ADSP-BF54x Blackfin®处理器增强的UART

撰稿人 Benno Kusstatscher

Rev 1 – November 6, 2007

简介

与 ADSP-BF52x， ADSP-BF53x 和 ADSP-BF561 Blackfin® 相比，ADSP- BF54x 系列Blackfin处理器对UART模块引入了新的特性。本EE文件总结了UART模块的改进，并说明了其优点和代码移植过程中需要的帮助。

这些新的特性包括：

- 自动RTS/CTS硬件流控制
- 增加的接收FIFO
- 更好的比特率粒度
- 可编程模块更适合Blackfin架构
- 改进的中断处理
- 具有4个UART（两个有流控制）

本EE文件简要的涵盖了上述主题，并假定读者已熟悉了ADSP- BF53x UART模块。关于ADSP-BF54x UART的完整描述请参阅*ADSP-BF54x Blackfin处理器硬件说明* [1]。

可编程模块

ADSP-BF53x处理器的UART与工业标准完全一致，并与16450可编程模块兼容。除了明显的代码兼容优点外，该设计也有一个限制：一些过于陈旧的可编程模块不能充分支持Blackfin芯片的流水操作和优化的吞吐率架构。

破坏性读操作更少

UARTx_LSR和UARTx_IIR寄存器在读操作时具有破坏性行为，这一特殊性质要求特别注意推测读操作的条件。但是，将每个模块的接收、发送和状态处理完全区分开也几乎是不可能。

原本状态位的破坏性读操作现在被转换为写1再清除操作（W1C）。比如：下列的UARTx_LSR读顺序就隐含清除了帧错误（FE）状态位。

```
if (*pUART0_LSR & THRE) { ... }
```

现在，FE位要求一个明确的W1C清除操作：

```
*pUART0_LSR = FE;
```

UARTx_IIR寄存器更少

UARTx_IIR寄存器是过时的。由于其破坏性读操作特性和繁杂的优先级控制方案，UARTx_IIR寄存器已经逐渐被忽视。

中断服务程序可以通过轮询UARTx_LSR和UARTx_MSR状态寄存器来判断调用的中断源。

例如，如果某个例程选择检测UARTx_IIR来判别是否出现了行错误，

```
if (*pUART0_IIR == 0x6) { ... }
```

现在就应该检测UARTx_LSR状态寄存器：

```
if (*pUART0_LSR & (OE|FE|PE|BI))
{
  *pUART0_LSR = OE|FE|PE|BI;
  ...
}
```

通过运用中断分配寄存器（SIC_IARx），可以实现全局中断优先级控制。

UARTx_IER设置和清除寄存器组

通常，需要暂时禁止UART接收、发送或状态中断。在ADSP-BF53x处理器上实现该功能需要对中断使能（UARTx_IER）寄存器进行一个“复杂的”读-修改-写操作。

如果有三个不同的中断服务程序为三个中断服务—特别是使能了中断嵌套—要确保发送中断处理中的读-修改-写操作不被接收服务程序中断，这需要特别当心。多数情况下，除了禁止全局中断外，没有 “更简便”的应变措施了。

在新的UART中，UARTx_IER寄存器由一个寄存器组（UARTx_IER_ SET和UARTx_IER_CLEAR）代替。对UARTx_IER_SET寄存器中写入‘1’就使能中断，在UARTx_IER_ CLEAR寄存器中写入‘1’就可以禁止各自的中断，写入‘0’不会有任何效果。任何对寄存器的读操作将会返回使能位。

这种新的设计允许在不影响其他位的情况下分别实现中断处理、独立转换相关的中断使能位。

复位后，中断使能寄存器默认设置为0，但是可以通过下面的代码修改初始化设置指令

```
*pUART0_IER = ETBEI|ERBFI;
```

如果在复位后执行上述代码还可简化：

```
*pUART0_IER_SET = ETBEI|ERBFI;
```

如果过去状态未知，那么执行下面的代码：

```
*pUART0_IER_CLEAR = ~(ETBEI|ERBFI);
*pUART0_IER_SET = ETBEI|ERBFI;
```

