第8章　串行接口 SIO 和 UART 的 C 语言程序设计

8.1 串行口的硬件特性

SPCE061A 有两个串行接口，一个是 SIO，另一个是 UART。

SIO 提供了一个 1 位的串行接口，用于与其它设备进行数据通讯。在 SPCE061A 内通过 IOB0 和 IOB1 这 2 个端口实现与设备进行串行数据交换功能。其中，IOB0 用来作为时钟端口(SCK)，IOB1 则用来作为数据端口(SDA)，用于串行数据的接收或发送。

图 8.1 为 SIO 的读写操作时序。

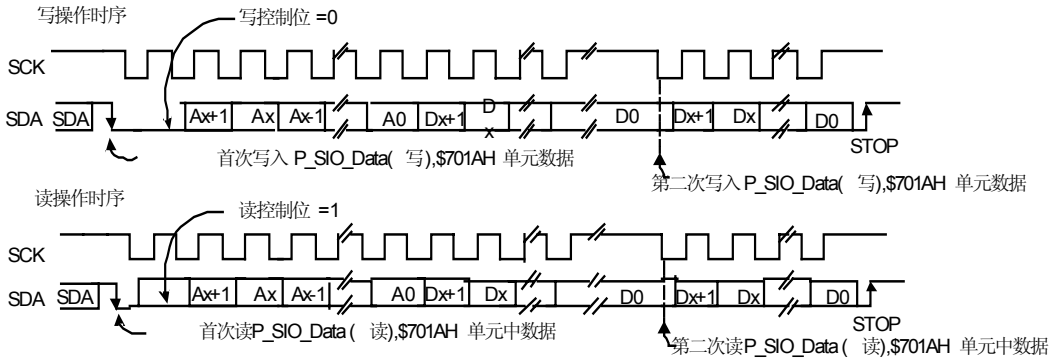

图8.1 SIO 的读写操作时序

UART 模块提供了一个全双工标准接口，用于完成 SPCE061A 与外设之间的串行通讯。借助于 IOB 口的特殊功能和 UART IRQ 中断，可以同时完成 UART 接口的接收发送数据的过程。此外，UART 还可以缓冲地接收数据。也就是说，它可以在读取缓存器内当前数据之前接收新的数据。但是，如果新的数据被接收到缓存器之前一直未从中读取先前的数据，会发生数据丢失。P_UART_Data ($7023H) (读/写)单元可以用于接收和发送数据的缓存,向该单元写入数据，可以将发送的数据送入缓存器；从该单元读数据，可以从缓存器读出数据字节。UART 模块的接收管脚 Rx 和发送管脚 Tx 分别与 IOB7 和 IOB10 共用。

使用 UART 模块进行通讯时，必须事先分别将管脚 Rx(IOB7)、Tx(IOB10)设置为输入状态、输出状态。然后，通过设置 P_UART_BaudScalarLow ($7024H)、P_UART_BaudScalarHigh ($7025H) 单元指定所需波特率。同时，设置 P_UART_Command1($7021H) 和 P_UART_Command2 ($7022H) 单元以激活 UART 通讯功能。以上设置完成后，UART 将处于激活状态。设置 P_UART_Command1 单元的第 6、7 位可以激活 UART IRQ 中断，并决定中断是由 TxRDY 或 RxRDY 信号触发以及由二者共同触发。设置 P_UART_Command2 单元的第 6、7 位可以激活 UART Tx、Rx 管脚功能。当 SPCE061A 接收或发送一个字节数据时，P_UART_Command2 ($7022H)单元的第 6、7 位被置为“1”且同时触发 UART IRQ。无论 UART IRQ 中断是否被激活，UART 接收/发送功能都可以由 P_UART_Command2 ($7022H)单元的第 6、7 位控制。在任意时刻读出 P_UART_Command2 ($7022H)单元将清除

