SPI、IIC 和 UART 是最常用的三种串行总线，这三种总线在 s3c2440 中都被集成了。在这里我们主要介绍 UART，另两个总线在后面的文章中给出。

UART（Universal Asynchronous Receiver/Transmitter，通用异步接收/发送装置）用于异步通信，可以实现全双工发送和接收。它不仅可以实现不同嵌入式系统之间的通信，还可以实现与 PC 之间的通信。

s3c2440 提供了三个 UART 端口，它们都可以通过查询、中断和 DMA 方式传输数据，而且每个 UART 都分别有一个 64 个字节的接收 FIFO 和一个 64 个字节的发送 FIFO。在这里，我们只给出非 FIFO 模式，即传输数据不利用 FIFO 缓存，一个字节一个字节地传输。

下面我们就给出如何用 s3c2440 来实现非 FIFO 的 UART 通信。要实现某种通信，就必须遵循该通信协议。UART 的协议包括传输数据的位数，停止位的位数，以及是否进行奇偶校验，这些设置是利用 ULCONn 寄存器完成的。另一个很重要的地方就是设置波特率。s3c2440 波特率的时钟源有三个：PCLK、FCLK/n 和 UEXTCLK。时钟源的选择是由 UCONn 的第 10 位和第 11 位来完成的。波特率的具体计算公式为：

时钟源频率÷（波特率×16）－1

这个计算结果很可能是小数，把该小数取最接近的整数，放入寄存器 UBRDIVn 中就完成了波特率的设置。如我们选择波特率的时钟源为 PCLK，它为 50MHz，我们设置的波特率为 115.2kHz，通过上式计算的结果为 26.13，取整后得到 26，那么我们把 26 放入 UBRDIVn 中即可。由于我们没有使用 FIFO 和 MODEM，所以可以不用设置 FIFO 控制寄存器 UFCONn 和 MODEM 控制寄存器 UMCONn。通过以上寄存器的设置，UART 就可以正常传输数据。

接收到的数据是放到接收缓存器 URXHn 中，要发送数据时，是把数据放入发送缓存器 UTXHn 中。由于 UART 是通过字节方式传输数据的，因此要区分是大端模式还是小端模式，也就是说这两个寄存器在这两种模式下，所在的地址是不同。为了了解当前数据传输的各种状态，还需要一些状态寄存器。传输状态寄存器 UTRSTATn 非常有用，它的第 0 位可以用来判断接受缓存器内是否有可接收的数据，第 1 位和第 2 位可以用来判断发送缓存器中是否为空，为空时可以发送数据。由于在这里我们不进行传输数据时错误的判断，因此错误状态寄存器 UERSTATn 不需要，FIFO 状态寄存器 UFSTATn 和 MODEM 状态寄存器 UMSTATn 在这里也不需要。