然而，如果在运行时需要转换某个特定的位，以前的代码（中断嵌套使能时）

```
short tmp;
/* temporarily enable */
tmp = cli();
*pUART0 IER |= ETBEI;
sti(tmp);
...
/* temporarily disable */
tmp = cli();
*pUART0 IER &= ~ETBEI;
sti(tmp);
```

可简化为：

```
/* temporarily enable */
*pUART0 IER SET = ETBEI;
...
/* temporarily disable */
*pUART0_IER_CLEAR = ETBEI;
```

MMR地址共享更少

ADSP-BF53x处理器内复用存储器映射寄存器（MMR）共享相同的地址。UARTx_LCR寄存器中的DLAB位的功能就是附加地址位，用于区分是访问除数锁存寄存器（UARTx_DLH和UARTx_DLL）还是数据寄存器（UARTx_RBR和UARTx_THR）。

现在放弃使用DLAB位，UARTx_DLH和UARTx_DLL有专用的MMR地址。

```
/* *pUART0_LCR = DLAB; */
*pUART0_DLL = divider & 0xFF;
```

```
*pUART0_DLH = (divider>>8) & 0xFF;
*pUART0_LCR = <new settings>;
```

因此，UARTx_DLH和UARTx_ DLL寄存器可以直接访问，并且可以直接删除上述代码的第一行。

中断强化

状态中断

除了正规的发送和接收中断请求通道外，每个UART还具有输出状态中断特性。ADSP-BF53x处理器的状态中断仅作为UART线路错误的信令。而ADSP-BF54x处理器状态中断还表示以下意义：

- 线路错误
- 可选的RX和TX信令
- 发送完成
- 调制解调器状态
- FIFO门限值

可选的RX和TX信令

由于发送和接收中断都是通过DMA控制路由，即使不使用DMA通道，也要求分配DMA通道。被禁止的DMA通道仍可向系统中断控制器（SIC）发送普通DMA中断请求。

除DMA通道外，ADSP-BF54x处理器还有多个具有中断能力的外设。特别之义是默认情况下UART2和UART3没有与之关联的DMA通道。

为了避免快速转发中断而大量使用DMA通道，ADSP-BF54x的UART可以选择重新将直接连接到SIC控制器的发送和接收事件定义到状态中断上。当全局控制寄存器中的EGLSI位置位，发送和接收事件就不是发到普通发送和接收中断通道的信号，而是发到与普通状态请求相与后的状态中断通道的信号。

通过这种方式，UART模块在不与相关DMA通道相连的非DMA模式下依旧可以实现全部功能。但是，这里有一个限制，即所有UART事件必须通过公共的中断服务程序进行服务。

发送完成中断

发送端中断时序也是有意义的讨论话题。正常情况下，当UARTx_THR寄存器新数据准备好时，UART发送器就发送一个中断或者DMA请求，作为THRE位的信号量。此时，发送移位寄存器TSR中的发送数据可能会被挂起。

因此，不允许发送中断服务程序禁止UART，处理器必须等待直到传送器空闲位（TEMT）变高。

ADSP-BF54x的UART，TEMT位还承担了新的中断功能。在UARTx_LSR寄存器中有一个新的可自我清除的位，是TEMT位的副本，称为发送完成指示位（TFI）。如果由ETFI位使能TFI，就会触发状态中断通道的请求。服务子程序必须对TFI位W1C。此时，就可以安全禁止UART和EIA-485格式线路驱动。

作为备选，通过DMA控制器也可以为发送完成事件提供信号量，这（通过使能EDTPTI位即可实现。当工作单元的最后一个数据字转移到UART时，DMA控制器就通知UART，在TEMT位变高的下一个时刻，将一个特殊命令送到DMA通道。当DMAEN也使能时，DMA通道就将该命令转成中断请求，并将其发送到SIC控制器。

如果DMA工作在停止模式，UART的EDTPTI位将被置位，而DI_EN位不会被置位，在最后一个数据字节离开UART之前，正常的DMA中断请求会被延迟。如果EDTPTI位和DI_EN位都被置位，则在同样的通道上将产生两个中断请求。