UART IRQ 中断标志。

注意：UART IRQ（IRQ7）中断向量存储在 0xFFFFH 单元，相对于其它 IRQ 中断来说，该中断的优先级别最低。

图8.2 UART 数据帧的格式

8.2 串行口控制的寄存器

P_SIO_Ctrl(读/写)($701EH)

用户必须通过设置 P_SIO_Ctrl ($701EH) (读/写)单元的第 7 位，将 IOB0、IOB1 分别设置为 SCK 管脚和 SDA 管脚。如果该单元的第 6 位被设置为“0”，串行输入/输出接口可以从用户指定的地址读出数据。该单元的第 3、4 位的作用是让用户自行指定数据传输速度；而通过设置第 0、1 位，可以指定串行设备的寻址位数。

SIO 的控制选择在 P_SIO_Ctrl 单元内进行，详见表 8.1。

表8.1 P_SIO_Ctrl 单元

b7	b6	b5	b4	b3	b2	b1	b0	设置功能说明
SIO_Config	R/W	R/W_EN	Clock_Sel		—	Addr_Select		
X	X	X	X	X	—	0	0	串行设备地址(缺省)设置为 16 位(A0~A15)
X	X	X	X	X	—	0	1	无地址设置
X	X	X	X	X	—	1	0	串行设备地址设置为 8 位(A0~A7)
X	X	X	X	X	—	1	1	串行设备地址设置为 24 位(A0~A23)
X	X	X	0	0	—	X	X	数据传输速率设为 *CPUClk*/16(缺省设置)
X	X	X	0	1	—	X	X	无用
X	X	X	1	0	—	X	X	数据传输速率设为 *CPUClk*/8
X	X	X	1	1	—	X	X	数据传输速率设为 *CPUClk*/32
1	X	X	X	X	—	X	X	设置 IOB0=SCK(串行接口时钟端口)，IOB1=SDA (串行接口数据端口) 。用户不必设置 IOB0 和 IOB1 的输入输出状态
0	X	X	X	X	—	X	X	用作普通的 I/O 口(默认)
X	1	X	X	X	—	X	X	设置数据帧的写传输
X	0	X	X	X	—	X	X	设置数据帧的读传输(默认)
X	X	1	X	X	—	X	X	关断读/写帧的传输
X	X	0	X	X	—	X	X	接通读/写帧的传输(默认)

关于 CPU 时钟 CPU_CLK 请参考系统时钟部分。P_SIO_Data(读/写)($701AH)

P_SIO_Data(读/写)($701A)

该单元为接收/发送串行数据的缓冲单元。向该单元写入或读出数据，可按串行方式发送或接收数据字节。用户须通过写入 P_SIO_Start ($701FH)单元来启动 P_SIO_Data ($701AH)

单元与串行设备数据交换的过程。传输是从串行设备的起始地址(由 P_SIO_Addr_Low, P_SIO_Addr_Mid 和 P_SIO_Addr_High3 个单元指定)开始，然后是数据。

进行写操作时，第一次向 P_SIO_Data (写) ($701AH 如表 8.2)单元写入数值是在写入 P_SIO_Start(写)单元任意一个数值之后，即必须先启动数据传输，随后，SIO 将从串行设备的起始地址开始传送，后面接着传送写入 P_SIO_Data 单元中的 8 位数据。

进行读操作时，第一次读 P_SIO_Data (读) ($701AH)单元数据是在向 P_SIO_Start (写) ($701FH)单元写入任一数值后，SIO 将首先传送串行设备的起始地址。

表8.2　P_SIO_Data

b7	b6	b5	b4	b3	b2	b1	b0
D7	D6	D5	D4	D3	D2	D1	D0

P_SIO_Addr_Low(读/写)($701B)

串行设备起始地址的低字节(默认值为 00H)。

表8.3 P_SIO_ Addr_Low

b7	b6	b5	b4	b3	b2	b1	b0
A7	A6	A5	A4	A3	A2	A1	A0

P_SIO_Addr_Mid(读/写)($701CH)

