更多关于 ADI 公司的 DSP、处理器以及开发工具的技术资料，
请访问网站：http://www.analog.com/ee-note 和 http://www.analog.com/processor
如需技术支持，请发邮件至 processor.support@analog.com 或 processor.tools.support@analog.com

SHARC®处理器和 Blackfin®处理器的 SPI 连接

作者：Jeyanthi Jegadeesan Andreas Pellkofer

Rev 1 – July 8, 2008

引言

该应用笔记介绍如何将 ADI 公司的 Blackfin®处理器与 SHARC®处理器通过串行外设接口(SPI)连接起来。

内容包括：

- SPI 描述
- 物理层设置
- SPI 的配置
- Blackfin 处理器的 SPI 接口编程模式
- SHARC 处理器的 SPI 接口编程模式
- 代码示例

目的

现今的嵌入式系统常常要求有多个处理器，每个处理器都有其特定的应用范围。音频领域的应用是一种同时采用了 ADI 公司的 Blackfin 处理器和 SHARC 处理器的典型系统。

SHARC 处理器的典型应用包括高动态范围，高性能，以及浮点音频处理。Blackfin 处理器集成一个 DSP 和微型控制器，可作为主机来控制 SHARC 处理器，比如：

- 引导 SHARC 处理器(SPI 从引导)
- 发送信息(如：低音，高音，音量，衰减)

图 1 显示了一个多处理器系统例子的信号流程图。

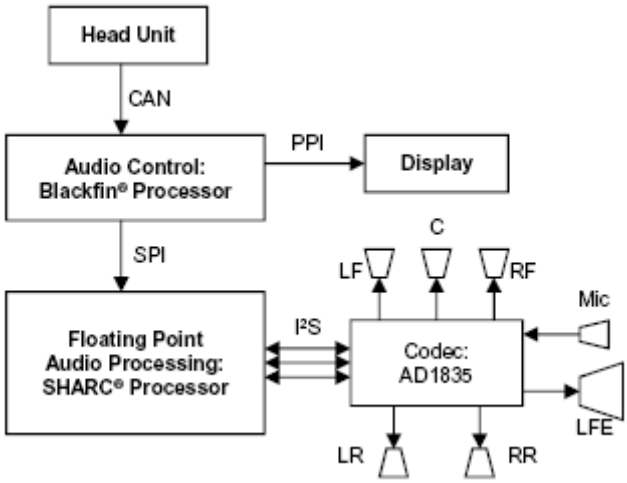

图 1. 系统设置

SPI 描述

SPI®全称为同步串行数据中继器，是由摩托罗拉公司命名的一种工业标准。这一标准还没有完整的规范，只包括硬件，而没有软件协议。

它支持与多种 SPI 兼容设备之间的通讯。与 I^2C 等其他串行接口不同，SPI 外设是一个四线接口——I^2C 用两条线——由两个数据引脚(MOSI 和 MISO)；一个设备选择引脚(/SPISS)和一个门控时钟引脚(SCK)组成。这两个数据引脚允许对其他 SPI 兼容设备进行全双工通信操作。SPI 还包括可编程的波特率，时钟相位和时钟极性。设备

之间以主/从模式进行通信，由主设备首先发送数据帧，并允许有多个从动设备，因为每个从芯片有单独的从/芯片选择信号线。

美国国家半导体公司的 Microwire™(μWire)是 SPI 的一个有限子集，它以 SPI 为基础并与之兼容。该应用笔记中所提到的 *SPI* 指的是摩托罗拉公司的 SPI。

典型的 SPI 兼容外围设备包括：

- 微处理器
- 编解码器
- A/D 和 D/A 转换器
- 传感器
- 闪存设备
- SP/DIF 和 AES/EBU 数字音频发射机和接收机
- LCD 显示