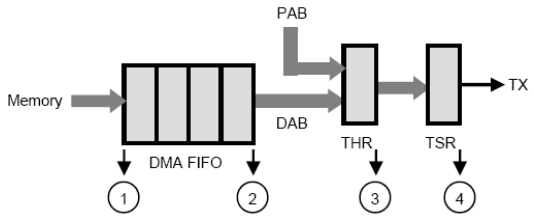

图1 发送中断选项

图1说明了UART发送操作中四个中断选项的效果。

1. 正规TX DMA中断。此时可初始化新的工作单元，但是由于数据仍然在DMA FIFO中，因此不能禁止DMA。
2. SYNC位被置位的TX DMA中断。DMA中断会延迟到所有数据都移出DMA FIFO后，此时可以安全地禁止和重新分配DMA通道。
3. 正规TX DMA中断。在DMA模式下，该请求的功能为DMA请求，无法产生中断。在非DMA模式下，DMA通道将该传送请求前向传输到SIC控制器。当EGL位置位时，该请求将会进入状态中断通道。
4. 发送完成中断。此时，所有数据都移出了UART，因此可以安全地禁止UART模块，如果使能了ETFI位，则还可产生状态中断。如果使能DMA且EDTPTI位被置位，则DMA控制器通过TX DMA通道请求中断。用户通过发送或状态中断服务程序可为TFI事件服务。

接收FIFO

ADSP-BF53x UART以一个二级接收缓冲器为特征，该缓冲器由接收缓冲寄存器（UARTx_RBR）和串行采样寄存器（RSR）构成，软件无法访问RSR寄存器。

如果使能，当数据字从RSR复制到UARTx_RBR时可发布接收中断，且在第一个停止位被采样时执行。如果软件（或者DMA）在接收第二个数据字停止位前没有读UARTx_RBR寄存器，则第二个字会覆盖UART_RBR中的第一个字，并且报告出现溢出情况。为了避免溢出错误，意味着软件必须在一个数据帧内要响应。

ADSP-BF54x处理器RSR寄存器和UARTx_RBR寄存器之间插入了一个附加的深度为4的FIFO，如图2所示。与ADSP-BF53x的实现相比，中断响应时间的要求放宽到5倍，该特性不需要对软件作任何修改就可实现。

在新引入的UARTx_MSR状态寄存器中有一个状态位，称为RFCS位，它是接收FIFO计数状态的信号量。RFCS位的行为取决于UARTx_MCR寄存器中新定义的接收FIFO门限值位（RFIT）。

如果RFIT=0，RFCS位就表示接收FIFO中有两个或更多的数据字有效。如果RFIT=1，则RFCS位标志接收FIFO中有四个或更多的数据字有效。由于处理器核或DMA对UARTx_RBR进行读操作，当FIFO中的数据少于门限值时，RFCS位将自动清零。

基于RFCS位，使能接收FIFO计数中断位（ERTCI）可使能中断，并作为UART状态中断通道的信号量。在非DMA模式，中断服务程序可能依赖状态中断而非普通的接收中断，这样可将中断负载减为原来的一半或者四分之一。可以从接收缓冲器中读取两个（RFIT=0）或者四个（RFIT=1）字节。当RFCS为清零时，也就取消了中断请求。