串行设备起始地址的中字节(默认值为 00H)。

表8.4 P_SIO_Addr_ Mid

b7	b6	b5	b4	b3	b2	b1	b0
A15	A14	A13	A12	A11	A10	A9	A8

P_SIO_Addr_High(读/写)($701DH)

串行设备起始地址的高字节(默认值为 00H)。

表8.5 P_SIO_Addr_High

b7	b6	b5	b4	b3	b2	b1	b0
A23	A22	A21	A20	A19	A18	A17	A16

P_SIO_Start(读/写)($701FH)

向 P_SIO_Start(写)($701FH 如表 8.6)单元写入任意一个数值，可以启动数据传输。接着，当对 P_SIO_Data ($701AH) 单元读写操作时会使 SIO 根据 P_SIO_Addr_Low、P_SIO_Addr_Mid 和 P_SIO_Addr_High 的内容传输读/写操作的起始地址，之后再读写 P_SIO_Data 单元时 SIO 将不再传输此起始地址。

如果需要重新指定一个起始地址进行数据传输，用户可以向 P_SIO_Stop ($7020H)单元写入任意一个数值以停止 SIO 操作，然后向 P_SIO_Addr_Low、 P_SIO_Addr_Mid 和 P_SIO_Addr_High 写入新的地址；最后，向 P_SIO_Start(写)($701FH)单元写入任意一个数值重新启动 SIO 操作。

读出 P_SIO_Start($701FH)单元可获取 SIO 的数据传输状态，该单元的第 7 位 Busy 为占用标志位，Busy=‘1’表示正在传输数据，传输操作完成后，该位将被清为‘0’，可以开始传输新的数据字节。

表8.6　P_SIO_Start

b7	b6	b5	b4	b3	b2	b1	b0
Busy	-	-	-	-	-	-	-

P_SIO_Stop(写)($7020H)

向 P_SIO_Stop(写)($7020H)单元写入任一数值，可以停止数据传输。通常，停止数据传输的终止指令应出现在激活数据传输的启动指令之前。但上电复位后的第一个启动命令之前不需要终止命令。

P_UART_Command1(写)($7021H)

P_UART_Command1 单元为 UART 控制端口（如表 8.7 所示）。设置该单元的第 2、3 位可以控制数据奇偶校验功能。第 6、7 位控制着 UART IRQ 中断，二者的区别在于：如果第 6 位 TxIntEn=1，中断由 TxRDY 信号触发，即数据发送完毕将产生 UART IRQ 中断；如果第 7 位 RxIntEn=1，中断由 RxRDY 信号触发，即数据接收完毕将产生 UART IRQ 中断。如果该单元的第 5 位 I_Reset=1，所有 UART 控制寄存器、状态寄存器将恢复为系统默认值。P_UART_Command1(写)($7021H)单元的缺省值为$00。

表8.7 P_UART_Command1 单元

b7	b6	b5	b4	b3	b2	b1	b0	功能
RxIntEn	TxIntEn	I_Reset	-	Parity	P_Check	-	-	
1	—	—	—	—	—	—	—	允许 UART IRQ 中断(由 RxRDY 信号触发)
0	—	—	—	—	—	—	—	禁止 UART IRQ 中断
—	1	—	—	—	—	—	—	允许 UART IRQ 中断(由 TxRDY 信号触发)
—	0	—	—	—	—	—	—	禁止 UART IRQ 中断
—	—	1	—	—	—	—	—	内部复位信号复位
—	—	0	—	—	—	—	—	内部复位信号置位
—	—	—	—	1	—	—	—	激活偶校检功能
—	—	—	—	0	—	—	—	激活奇校检功能
—	—	—	—	—	1	—	—	激活奇偶校验功能
—	—	—	—	—	0	—	—	屏蔽奇偶校检功能

P_UART_Command2(写)($7022H)

该单元写入时为 UART 数据发送/接收控制端口，第 6、7 位分别控制着数据发送和接收管脚的允通/禁止。P_UART_Command2(写)($7022H)单元的缺省值为$00。