Blackfin 处理器和 SHARC 处理器上的 SPI 接口具有以下特性：

- 全双工同步串行接口
- 8 位或 16 位字长(Blackfin 处理器)
- 32 位字长(SHARC 处理器)
- 低位在前或者高位在前的数据格式
- 可编程波特率，时钟极性和相位
- 主控，从动及多主控模式
- 漏极开路输出以避免由于数据冲突引起的可能的驱动冲突，从而支持多主机环境。
- 来自一个 SPI 设备的主或从加载
- 允许无需处理器内核辅助操作而进行数据传送的 DMA 性能

图 2 和图 3 示出了 SPI 的结构图。

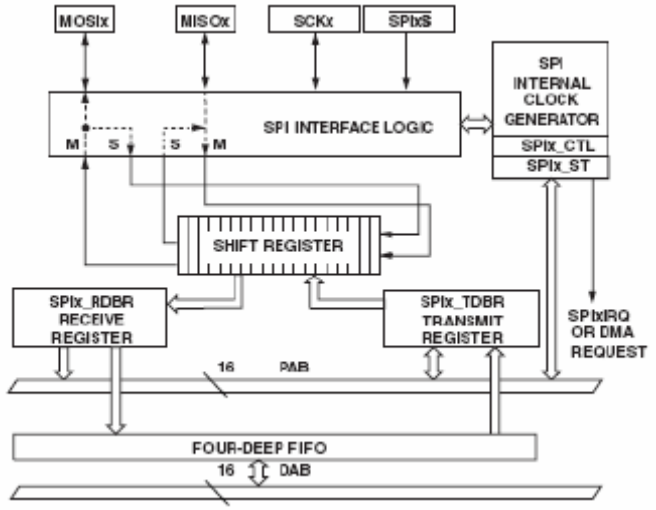

图 2 Blackfin SPI 结构图

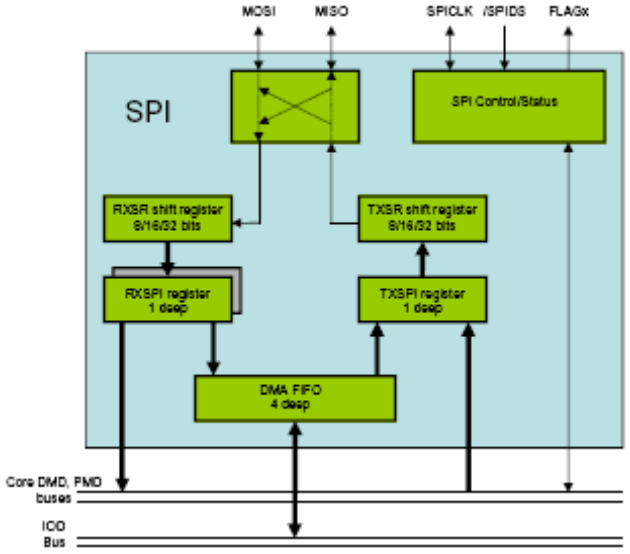

图 3 SHARC SPI 结构图

如需详细资料，请分别查阅 Blackfin 处理器系列和 SHARC 处理器系列的*硬件参考*手册(HRM)中的 SPI 部分。

设置物理层

图 4 描述的是一个 SHARC SPI 从设备和一个 Blackfin SPI 主控设备的信号连接与传输方向。

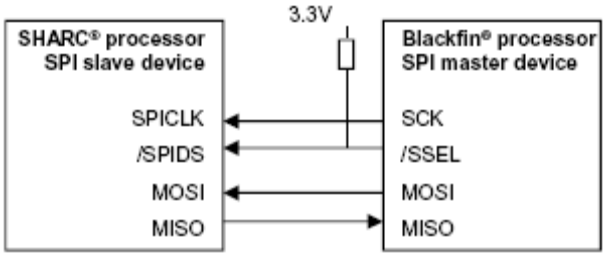

图4 主——从连接

一个 SPI 主设备常常同时驱动时钟和从设备选择信号。一个 SPI 从设备只有在时钟信号和从选择信号两者都存在时才起作用。

两个处理器上的 SPI 主控设备和 SPI 从设备之间的区别仅是 SPI 控制寄存器中的一个二进制位(MSTR)。

对于一个主控 SPI 设备，在一个多主机环境中，/SPISS 引脚只在错误信号输入时才起作用。推荐接一个上拉电阻。

SPI 从选择输出使能信号(/SSEL)在 SPI 协议中的常态为低电平。鉴于各个引脚在重启过程中没有被驱动，所以推荐一个电阻上拉。

Blackfin处理器需要在SCK引脚上加一个上拉电阻。SHARC处理器的DPI提供一个内部上拉电阻给SPICLK引脚，因此这里不需要外部上拉电阻。

Blackfin 处理器的编程模式

Blackfin 处理器的 SPI 接口的编程模式在 Blackfin 系列*硬件参考*手册中有所描述。在 ADSP-BF537 HRM 中所提供的 SPI 流程图为 SPI 传输的设置提供了一个很好的参考。

该部分对编程模式的细节进行了描述，并作为附录和对现有文献的简要总结。