我们给出 UART 通信的两种方法：查询和中断。为了验证程序，使用任一款的串行通信软件来实现 PC 和 s3c2440 之间的通信即可。

首先给出的是查询程序。它是在主程序的循环体内不断查询 UART 端口，当有数据来时，就接收数据，并再通过 UART 发送该数据。然后根据所接收数据的不同，分别执行不同的内容，如点亮、熄灭 LED，蜂鸣器响、或不响。在这里，我们每次只完成一个字节的传输。

```
#define rGPBCON        (*(volatile unsigned *)0x56000010)          //Port B control

#define rGPBDAT        (*(volatile unsigned *)0x56000014)           //Port B data

#define rGPBUP         (*(volatile unsigned *)0x56000018) //Pull-up control B


#define rGPHCON        (*(volatile unsigned *)0x56000070)          //Port H control

#define rGPHUP         (*(volatile unsigned *)0x56000078)           //Pull-up control H


#define rULCON0         (*(volatile unsigned *)0x50000000)         //UART 0 Line control

#define rUCON0          (*(volatile unsigned *)0x50000004)         //UART 0 Control

#define rUFCON0         (*(volatile unsigned *)0x50000008)         //UART 0 FIFO control

#define rUMCON0          (*(volatile unsigned *)0x5000000c)       //UART 0 Modem control

#define rUTRSTAT0      (*(volatile unsigned *)0x50000010)          //UART 0 Tx/Rx status

#define rUERSTAT0      (*(volatile unsigned *)0x50000014)          //UART 0 Rx error status

#define rUFSTAT0       (*(volatile unsigned *)0x50000018)          //UART 0 FIFO status

#define rUMSTAT0       (*(volatile unsigned *)0x5000001c)          //UART 0 Modem status

#define rUBRDIV0       (*(volatile unsigned *)0x50000028)         //UART 0 Baud rate divisor
```

```
//little endian

#define rUTXH0 (*(volatile unsigned char *)0x50000020) //UART 0 Transmission Hold

#define rURXH0 (*(volatile unsigned char *)0x50000024) //UART 0 Receive buffer


void Main(void)

{

        char ch;

        rGPBCON = 0x015551;

rGPBUP    = 0x7ff;

rGPBDAT = 0x1e0;


rGPHCON = 0x00faaa;                    //使用 UART0 功能

rGPHUP    = 0x7ff;


rULCON0 = 0x3;                              //设置 UART0 无奇偶校验，一位停止位，8 位
数据

        rUCON0 = 0x245;                    //PCLK 为时钟源，接收和发送数据为查询或中
断方式

rUFCON0 = 0;                          //

rUMCON0 = 0;                          //

rUBRDIV0 = 26;                        //设置波特率，PCLK 为 50MHz，波特率为 115.2kHz
```

```
while(!(rUTRSTAT0 & 0x2));          //等待并判断发送缓存是否为空
        rUTXH0 = 0xaa;                                  //是空，则发送 0xAA 字节

while(1)
{
                while(!(rUTRSTAT0 & 0x1)); //等待并判断接收缓存是否准备好
ch = rURXH0;                            //接收一个字节数据
                while(!(rUTRSTAT0 & 0x2));          //等待并判断发送缓存是否为空
                rUTXH0 = ch;                            //发送一个字节数据

switch(ch)                  //根据所接收数据的不同，执行不同的程序
                {
        case 0x11:                  //灭 LED
                                rGPBDAT |= 0x1e0;
        break;
                        case 0x22:                  //亮 LED
                rGPBDAT &= 0x1f;
                        break;
        case 0x33:                  //蜂鸣器不响
                                rGPBDAT &= 0x1e0;
                        break;
                        case 0x44:                  //蜂鸣器响
```

```
                rGPBDAT |= 0x1;
            break;
            default:                    //LED 灭，蜂鸣器不响
                rGPBDAT = 0x1e0;
            break;
}
}
}
```