表8.8 P_UART_Command2(写)($7022H)

b7	b6	b5	b4	b3	b2	b1	b0
RxPinEn	TxPinEn	-	-	-	-	-	-
1：允通接收管脚 0：禁止接收管脚	1：允通发送管脚 0：禁止发送管脚						

此时 IOB7 和 IOB10 必须分别被设置为输入和输出管脚作为 Rx 和 Tx 管脚。当发送管脚被允通时，IOB10 Tx 输出管脚将自动被置为高电平。

P_UART_Command2 单元读出为 UART 状态信息。第 7 位是 RxRDY 标志位，当接收到数据时该标志位被置为“1”，读 P_UART_Data 单元将清除该标志位；第 6 位是 TxRDY 标志位，当通过写入本单元第 6 位为’1’来允通发送管脚后，该标志位被置为“1”，表示发送器的数据缓存器为空，已准备好，可以发送写入 P_UART_Data 单元的数据。

向 P_UART_Data 单元写入数据可以清除 TxRDY 标志位。读出 P_UART_Command2 ($7022H)单元的第 3~5 位是传输错误标志位，如果在传输过程中发生错误，相应位将被置为“1”；读 P_UART_Data($7023H)单元数据将清除错误标志位。

表8.9 P_UART_Command2（读）($7022H)

b7	b6	b5	b4	b3	2	1	0
RxRDY	TxRDY	FE	OE	PE			
1：数据接收完毕 0：未接收到数据	1：发送已准备好 0：发送未准备好	1：存在帧错误 0：无帧错误	1：存在溢出错误 0：无溢出错误	1：奇偶校验错误 0：奇偶校验无错			

以上错误信号代表传输过程可能出现的错误。给出了出错的原因及解决方法。

表8.10 出错原因及解决方法

错误类型	原因	解决方法
FE (帧错误)	发送管脚 TX 和接收管脚 RX 的数据帧的格式或波特率不一致	1. 使用一致的数据格式 2. 设置一致的波特率
OE (溢出错误)	接收端 RX 接收数据的速度低于发送端 TX 发送数据的速度，从而导致 RX 端数据溢出	1. 提高接收数据的速度 2. 降低数据传输速度
PE (奇偶校验错误)	传输条件差，可能有噪声干扰	改善传输条件

P_UART_Data(读/写)($7023H)

P_UART_Data 用于存放发送或接收到的数据，8bit。

P_UART_BaudScalarLow(读/写)($7024H)

P_UART_BaudScalarHigh (读/写)($7025)

P_UART_BaudScalarHigh ($7025)和 P_UART_BaudScalarLow($7024H)单元的组合控制数据的传输速率(波特率)。UART 波特率的计算公式如下：

波特率= (Fosc / 2) / Scale

其中 Scale=(Fosc/2)/波特率(Scale 为 7024H 单元和 7025H 单元组成的十进制整数)Fosc=24.576MHz(或 20.480MHz 等)，取决于 P_SystemClock 单元的第 5~7 位。

表 8.11 列出当 Fosc = 24.576MHz 时常用的波特率值。

表8.11 Fosc = 24.576MHz 时常用的波特率值

波特率(bps)	高字节($7025H)	低字节($7024H)	Scale (十进制)	实际波特率(bps)
1500 (最小值)	1FH	FFH	8192	1500
2400	14H	00H	5120	2400
4800	0AH	00H	2560	4800
9600	05H	00H	1280	9600
19200	02H	80H	640	19200
38400	01H	40H	320	38400
48000 (默认值)	01H*	00H*	256	48000
51200	00H	F0H	240	51200

57600	00H	D5H	213	57690
102400	00H	78H	120	102400
115200 (最大值)	00H	6BH	107	114841

8.3 串行口设置的 C 函数

SPCE061.lib 中提供了相应的 API 函数如下所示：

函数原型

void Set_SIO_Start(void);

功能说明 Set SIO Start，Send Start Signal to Start SIO Interface

用法 Set_SIO_Start(void);

