电阻的输入管脚

2．PWM：脉宽调制(Pulse Width Modulation)

3.3 IO 端口设置的 C 库函数

SPCE061.lib 中提供了相应的 API 函数如下所示：

函数原型

void Set_IOA_Dir(unsigned int);

void Set_IOB_Dir(unsigned int);

功能说明　设置 IO Dircetion 信息

用法　Set_IOA_Dir(Direction_A);

Set_IOB_Dir(Direction_B);

参数　1 代表输出，0 代表输入

返回值　无

函数原型

unsigned int Get_IOA_Dir(void);

unsigned int Get_IOB_Dir(void);

功能说明　获取 IO Dircetion 信息

用法　Direction_A ＝Get_IOA_Dir();

Direction_B ＝Get_IOB_Dir();

参数　无

返回值　1 代表输出，0 代表输入

函数原型

void Set_IOA_Attrib(unsigned int);
void Set_IOB_Attrib(unsigned int);

功能说明　设置 IO Attribution 信息

用法　Set_IOA_Attrib (Attribution_A);
Set_IOA_Attrib (Attribution_B);

参数

返回值　无

函数原型

unsigned int Get_IOA_Attrib(void);
unsigned int Get_IOB_Attrib(void);

功能说明　获取 IO Attribution 信息

用法　Attribution_A ＝Set_IOA_Attrib ();
Attribution_B ＝Set_IOA_Attrib ();

参数　无

返回值

函数原型

void Set_IOA_Data(unsigned int);
void Set_IOB_Data(unsigned int);

功能说明　设置 IO Data 信息

用法　Set_IOA_Data(Data_A);
Set_IOB_Data(Data_B);

参数　1 代表高电平，0 代表低电平

返回值　无

函数原型

unsigned int Get_IOA_Data(void);
unsigned int Get_IOB_Data(void);

功能说明　获取 IO Data 信息

用法　Data_A ＝Set_IOA_Data();
Data_B ＝Set_IOB_Data();

参数　无

返回值　1 代表高电平，0 代表低电平

函数原型

void Set_IOA_Buffer(unsigned int);
void Set_IOB_Buffer(unsigned int);

功能说明　设置 IO Buffer 信息

用法　Set_IOA_Buffer(Buffer_A);
Set_IOB_Buffer(Buffer_B);

参数　　1 代表高电平，0 代表低电平
返回值　无

函数原型

unsigned int Get_IOA_Buffer(void);
unsigned int Get_IOB_Buffer(void);
功能说明　　获取 IO Buffer 信息
用法　　Buffer_A ＝Set_IOA_Buffer();
Buffer_B ＝Set_IOB_Buffer();
参数　　无
返回值　1 代表高电平，0 代表低电平

函数原型

void Get_IOA_Latch(void);
功能说明　　读 P_IOA_Latch 单元，以此来锁存 IOA0~IOA7 管脚上的键状态
用法　　Get_IOA_Latch();
参数　　无
返回值　无

另：

sp_lib.asm 中定义了两个很有用的 IO API，在 C 中可以调用。它们是 SP_Init_IOA()，SP_Init_IOB()。

函数原型

void SP_Init_IOA(unsigned int, unsigned int, unsigned int);
void SP_Init_IOB(unsigned int, unsigned int, unsigned int);
功能说明　　同时设置 IO Dircetion、Attribution 和 Data 信息
用法　　SP_Init_IOA(Direction_A, Data_A, Attribution_A);
SP_Init_IOB(Direction_B, Data_B, Attribution_B);
参数
返回值　无

3.4 IO 端口的应用实例

例 3．1

硬件连接图如图 3.2 所示。A 口低八位接 1×8 键盘，B 口低八位接八个 LED，要求按每个按键的时候相应的 LED 被熄灭。

分析：显然，应该把 A 口的低八位设置为输入，B 口的低八位设置为输出。我们从图 3.2 看出，按键按下的时候会给 A 口一个高电平。为了使 A 口在按键触发后回到不被触发的状态，我们把 A 口设置为带下拉电阻的输入。同时，我们注意到 8 个 LED 是共阳极的（注意，本书有不少例程用到 8 个 LED，如果没有特别声明，指的都是共阳极的），低电平点亮，高电平熄灭。所以，我们在把 B 口设置为同相输出的时候，要让某个 LED 熄灭，只需