下面是 UART 中断程序，它要比查询复杂一些，因为涉及到了中断处理，并且 UART 发送数据和接收数据是一个中断源。主程序循环体内不执行任何程序，都在 UART 中断程序内执行。当接收到 0x55 字节数据时，亮两个 LED，当接收到其他数据时，发送该字节，并在发送部分执行亮 4 个 LED 程序。

```
#define _ISR_STARTADDRESS 0x33ffff00
#define pISR_UART0              (*(unsigned *)(_ISR_STARTADDRESS+0x90))
#define U32 unsigned int

#define rGPBCON     (*(volatile unsigned *)0x56000010)      //Port B control
#define rGPBDAT     (*(volatile unsigned *)0x56000014)       //Port B data
#define rGPBUP      (*(volatile unsigned *)0x56000018) //Pull-up control B
```

```
#define rGPHCON        (*(volatile unsigned *)0x56000070)          //Port H control

//#define rGPHDAT       (*(volatile unsigned *)0x56000074)         //Port H data

#define rGPHUP         (*(volatile unsigned *)0x56000078)           //Pull-up control H


#define rULCON0         (*(volatile unsigned *)0x50000000)         //UART 0 Line control

#define rUCON0          (*(volatile unsigned *)0x50000004)         //UART 0 Control

#define rUFCON0         (*(volatile unsigned *)0x50000008)         //UART 0 FIFO control

#define rUMCON0         (*(volatile unsigned *)0x5000000c)        //UART 0 Modem control

#define rUTRSTAT0      (*(volatile unsigned *)0x50000010)          //UART 0 Tx/Rx status

#define rUERSTAT0      (*(volatile unsigned *)0x50000014)          //UART 0 Rx error status

#define rUFSTAT0       (*(volatile unsigned *)0x50000018)          //UART 0 FIFO status

#define rUMSTAT0       (*(volatile unsigned *)0x5000001c)          //UART 0 Modem status

#define rUBRDIV0       (*(volatile unsigned *)0x50000028)         //UART 0 Baud rate divisor


//little endian

#define rUTXH0 (*(volatile unsigned char *)0x50000020) //UART 0 Transmission Hold

#define rURXH0 (*(volatile unsigned char *)0x50000024) //UART 0 Receive buffer


#define rSRCPND         (*(volatile unsigned *)0x4a000000)         //Interrupt request status

#define rINTMSK         (*(volatile unsigned *)0x4a000008)          //Interrupt mask control

#define rINTPND         (*(volatile unsigned *)0x4a000010)          //Interrupt request status
```

```
#define rSUBSRCPND    (*(volatile unsigned *)0x4a000018)       //Sub source pending

#define rINTSUBMSK    (*(volatile unsigned *)0x4a00001c)       //Interrupt sub mask


void __irq uartISP(void)

{

        char ch;


        rSUBSRCPND |= 0x3;

        rSRCPND = 0x1<<28;

        rINTPND = 0x1<<28;


        if(rUTRSTAT0 & 1)                    //接收数据处理部分

        {

                ch = rURXH0;                                //接收字节数据

                if(ch==0x55)

                        rGPBDAT = ~0x61;            //亮两个 LED

                else

                        rUTXH0 = ch;                                //发送字节数据

        }

        else                        //发送数据处理部分

        {

                rGPBDAT = ~0x1e1;                    //亮 4 个 LED
```

```
        }
}

void Main(void)
{

        rGPBCON = 0x015551;
rGPBUP    = 0x7ff;
        rGPBDAT = 0x1e0;

rGPHCON = 0x00faaa;
        rGPHUP    = 0x7ff;

rULCON0 = 0x3;
rUCON0 = 0x5;
rUFCON0 = 0;
rUMCON0 = 0;
rUBRDIV0 = 26;

rSRCPND = 0x1<<28;
        rSUBSRCPND = 0x3;
rINTPND = 0x1<<28;
```

```
        rINTSUBMSK = ~(0x3);                    //打开 UART0 发送和接收中断屏蔽

        rINTMSK = ~(0x1<<28);                   //打开 UART0 中断屏蔽


        pISR_UART0 = (U32)uartISP;


while(1)

    {


}

}
```

最后还要强调几点关于非 FIFO 模式下 UART 中断的一些注意事项：

1. 对于 s3c2440 来说，接收数据是被动的，发送数据是主动的，因此一般来说，接收数据用中断方式，发送数据用查询方式较好；

2. 在中断方式下，当接收到数据时，尽管可能该数据无用，但也一定要读取它，否则下次再接收数据时，不会再引起中断，因为接收数据缓存器被上次接收到的数据所霸占，只要没有读取它，它就永远在那里；
3. 由于 UART 中断涉及到 SUBSRCPND 寄存器，因此在中断处理程序中不仅要清 SRCPND 寄存器，还要清 SUBSRCPND 寄存器，它们的顺序一定是先清 SUBSRCPND 寄存器，再清 SRCPND 寄存器，否则就会引起一个中断两次响应的问题。因为是否中断由 SRCPND 寄存器决定，而 SRCPND 寄存器的相关状态位由 SUBSRCPND 寄存器决定，如果先清 SRCPND 寄存器，而还没有清 SUBSRCPND 寄存器的话，SRCPND 寄存器的相关位还是会被置 1，而一旦被置 1，则一定还会引起中断。