这篇应用笔记针对一个单一主控/单一从动的环境，该环境中，ADSP-BF537 处理器作为SPI主控器，而ADSP-21369 处理器则作为一个SPI从设备。文件提供的例子中还包括基于这两个处理器的一些其他设置。

准备工作

大多数Blackfin处理器利用引脚复用来减少引脚数量。在 ADSP-BF537 处理器上，大部分 SPI 信号都通过端口 F 进入。最重要的 5 个信号(SCK，MISO，MOSI，/SPISS 和 SPISSEL1)与其他外设信号不复用。

对于 ADSP-BF537 处理器，必须写入 PORTF_FER 来使能 SPI 信号。通过默认设置，GPIO 功能使能。

要使用 SPISSEL2 – SPISSEL7，请参考 HRM 中的 SPI 部分和通用端口部分内容。如果有必要，这些信号也应使能。

中断用来标志来自/到达一个外设的一次传输的结束。否则处理器就必须周期性的查询状态位来检测传输是否结束。如果处理核和系统/外设之间的时钟比率较高(比如：SPI 时钟为 1MHz，而处理核时钟为 500MHz)，查询结果可能就不准确，这将导致其他操作的同时执行。

内核事件控制器和系统中断控制器(SIC)都必须进行正确配置。详细资料请查阅 *Blackfin 处理器编程参考*手册(PRM)中的“程序控制器”部分和硬件参考手册中的“系统中断”部分内容。

SPI 初始化

SPI 接口由一组用于系统配置的(MMR)存储器映射的寄存器(SPI_FLG，SPI_BAUD，和 SPI_CTL)组成。

SPI 状态可从 SPI_STAT 寄存器读出。

对于内核驱动的传输，还需要有 SPI 发送/接收数据缓冲寄存器(SPI_TDBR/SPI_RDBR)和 SPI_SHADOW。

SPI_FLG 寄存器

对于一个 SPI 主控设备来说，从设备的选择要通过写入 SPI_FLG，设置适当的从设备选择使能(FLSx)位来完成。

从设备选择位(FLGx)决定驱动到从设备选择信号的值。

如果 CPHA=1(在 SPI_CTL 中设置 CPHA 位)，输出值就由 FLGx 位的软件控制来设置。

如果 CPHA=0，则由 SPI 硬件设置输出值，而 FLGx 位被忽略。这意味着硬件对于每一个信号字都要选择或撤销选择从动设备(/SSEL)(见图 5 和图 7)。

下面的图说明了 SPI 上一个 16 位字长的传输。前两个字带有图解。Blackfin 处理器是主控设备，并驱动 SPI 时钟和从设备选择信号。SHARC 处理器则作为从设备，接收来自主控 SPI 的数据。