在对应的 IO 口位输出高电平。

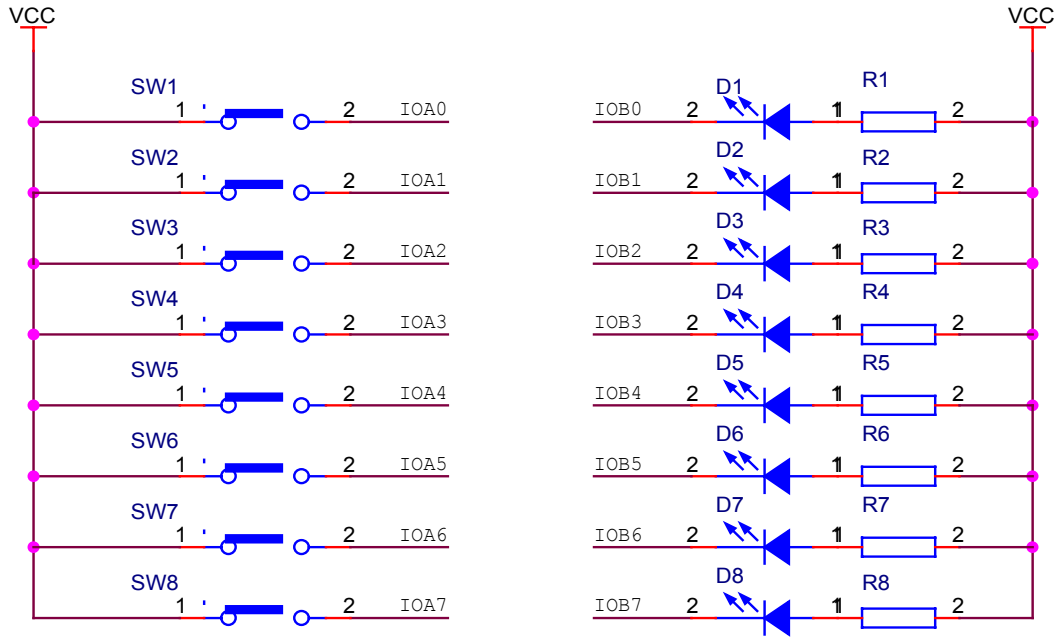

图3.2 例 3.1 的硬件连接图

实现的主程序如下所示：

```
#include  "SPCE061.H"
int main()
{
    unsigned int I_Key;                  //定义一个变量，用以保存输入的键值

    //设置 A 口为带下拉电阻的输入                    //IO 初始化开始
    Set_IOA_Dir(0x0000);
    Set_IOA_Attrib(0x0000);
    Set_IOA_Data(0x0000);

    //设置 B 口为同相低电平输出
    Set_IOB_Dir(0xffff);
    Set_IOB_Attrib(0xffff);
    Set_IOB_Data(0x0000);                //IO 初始化完成

    while(1)
    {
        I_Key = Get_IOA_Data();          // 获取 A 口输入的键值
        switch(I_Key)                    // 散转
        {
            case 0x0000:                 // 无键按下
                break;
            case 0x0001:                 // 按下的键号为 1
```

```
                case 0x0002:                          // 按下的键号为 2
                case 0x0004:                          // 按下的键号为 3
                case 0x0008:                          // 按下的键号为 4
                case 0x0010:                          // 按下的键号为 5
                case 0x0020:                          // 按下的键号为 6
                case 0x0040:                          // 按下的键号为 7
                case 0x0080:                          // 按下的键号为 8
                Set_IOB_Buffer(I_Key);                // 把 A 口输入的键值送到 B 口
                    break;
                default:
                    break;
            }
        }
    }
```

在程序运行之前，还有三个步骤要做（这几个步骤在以后各章节例程中也要做的）：第一步，在 Tools\Options\Directies 下选择 Include files,把 SPCE061.H 加入;第二步，在 Project\Setting\Link\Library modules 下加入 SPCE061.lib。

以上程序 IO 初始化部分的代码，也可以只写以下两行：

```
SP_Init_IOA(0, 0, 0);
SP_Init_IOB(0xffff, 0, 0xffff);
```

如果读者不喜欢使用 SPCE061.H，也不想包含 SPCE061.lib，你也完全可以象下面这样写程序，其实，这里只是将 SPCE061.lib 中我们用到的部分代码摘出来了。

```
volatile unsigned int *P_IOA_Data= (unsigned int*)(0x7000);        // Data vector for IOA
volatile unsigned int *P_IOA_Dir = (unsigned int*)(0x7002);        // Direction vector for IOA
volatile unsigned int *P_IOA_Attrib = (unsigned int*)(0x7003);     // Attribute vector for IOA
volatile unsigned int *P_IOB_Data = (unsigned int*)(0x7005);       // Data vector for IOB
volatile unsigned int *P_IOB_Dir = (unsigned int*)(0x7007);        // Direction vector for IOB
volatile unsigned int *P_IOB_Attrib = (unsigned int*)(0x7008);     // Attribute vector for IOB
int main()
{
    unsigned int I_Key;                    //定义一个变量，用以保存输入的键值

    //设置 A 口为带下拉电阻的输入                    //IO 初始化开始
    *P_IOA_Dir  =0;
    *P_IOA_Attrib  =0;
    *P_IOA_Data  =0;

    //设置 B 口为同相低电平输出
    *P_IOA_Dir  =0xffff;
    *P_IOB_Attrib  =0xffff;
    *P_IOB_Data  =0;                              //IO 初始化完成
```

```
    …                                    //以下代码不变
    …
    ….
}
```

另外还有一种用 C 嵌入式汇编的方法来实现 IO 的初始化，在上一章有例子，在此不再赘述。总之对 IO 进行编程的方法很多，读者可以根据自己的习惯选用其中一种。