输出：NONE

```
void IIC_MasterWrP(u8 cSlaveAddr,u8 * pData)
{
    u32 uTmp0;
    u32 uTmp1;
    u8 cCnt;
    s32 sDcnt = 100;
    u32 uPT = 0;

    uTmp0 = Inp32(rIICSTAT);
    while(uTmp0&(1<<5))         // 等待，直到IIC总线被释放
        {
            uTmp0 = Inp32(rIICSTAT);
        }
    uTmp1 = Inp32(rIICCON);
    uTmp1 |= (1<<7);
    Outp32(rIICCON,uTmp1);                  // Ack发生使能

    Outp32(rIICDS,cSlaveAddr);

    Outp32(rIICSTAT,0xf0);                  // 主控器发送开始

    while(!(sDcnt == -1))
        {
            if(Inp8(rIICCON)&0x10)
            {
                if((sDcnt--) == 0)
                {
                Outp32(rIICSTAT,0xd0);          //停止主控器发送条件, ACK标志清除
```

```
                uTmp0 = Inp32(rIICCON);
                uTmp0 &= ~(1<<4);               //清除等待位，重新开始
                Outp32(rIICCON,uTmp0);

                Delay(1);                       //等待，直到停止条件生效
                break;
                }
                Outp8(rIICDS,pData[uPT++]);
                for(cCnt=0;cCnt<10;cCnt++);     //用于建立时间  (IICSCL的上升沿)

                uTmp0 = Inp32(rIICCON);
                uTmp0 &= ~(1<<4);               //清除等待位，重新开始
                Outp32(rIICCON,uTmp0);
            }
        }
}
```

2. IIC_MasterRdP函数：功能主要是通过轮询操作进行的主控器接收模式。

输入：cSlaveAddr [8bit SlaveDeviceAddress],

pData [pointer of Data which you want to Rx]

输出：NONE

```
void IIC_MasterRdP(u8 cSlaveAddr,u8 * pData)
{
    u32 uTmp0;
    u32 uTmp1;
    u8 cCnt;

    uTmp0 = Inp32(rIICSTAT);
```

```
while(uTmp0&(1<<5))                    //等待，直到IIC总线被释放
{
        uTmp0 = Inp32(rIICSTAT);
    }
uTmp1 = Inp32(rIICCON);
uTmp1 |= (1<<7);
Outp32(rIICCON,uTmp1);                  // Ack  发生使能

Outp32(rIICDS,cSlaveAddr);

Outp32(rIICSTAT,0xB0);                  //主控器接收开始

while((Inp8(rIICSTAT)&0x1))
{
}
cCnt=0;

while(cCnt<101)
{
    if(Inp8(rIICCON)&0x10)
    {
        pData[cCnt]=Inp8(rIICDS);
        cCnt++;

        uTmp0 = Inp32(rIICCON);
        uTmp0 &= ~(1<<4);               //清除等待位，重新开始
        Outp32(rIICCON,uTmp0);
    }
}
```

```
    Outp8(rIICSTAT,0x90);                    //停止位产生
}
```

3. IIC_SlaveRdP函数：功能主要是通过轮询操作进行的从属器接收模式。

输入：pSlaveAddr [pointer of 8bit SlaveDeviceAddress],

pData [pointer of Data which you want to Rx]

输出：NONE

```
void IIC_SlaveRdP(u8 *pSlaveAddr,u8 *pData)
{
    u32 uTmp0;
    u32 uTmp1;
    u8 cCnt;

//  g_PcIIC_BUFFER =pData;
//  g_uIIC_PT=0;

    uTmp0 = Inp32(rIICSTAT);
    while(uTmp0&(1<<5))                  //等待，直到IIC总线被释放
        {
            uTmp0 = Inp32(rIICSTAT);
        }

    uTmp1 = Inp32(rIICCON);
    uTmp1 |= (1<<7);
    Outp32(rIICCON,uTmp1);               //Ack发生使能


    uTmp0 = Inp32(rIICSTAT);
    uTmp0 &= ~(1<<4);
```

```
    Outp32(rIICSTAT,uTmp0);                 //禁用接收/发送，用于设置从属器地址
    Outp8(rIICADD,*pSlaveAddr);

    Outp32(rIICSTAT,0x10);                  //从属器接收开始

    printf("Wait for Slave Addr\n");
    cCnt=0;

//  while(!((Inp8(rIICSTAT)>>2)&(0x1)));
    while(cCnt<101)
    {
        if(Inp8(rIICCON)&0x10)
        {
//      printf("IICSTAT = %x    ",Inp32(rIICSTAT));

            pData[cCnt]=Inp8(rIICDS);
            cCnt++;

            uTmp0 = Inp32(rIICCON);
            uTmp0 &= ~(1<<4);               //清除等待位，重新开始
            Outp32(rIICCON,uTmp0);
        }
    }
    *pSlaveAddr = pData[0];
    Outp8(rIICSTAT,0x0);
}
```