此外，图 7 和图 8 还示出了一个传输过程的 SCK 低有效的版本；在这种情况下时钟极性反转。

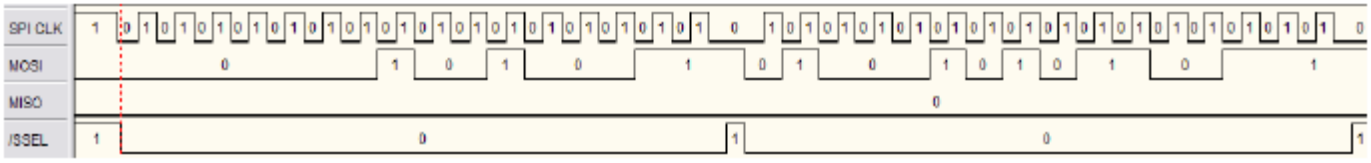

Figure 5. CPHA = 0, active high SCK (CPOL = 0)

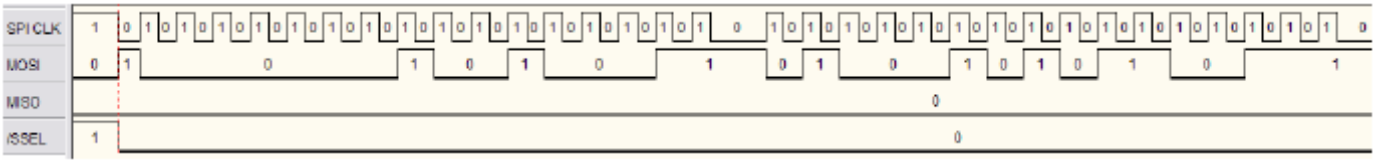

Figure 6. CPHA = 1, active high SCK (CPOL =0)

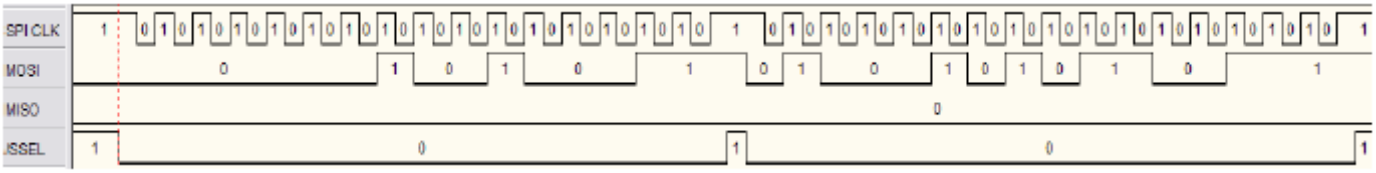

Figure 7. CPHA = 0, active low SCK (CPOL=1)

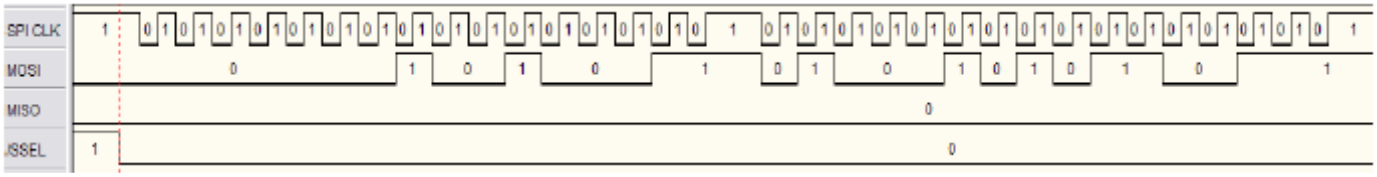

Figure 8. CPHA = 1, active low SCK (CPOL = 1)

SPI_BAUD

SPI_BAUD 寄存器允许您设置 SPI 的时钟频率。这是源于 Blackfin 处理器的系统时钟(SCK)。如果 SCK 被设置为最大值 133MHz，这一频率会被一个 SPI 频率所限制~33MHz(SCK/4)。如果需要更高的频率，Blackfin 处理器的 SPORT 接口可以被用来模拟 SPI。连续的 66MHz 的时钟速率是可以达到的，因为 SPORT 最高的时钟频率为 SCK/2。参考 EE-304[4]。