参数 无

函数原型

unsigned int Get_SIO_Start(void);

功能说明 Get SIO Start

用法 SIO Busy Flag = Get_SIO_Start();

返回值 SIO Busy Flag →

Busy → 0x0080

Ready → 0x0000

函数原型

void Set_SIO_Stop(void);

功能说明 Set SIO Stop，Send End Signal to Stop SIO Interface

用法 Set_SIO_Stop();

参数 无

函数原型

void Set_SIO_Data(unsigned int);

功能说明 Set SIO Data

用法 Set_SIO_Data();

参数 SIO Data,it Range b7..b0

函数原型

int Get_SIO_Data(void);

功能说明 Get SIO Data

用法 SIO Data = Get_SIO_Data();

返回值 SIO Data,it range b7..b0

函数原型

void Set_SIO_Ctrl(unsigned int);

功能说明 Set SIO Control Register

用法　Set_SIO_Ctrl(SIO Ctrl Data);

参数　SIO Ctrl Data = Address Mode + Clock Select + RW_EN + R/W + SIO Config

SIO Ctrl Data contents →

Address Mode →

C_SIO_Addr8	→ Address = 8	(A7~A0)
C_SIO_Addr16	→ Address = 16	(A15~A0) (default)
C_SIO_Addr24	→ Address = 24	(A23~A0)

Clock Select →

C_SIO_Clk_Div_4	→ CPU CLK/4
C_SIO_Clk_Div_8	→ CPU CLK/8
C_SIO_Clk_Div_16	→ CPU CLK/16 (default)
C_SIO_Clk_Div_32	→ CPU CLK/32

RW_EN

C_SIO_RW_Dis	→ Read /Write control bit applied disable
C_SIO_RW_ENB	→ Read /Write control bit applied (default)

R/W

C_SIO_RD	→ SIO Read
C_SIO_WR	→ SIO Write

SIO Config

C_SIO_ENB	→ Enable SIO interface Configuration
C_SIO_Dis	→ Disable SIO interface(default)

函数原型

unsigned int Get_SIO_Ctrl(void);

功能说明　Get SIO Control Register

用法　SIO Ctrl Data = Get_SIO_Ctrl();

返回值　SIO Ctrl Data = Address Mode + Clock Select + RW_EN + R/W + SIO Config

函数原型

void Set_SIO_Address(unsigned long int);
void Set_SIO_Addr_Low(unsigned int);
void Set_SIO_Addr_Mid(unsigned int);
void Set_SIO_Addr_High(unsigned int);

功能说明　Set SIO Address

用法　Set_SIO_Address(SIO Address);
Set_SIO_Addr_Low(SIO Low Address);
Set_SIO_Addr_Mid(SIO Middle Address);
Set_SIO_Addr_High(SIO High Address);

参数　SIO Address，SIO Low Address，SIO Middle Address，SIO High Address，

Note: SIO Address = SIO High + Middle + Low Address

函数原型

unsigned long int Get_SIO_Address(void);
unsigned int Get_SIO_Addr_Low(void);
unsigned int Get_SIO_Addr_Mid(void);
unsigned int Get_SIO_Addr_High(void);

功能说明　Get SIO Address

用法　SIO Address = Get_SIO_Address();
SIO Low Address = Get_SIO_Addr_Low();
SIO Middle = Get_SIO_Addr_Mid();
SIO High Address = Get_SIO_Addr_High();

返回值　SIO Address，SIO Low Address，SIO Middle Address，SIO High Address，
Note: SIO Address = SIO High + Middle + Low Address

函数原型

void Set_UART_Command1(unsigned int);

功能说明　Set UART Command1 Register

用法　Set_UART_Command1(UART Command1 Register);

参数　UART Command1 Register =

C_UART_Parity_ENB	→ Enable UART Parity
C_UART_Parity_Odd	→ Enable UART Parity Odd
C_UART_Parity_Even	→ Enable UART Parity Even
C_UART_Reset	→ UART Internal Reset
C_UART_Tx_IRQ_ENB	→ UART TX IRQ Enable
C_UART_Rx_IRQ_ENB	→ UART RX IRQ Enable