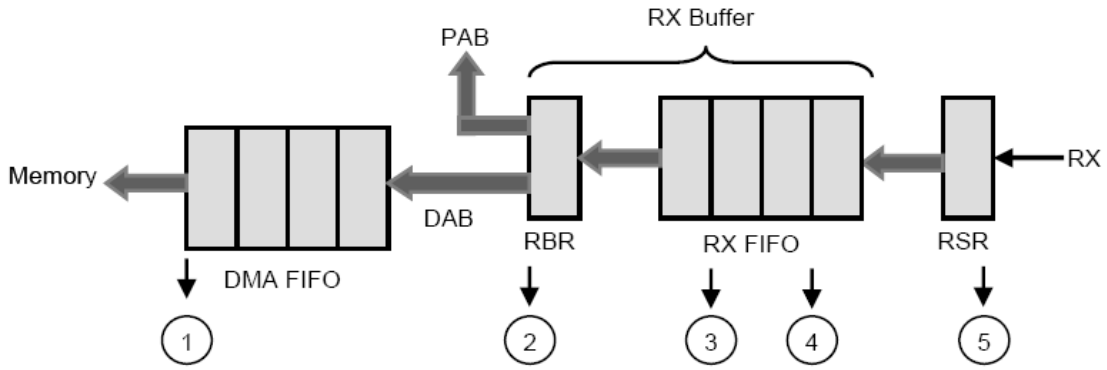

图2 接收中断选项

图2说明了UART接收操作中五个中断选项的效果。

1. 正规RX DMA中断。此时已完成所有接收操作，除非有更多的数据在接收流水中，否则可以安全地禁止DMA通道或UART控制器。
2. 当ERBFI位置位时，在非DMA模式可产生正常中断。此时会报告所有的接收状态标志位（DR，PE，FE和BI）。因此，ELSI中断在这里也是部分对齐的。
3. FIFO门限值中断选项1。当RFIT=0时，ERFCI中断表示在UART接收缓冲器中已有两字节的数据。
4. FIFO门限值中断选项2。当RFIT=1时，ERFCI中断表示在UART接收缓冲器中已有四字节的数据。
5. 溢出错误。在接收缓冲器清空之前接收第六个字停止位时，会产生溢出错误，并且报告给ELSI中断。

硬件流控制

硬件流控制是一个公用握手协议，用于使能UART接收器来防止当接收缓冲器溢出时，对应的发送器依旧发送数据。该特征对UART0和UART2无效。

接收器产生称为请求发送的$\overline{\text{RTS}}$输出信号，当接收缓冲器填满并超过门限时，将使该信号无效。发送器接收称为清除发送的$\overline{\text{CTS}}$输入信号，当$\overline{\text{CTS}}$无效时就停止发送。当然，仍然会完成当前处理字的发送。

对于双向数据流，硬件连接如图3所示。

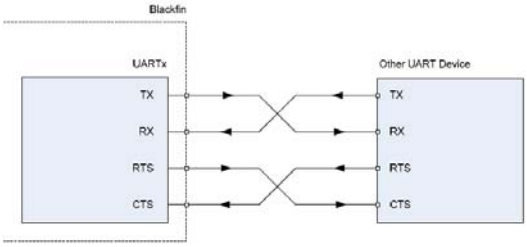

图3 UART硬件流控制连接

在工业设备中，$\overline{\text{RTS}}$和$\overline{\text{CTS}}$信号通常低电平有效（低电平时发送）。通过设定UARTx_MCR寄存器中的FCPOL位可改变信号极性，使$\overline{\text{RTS}}$和$\overline{\text{CTS}}$变为高电平有效（高电平时发送）。

接收器$\overline{\text{RTS}}$产生

UARTx_MCR寄存器中的ARTS位可使能自动产生$\overline{\text{RTS}}$信号。如果ARTS=0，则$\overline{\text{RTS}}$输出信号的状态由MRTS位控制。若ARTS=1，$\overline{\text{RTS}}$输出说明接收缓冲器的状态。

与控制中断时序的RFIT位类似，接收FIFO RTS门限值位（RFRT）控制$\overline{\text{RTS}}$信号的有效性。与中断不同，$\overline{\text{RTS}}$信号的有效和无效都有滞后现象。

当接收缓冲器中有两个（RFRT=0）或者四个（RFRT=1）字节，且检测到第三或者第五个开始位时，则$\overline{\text{RTS}}$信号无效。由于UARTx_RBR的读操作，当接收缓冲区低于两个（RFRT=0）或四个（RFRT=1）字节的门限时，则$\overline{\text{RTS}}$将再次有效。

由于在端口复用级也必须使能$\overline{\text{RTS}}$信号，给该信号加上拉电阻（当FCPOL=1时上拉）有利于防止处理器复位周期中的浮动。

同样，TX输出端的上拉电阻也可以避免处理器复位周期中的浮动。

发送器监测$\overline{\text{CTS}}$

在UARTx_MCR寄存器中有个新的称为ACTS的位，当置位时，使能自动$\overline{\text{CTS}}$监测。一旦发送器检测到$\overline{\text{CTS}}$输入引脚的电平被撤销，就完成当前TSR移位寄存器中的字发送。然而，它会阻止UARTx_THR寄存器中当前的字发送给TSR寄存器，直到$\overline{\text{CTS}}$再次有效为止。