SPI_CTL

SPI_CTL 寄存器在 HRM 中有所描述。

如在“SPI 描述”中所提到的，摩托罗拉公司没有为 SPI 专门指定一个(定时)协议，典型应用中有四个不同的模式。这些模式由两个位设置：CPLD(时钟极性)和 CPHA(时钟相位)。

目前的 Blackfin 处理器的 SPI 接口，相对于 SHARC 处理器的 SPI 接口来讲，不支持 32 位数据。

SPI_STAT

SPI 状态寄存器(SPI_STAT)可以分为两组数据位。第一组用于检测错误，比如缓存器的欠载运行，缓存器溢出，以及多主控环境中的冲突。另一组数据位显示发送缓存器或接收缓存器的状态以及 SPI 状态(SPI 已完成)。

详细信息请参阅说明如何应用这些状态位的“SPI DMA 传输和中断服务”部分内容。

SPI DMA 初始化

SPI 有一个专用 DMA 通道来完成同一时间内的单独发送或接收操作。

详细信息请参考 HRM 中的“直接内存存取”章节的内容。

SPI 传输启动

对于一个 SPI 主控设备，从设备的选择是通过写 SPI_FLG，设置适当的从设备选择使能(FLSx)位完成的。

在 SPI DMA 传输模式中，SPI 接口和 DMA 不应在初始化过程中使能。如果两者都配置正确，先启动 DMA(DMA=1)，然后再启动 SPI(SPI=1)。

在 SPI 内核模式下，如果在 TIMOD=00 的模式下运行时，就可在初始化过程中使能 SPI。TIMOD=01 时，SPI 位被设置后就立即启动。可以通过先填充 SPI_TDBR 然后使能 SPI，也可用 TIMOD=00 来完成 SPI 传输操作。对于后者，用 SPI_RDBR 的一次假读操作就可以完成传输过程的启动。

SPI DMA 传输和中断服务

	DMA Tx	DMA Rx	Core Tx	Core Rx
DMA_RUN	1x	-	-	
TXS	2x	-	1x	
RXS	-	-	-	≥1x
SPIF	1x	-	1x	1x

表 1 被查询的状态位

对 SPI DMA 操作，必须等到出现 DMA 中断的发生，也就是要等到在位 DMAx_IRQ_STATUS 寄存器中 DMA_DONE 位的置位。

DMA_DONE中断在最后一个存储器访问(读或写)全部完成之前都是不确定的。为了向外围设备传输数据(存储器读)，中断发生时，在通道的DMA FIFO中可能会有多达四个数据字。

SPI 接收或 SPI 发送的服务程序必须首先清除中断源，要向 DMA_DONE 位写入一个“1”(*W1C——写入-1-完成清除*)，否则就会再次出现一个中断。

然而，如果应用程序需要知道最后一个数据项是何时发送至外围设备，应用程序可以测试或查询DMA_RUN位。只要在FIFO中存在未读取数据，DMA_RUN位就为1。

持续询问 DMAx_IRQ_STATUS 寄存器中的 DMA_RUN 位(DMA_RUN=1)，直到它变为低(DMA_RUN=0)。

DMA 应用于 SPI 发送过程时，DMA_DONE中断标志着DMA FIFO为空。但是，这时外围设备的 (SPI) DMA FIFO中仍可能有数据等待发送。因此，软件就需要对SPI_STAT寄存器中的TXS位进行查询，直到两次连续读操作的结果都为低，此时SPI DMA FIFO将为空。