函数原型

void Set_UART_Command2(unsigned int);

功能说明　Set UART Command2 Register

用法　Set_UART_Command2(UART Command2 Register);

参数　UART Command2 Register =

C_UART_Tx_Pin_ENB	→UART TX Pin Enable
C_UART_Rx_Pin_ENB	→UART RX Pin Enable

函数原型

unsigned int Get_UART_Command2(void);

功能说明　Get UART Command2 Register

用法　UART Command2 Register = Get_UART_Command2();

返回值　UART Command2 Register =

C_UART_Parity_Error	→ UARTParity Error
C_UART_OverRun_Error	→ UARTOver Run Error
C_UART_Frame_Error	→ UARTFrame Error
C_UART_Tx_RDY	→ UARTTX　Ready

C_UART_Rx_RDY　　→ UARTRX　Ready

函数原型

void Set_UART_Data(unsigned int);

功能说明　Set UART Data

用法　Set_UART_Data(UART Data);

参数　UART Data，range D7..D0

函数原型

unsigned int Get_UART_Data(void);

功能说明　Get UART Data

用法　UART Data = Get_UART_Data();

返回值　UART Data，range D7..D0

函数原型

void Set_UART_BaudRate(unsigned int);
void Set_UART_BaudScalarLow(unsigned int);
void Set_UART_BaudScalarHigh(unsigned int);

功能说明　Set UART Baud Rate

用法　Set_UART_BaudRate(Baud Rate);
Set_UART_BaudScalarLow(Low Baud Rate);
Set_UART_BaudScalarHigh(High Baud Rate);

参数　Baud Rate, Low Baud Rate, High Baud Rate
Note: Baud Rate = (Fosc / 2) / Scale = High + Low Baud Rate
User can use the pre-define baud rate ，running under Fosc=24.576MHz →

C_BaudRate_2400　　→ UART Baud Rate 2400 bps
C_BaudRate_4800　　→ UART Baud Rate 4800 bps
C_BaudRate_9600　　→ UART Baud Rate 9600 bps
C_BaudRate_19200　　→ UART Baud Rate 19200 bps
C_BaudRate_38400　　→ UART Baud Rate 38400 bps
C_BaudRate_48000　　→ UART Baud Rate 48000 bps

函数原型

unsigned int Get_UART_BaudRate(void);
unsigned int Get_UART_BaudScalarLow(void);
unsigned int Get_UART_BaudScalarHigh(void);

功能说明　Get UART Baud Rate

用法　Baud Rate = Get_UART_BaudRate();
Low Baud Rate = Get_UART_BaudScalarLow();
High Baud Rate = Get_UART_BaudScalarHigh();

返回值　Baud Rate, Low Baud Rate, High Baud Rate
Note: Baud Rate = High + Low Baud Rate

8.4 串行口应用实例

例 8.1 通过串行口扩展 Flash ROM SPR4096

通过串行输入/输出接口将 SPCE061A 与 SPR4096 连接起来，如图 8.2 所示，图 8.2 只画出了 SPCE061A 与 SPR4096 连接的线路。SPR4096 的详细资料请参见其 datasheet。