4. IIC_SlaveWrP函数：功能主要是通过轮询操作进行的从属器发送模式。

输入：pSlaveAddr [pointer of 8bit SlaveDeviceAddress],

```
        pData [pointer of Data which you want to Tx]
    输出：NONE
void IIC_SlaveWrP(u8 *pSlaveAddr,u8 *pData)
{
    u32 uTmp0;
    u32 uTmp1;
    u8 cCnt;
    s32 sDcnt = 100;
    u32 uPT = 0;

//  g_PcIIC_BUFFER =  pData;
//  g_uIIC_PT      =  0;
    uTmp0 = Inp32(rIICSTAT);
    while(uTmp0&(1<<5))                 //等待，直到IIC总线被释放
        {
            uTmp0 = Inp32(rIICSTAT);
        }

    uTmp1 = Inp32(rIICCON);
    uTmp1 |= (1<<7);
    Outp32(rIICCON,uTmp1);              //Ack发生使能

    uTmp0 = Inp32(rIICSTAT);
    uTmp0 &=~(1<<4);
    Outp32(rIICSTAT,uTmp0);             //禁用接收/发送，用于设置从属器地址

    Outp8(rIICADD,*pSlaveAddr);

    Outp32(rIICSTAT,0x50);              //从属器发送开始
```

```
//      while(!((Inp8(rIICSTAT)>>2)&(0x1)));
        while(!(sDcnt == -1))
            {
                if(Inp8(rIICCON)&0x10)
                {
                // printf("IICSTAT = %x ",Inp32(rIICSTAT));
                    if((sDcnt--) == 0)
                    {
                    Outp32(rIICSTAT,0xd0);              //停止从属器发送条件, ACK标志清除

                    uTmp0 = Inp32(rIICCON);
                    uTmp0 &= ~(1<<4);                   //清除等待位，重新开始
                    Outp32(rIICCON,uTmp0);

                    Delay(1);                           //等待，直到停止条件生效
                    break;
                    }
                    Outp8(rIICDS,pData[uPT++]);
                    for(cCnt=0;cCnt<10;cCnt++);         //用于建立时间  (IICSCL上升沿)

                    uTmp0 = Inp32(rIICCON);
                    uTmp0 &= ~(1<<4);                   //清除等待位，重新开始
                    Outp32(rIICCON,uTmp0);
                }
            }
}
```

31 UART 接口

这一节介绍 S3C6410 RSIC 微处理器上的通用异步接收/发送器（UART）串行端口。

该 S3C6410 通用异步接收和发送器（ UART）提供了四个独立的异步串行 I / O （SIO）端口。每个异步串行 I/O（SIO）端口通过中断或者直接存储器存取（DMA）模式来操作。换句话说，UART 是通过产生一个中断或 DMA 请求,在 CPU 和 UART 之间传输数据的。该 UART 使用系统时钟的时间可以支持的比特率最高为 115.2kb/s。如果一外部设备提供 ext_uclk0 或 ext_uclk1 ，则 UART 可以以更高的速度运行。每个 UART 的通道包含了两个 64 字节收发 FIFO 存储器。

该 S3C6410 的 UART 包括可编程波特率，红外线（ IR ）的传送/接收，一个或两个停止位插入，5 位，6 位，7 位或 8 位数据的宽度和奇偶校验。

31.1 UART 接口特性

UART 的特性包括：

（1）基于 rxd0 ， txd0 ， rxd1 ， txd1 ， rxd2 ， txd2 ， rxd3 和 txd3 的 DMA 或中断来操作。

（2）UART 通道 0、1、2 符合 IrDA 1.0 要求，且具有 16 字节的 FIFO。

（3）UART 通道 0、1 具有 nRTS0, nCTS0, nRTS1 和 nCTS1。

（4）支持收发时握手模式。

每个 UART 包含一个波特率发生器，发送器，接收器和控制单元。该波特率发生器由 pclk ， ext_uclk0 或 ext_uclk1 进行时钟控制 。发射器和接收器包含 64 字节的 FIFO 存储器和数据移位寄存器。发送数据之前，首先将数据写入 FIFO 存储器 ，然后复制到发送移位寄存器。通过发送数据的引脚（txdn）将数据发送，同时，通过数据接收的引脚（rxdn）将接收到的数据从接收移位寄存器复制到 FIFO 存储器。