当发送缓存器存满时 TXS 位置位，但是数据还可能停留在 SPI 移位寄存器中。

在上一个 SCK 边沿后的半个 SCK 周期时刻 SPIF 位置位，也就是上一位字被移出的时刻，这时 SPI 传输已完成。

RXS 位标志接收缓存器可以读取。当单个数据字末端开始传输时，RXS 位置位，标志着一个新的数据字刚刚被接收并被存入接收缓存器 SPI_RDBR。对于主控 SPI，RXS 在最后一个 SCK 采样边沿后很快置位。对于从 SPI，RXS 在最后一个 SCK 边缘后很快置位，而忽略了 CPHA 或 CPOL。等待时间通常为几个 SCLK 周期，不受 TIMOD 和波特率的影响。如果要在 SPI_RDBR 满载(TIMOD=00)时设置产生一个中断，则中断会在 RXS 置位后延时一个 SCLK 周期发出。当不依赖这个中断时，传输是否终止将通过对 RXS 位的查询而检测到。

询问对于 DMA 接收操作是不必要的，因为中断发生在 DMA 的最后一个存储器写操作之后。对于 DMA 发送操作，需要注意 DMA FIFO 与 SPI DMA FIFO 之间的数据等待时间。

SPI 处理内核传输

在 SPI 处理内核传输过程中，也可利用中断。但是如果 SPI 时钟与内核时钟的比率接近于 1 时，就没有太大意义。对每个单独数据字都要有一个中断，这样 MIPS 性能将受到很大的抑制。

因此，为数据传输过程设置硬件循环(Blackfin 序列发生器的一种特性)是个很好的做法。在这个循环中，必须对每一个传输过程中的每个数据字的适当状态位进行查询。

SPI 传输终止

SPI 的终止操作与其启动程序类似。通过清除适当的从设备选择使能位(FLSx)来取消选择从设备，然后终止 DMA，使 SPI 失效。

另外，可以通过清除错误位(*W1C*)来重置 SPI_STATUS 寄存器。要清除 RXS 位，应对 SPI_RDBR 进行一次假读操作。

概述：DMA 和处理器内核 SPI 传输

Blackfin SPI 控制器提供了两种方法来实现传输。可设置一个基于 DMA 的传输，或者利用处理器内核来访问 SPI_TDBR 或 SPI_RDBR 寄存器，驱动传输过程，并对两个 TIMOD(传输起始模式)位进行设置。

一个基于 DMA 的传输减少了处理器的负荷——其他计算或任务可与 DMA 并行完成。如果需要确认传输(指 DMA 传输)已完成，可以使能中断；否则，就必须由处理器查询 DMA 状态位。

基于 DMA 是首选的传输方式，尤其适用于大量数据传输。

当 Blackfin 处理器在 DMA 接收模式(TIMOD=10)下并且为主控(MSTR=1)设备时，基于 DMA 的传输存在一个弊端。这种情况下，即使最后一个数据字已接收并且传输已完成，SPI 仍旧会驱动 SPI 时钟和从设备选择线路；内循环计数寄存器的当前值为 0(DMAx_CURR_X_XOUNT=0)。这是由于 SPI 和 DMA 不是关于字数同步的。每一个对 SPI_RDBR 的读访问(不管是 DMA 还是内核进行此操作)都会驱动 SPI 从 MISO 线路上得到新的数据，直到 SPI 被终止(SPE=0)。

如果已连接的从设备无法知道发送数据字的数目而且/或者没有计数器在运行以便在适当的时候终止从动设备，从动设备就有可能偶然地被驱动来传输多字，字数超过所请求数目。

在DMA接收模式(TIMOD=10)下，只要有数据在SPI DMA FIFO中(即：FIFO非空)，SPI就持续请求对存储器的DMA写操作。DMA装置持续从SPI DMA FIFO中读取一个数据字并向存储器写，直到SPI DMA字数寄存器从1变为0为止。SPI持续接收数据字直到SPI DMA模式失效。

SPI DMA传输的整个FIFO深度为6：DMA外围设备(SPI)FIFO(4 16/8位字)+SPI_TDBR/SPI_RDBR+SPI接口移位寄存器。参考图10。