另外，在UARTx_MCR中还有一个新的称为CTS的位，表示$\overline{\text{CTS}}$输入信号是否有效。此外，还有一个与CTS位相关联的SCTS位，W1C软件操作可对该位清零。

如果使能了EDDSI位，无论SDTS是否为1，都会产生所谓的调制解调器状态中断，并作为UART状态中断的信号量。

与$\overline{\text{CTS}}$输入引脚相似，软件可以暂停发送。该过程可以通过设定UARTx_MCR寄存器中的XOFF位而立即实现。

在自回环模式（LOOP_ENA=1），接收器的$\overline{\text{RTS}}$输出直接由内部回送到发送器的$\overline{\text{CTS}}$输出。

比特率产生

UART模块的基准时钟由系统时钟（SCLK）得到，通过UARTx_DLL和UARTx_DLH 8位寄存器表示的16位整数除系统时钟得到。

为了安全，接收的UART数据以16倍的过采样率采样，因此，有效位速率比基时钟慢16倍。

如果UART以高比特率运行，则可支持的比特率粒度就十分有限。比如，设SCLK的频率为120MHz，额定比特率为1Mb/s，除数的值为7，则比特率即为1071Mb/s，粒度为7%，如果除数值为8，则比特率为937500b/s，粒度也没变优多少。

基于以上原因，ADSP-BF54x UART在全局控制寄存器中指定了新的允许除以一（EDBO）位。如果该位置位，则旁路掉16位的除法电路。因此，有效的比特率即为

$$bitrate = \frac{SCLK}{16^{1-EDBO} \times (DLH : DLL)}$$

当EDBO位置位时，接收器仍然过采样，且在三个样本中选一个做主要判定。但是，比特中间的采样点不再由传统的异步逻辑产生。

在上例中，除数值为120时，如果EDBO置位，则可产生期望的比特率。

自动波特检测

与ADSP-BF53x UART相比，自动波特检测的原理没有变。ADSP-BF54x处理器四个UART都有与之相关的GP计时器，其交替的捕获输入（TACIx）监测UART接收输入。

- TACI0监测UART0 RX
- TACI1监测UART1 RX
- TACI2监测UART2 RX
- TACI3监测UART3 RX

如果使用推荐的自动波特模式0x40，且计时器配置为捕获两个下降沿之间的周期，则TIMERx_PERIOD寄存器包含了图4中所述的8位时间值。

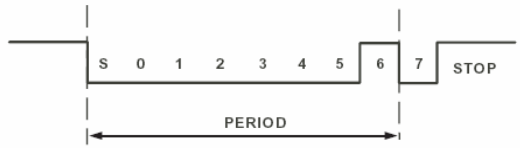

图4 使用0x40模板的自动波特检测

UART时钟除数的值仍然由TIMERx_PERIOD寄存器中的值向右移动7位计算得到。但如果EDBO位置位，则TIMERx_PERIOD寄存器中的值只移动3位。

代码移植校验表

将ADSP-BF53x工程的代码移植到ADSP-BF54x处理器通常很快，然而，必须更新一些代码行。

需要做的改变

- 去掉DLAV指令，直接访问UARTx_DLL和UARTx_DLH。
- 用W1S和W1C操作取代繁琐的UART_IER读-修改-写操作，实现对UARTx_IER_SET寄存器和UART_IER_CLEAR寄存器操作。
- 查询UARTx_LSR和/或UARTx_MSR，而不是UARTx_IIR。

推荐的修改

- 由TFI中断取代繁琐的TEMT轮流监测循环。
- 非DMA模式中所有三个UART中断通道都由相同的中断服务子程序服务，设置EGLSI位释放DMA资源。
- 在非DMA模式，为了降低中断负载，为ERFCI中断配置用户接收中断服务程序，而不是ERBFI中断。
- 如果用软件模拟流控制，应允许取代自动硬件流控制。
- 如果UART时钟除法器提供的比特率偏离额定比特率，为了提供更好的传输质量，应检查是否设置了EDBO位。

参考文献

[1] *ADSP-BF54x Blackfin Processor Peripheral Hardware Reference*. Rev 0.3. October 2007. Analog Device, Inc.

文档记录

Revision	Description
Rev 1 – November 6, 2007 *by B. Kusstatscher*	Initial Release