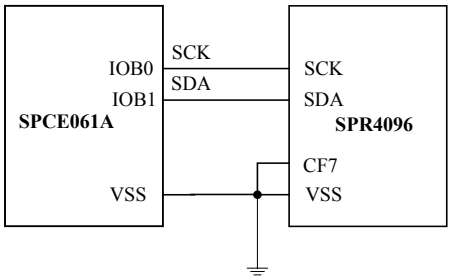

图8.2 SPCE061A 与 SPR4096 连接图

以下代码实现的功能是先对 SPR4096 进行 Mass Erase 操作，然后往 0x5678 单元写入 0x55AA 的值。注意，首先应将 SIO 设置为输出口。

```
#define        C_SIOCLOCK      0x0010;                  // CPUCLOCK/8
#include  "SPCE061V004.H"
main()
{
    long int i;
    unsigned long int ulAddr;
    unsigned int uiData;

    //初始化 SIO 口
    *P_IOB_Dir = 0xFFFF;
    *P_IOB_Attrib = 0xFFFF;
    *P_IOB_Data = 0x0003;
    *P_SIO_Ctrl = 0x00C3;

    // 对 SPR4096 进行 Mass Erase
    *P_SIO_Ctrl=0x00C0+C_SIOCLOCK;      // clk=    CPUclk/8, 16 位地址
    *P_SIO_Addr_Low=0x0000;             // SFLASH 地址
    *P_SIO_Addr_Mid=0x00C0;
    *P_SIO_Addr_High=0x00C0;
    *P_SIO_Start=1;                     // 启动写模式
    *P_SIO_Data=0;                      // A7~A0 = 0
    // 等待 SIO 空闲
    i=*P_SIO_Start;
    i&=0x0080;
    while(i)
    {
```

```
        i=*P_SIO_Start;
        i&=0x0080;
    }
    *P_SIO_Stop=1;
    i=0x7ff;                                    // 延时
    while(i--);


    // 往 0x5678 单元写入一个数据 0x55AA
    ulAddr=0x5678;
    *P_SIO_Addr_Low=ulAddr;                     // SFLASH 地址
    ulAddr>>=8;
    *P_SIO_Addr_Mid=ulAddr;
    ulAddr>>=8;
    ulAddr&=0x0007;
    *P_SIO_Addr_High=ulAddr;
    *P_SIO_Ctrl=0x00C3+C_SIOCLOCK;              // clk=CPUclk/8, 24 位地址
    *P_SIO_Start=1;                             // 启动写模式
    i=0x2ff;                                    // 延时
    while(i--);
    uiData=0x55AA;
    *P_SIO_Data=uiData;                         // 往 SIO 送数据 0x55AA
    // 等待 SIO 空闲
    i=*P_SIO_Start;
    i&=0x0080;
    while(i)
    {
        i=*P_SIO_Start;
        i&=0x0080;
    }
    *P_SIO_Stop=1;
}
```

例 8.2 用 UART 来接收 PC 机的 RS232 串行接口的数据

用 UART 来接收 PC 机的 RS232 串行接口的数据。如果接收到值为 79（大写的字符“O”）的 ACSII 码，就点亮和 IOA0～IOA7 相连的 LED，如果接收到值为 67（大写的字符“C”）的 ACSII 码，就熄灭和 IOA0～IOA7 相连的 LED。

本例的程序代码如下所示：

```
#include  "SPCE061V004.H"
main()
{
    unsigned int uiData,Ret;
    *P_IOA_Dir=0xffff;                          //IOA0~IOA7 初始化为输出
    *P_IOA_Attrib=0xffff;
```

```
    *P_IOA_Data=0xffff;

    *P_UART_Command1=0x20;                    //Uart 内部复位
    *P_UART_Command1=0x00;

    *P_UART_BaudScalarHigh=0x00;              //波特率设置为 115200bps
    *P_UART_BaudScalarLow=0x6B;

    *P_UART_Command1=0x000C;                  //允许接收
    *P_UART_Command2=0x00C0;

    Ret=*P_UART_Data;                         // 清接收缓冲区

    while(1)
    {
        Ret=*P_UART_Command2;
        Ret=Ret&0x0080;
        while(Ret==0)                         //等待接收完毕
        {
            Ret=*P_UART_Command2;
            Ret=Ret&0x0080;
            *P_Watchdog_Clear=C_WDTCLR;
        }
        uiData=*P_UART_Data;                  //读出接收的数据
        if(uiData==79)
            *P_IOA_Data=0x0000;               //点亮 LED
        else if(uiData==67)
            *P_IOA_Data=0xffff;               //熄灭 LED
    }
}
```