UART 的结构框图，如图 31-1 所示。

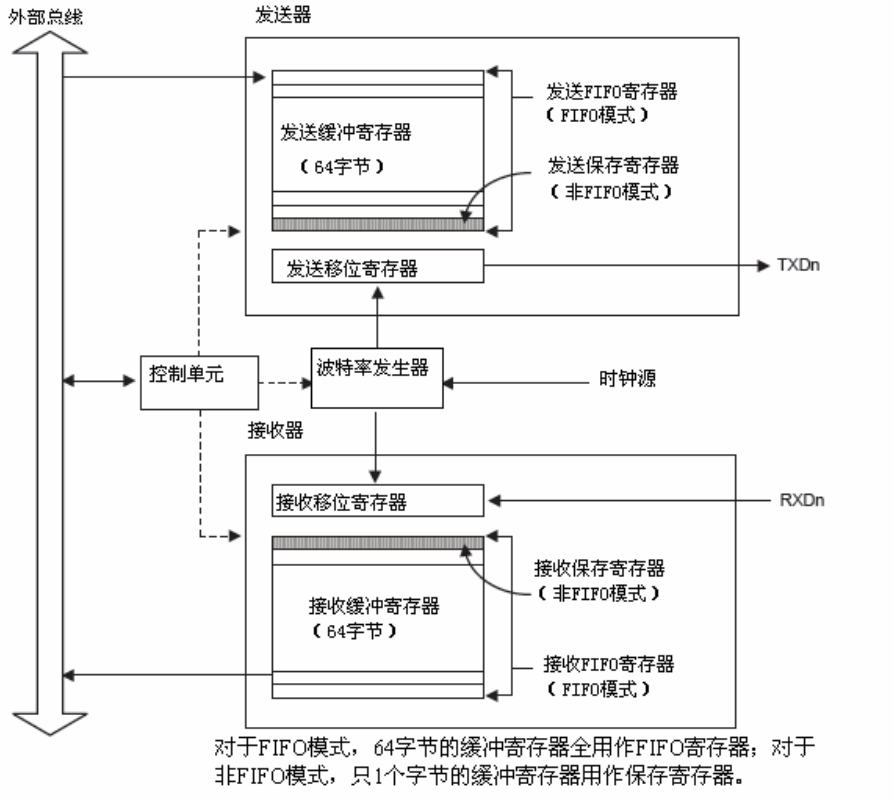

图 31-1 UART 结构框图

31.2 UART 的操作

下面介绍 UART 的操作，包括数据传输，数据接收，中断产生，波特率产生，环回模式，红外线模式和自动流量控制。

1. 数据发送

数据帧发送是可编程的。它由一个起始位，5～8 个数据位，一个可选的奇偶位和 1～2 个可由行控制寄存器（ULCONn ）指定的停止位组成。发送器也可以产生中断条件，在传输过程中，它通过置位逻辑状态 0 来强制串行输出。当目前的发送完全传输完成后，发送中断信号。然后不断传送数据到发送 FIFO 寄存器（在非 FIFO 的模式下，发送保存寄存器）。

2.数据接收

和数据发送一样，数据帧接收也是可编程的。它是由一个起始位，5～8 个数据位，一个可选的奇偶位和行控制寄存器指定的 1～2 个停止位组成。接收器可以检测到溢出错误、奇偶错误、帧错误和中断条件，并为它们设置错误标志。

溢出错误说明在数据被读取之前，新的数据已经将原有的数据覆盖。

奇偶错误说明接收器已经检测到一个意外的奇偶条件。

帧错误表示收到的数据没有有效的停止位。

中断条件表明接收过程中置位逻辑状态 0 的时间比发送一帧的时间长。

当三个字的时间（间隔由设置的字长决定）间隔内没有接收任何数据，并且 FIFO 模式下接收 FIFO 寄存器不为空，接收超时条件发生。

3.自动流量控制（AFC）

该 S3C6410 的 UART0 和 UART1 通过 nRTS 和 nCTS 信号支持自动流量控制。某种情况下，它可以连接到外部 UART。如果想要将 UART 和一台调制解调器连接，必须在 UMCONn 寄存器中禁用自动流量控制位，并且通过软件控制信号 nRTS。

在自动流量控制过程中，nRTS 依靠接收器的条件，nCTS 信号控制发送器的运作。只有当 nCTS 信号被激活（在 AFC 中，nCTS 意味着另一个 UART 的 FIFO 寄存器准备接收数据），UART 的发送器发送 FIFO 寄存器中的数据。 在 UART 接收数据之前，如果它接收 FIFO 寄存器超过两个字节以上的空间，nRTS 则被激活；如果它的接收 FIFO 寄存器只有不足一个字节的空间，nRTS 则停止活动（在 AFC 中，nRTS 意味着它本身的接收 FIFO 寄存器已经真被接收数据）。如图 31-2 所示显示了 UART AFC 接口的发送和接受状态图。

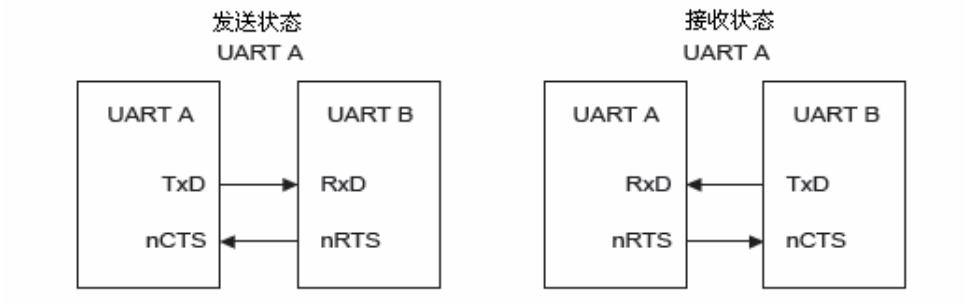

图 31-2 UART AFC 接口

非自动流量控制的例子。（通过软件控制 NRTS 和 NCTS）