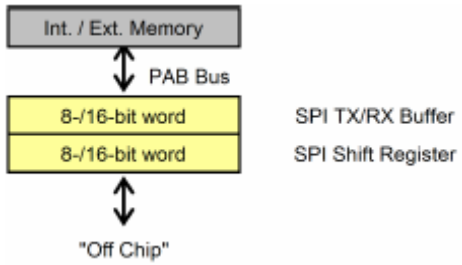

图9 SPI处理核FIFO/总线结构

在这种情况下，或者如果只有少数数据字(比如短期控制信息)必须发送时，一个内核驱动传输就是一个好的选择。有两种传输起始模式：模式一通过 SPI_RDBR(TIMOD=00)的读操作启动传输，而模式二则通过对 SPI_RDBR(TIMOD=01)的写操作启动传输。第二种模式在 SPI 使能(SPE=1)时不能用作传输的立即启动。

这意味着基于内核的SPI主控发送和接收应该以TIMOD 位设置为 00 来完成。对于 SPI 接收，只需要简单的对 SPI_RDBR 缓存器进行读操作。对于SPI发送，首先将SPI_TDBR数据缓存器装满，然后对启动传输的 SPI_RDBR 进行读访问。

关于调试，处理器内核 SPI 传输可以比 SPI DMA 传输更易于进行监控。比如，每个数据字可以以单步模式传输。正在运行的 DMA 不能被终止。如果处理器内核处于暂停状态，DMA 就在后台运行。

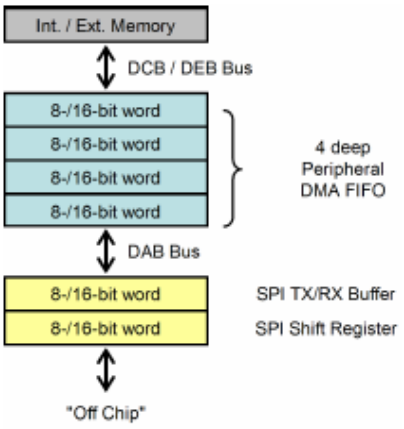

图10 SPI DMA FIFO /总线结构

SPI_RDBR的最先访问是一个假访问；第一个数据字是无效的。每一次对SPI_RDBR的访问会驱动SPI接口取得下一数据字。

最后一个传输后，最后一个数据字必须从SPI_SHADOW数据缓存器(它不会再驱动其他读传输)中读出，或在最后一次访问SPI_RDBR之前先使SPI失效(SPE=0)。仅仅一次SPI_RDBR的访问会清除RXS位(读数据缓存器满载)。

Blackfin 处理器 SPI 举例

关联的.ZIP 文件提供了 ADSP-BF537 Blackfin 处理器作为 SPI 主设备和 SPI 从设备的例子。更详细资料请参考“README.txt”文件。

SHARC 处理器的编程模式

SPI 端口在 ADSP-21161N，ADSP-2126x，ADSP-2136x，以及 ADSP-2137x SHARC 处理器上都可用。以下编程模式讨论主要针对于 ADSP-2126x，ADSP-2136x 和 ADSP-2137x 处理器的应用。对于 ADSP-21161N 处理器，SPI 编程模式不同。该部分描述编程模式的一些*特别的细节*。作为 HRM 现有文件的一个附录和简要总结。

准备工作

SHARC 处理器上的每一个 SPI 端口都有以下信号：

- SPICLK
- SPIDS#
- MOSI
- MISO
- SPI 从设备选择信号(SPIFLG3–0)

ADSP-2116x 和 ADSP-2126x 处理器有一个 SPI 端口，而 SPI 信号在专用的硬件引脚上可用。标记为 3–0 的引脚在 SPI 为主控时用作从设备选择信号。

ADSP-21362/3/4/5/6 处理器有两个 SPI 端口，其中主要的 SPI 信号端口拥有专用的硬件引脚，而次要的 SPI 端口可以通过信号路由单元(SRU)传送至处理器的数字应用接口(DAI)引脚。主要的 SPI 端口利用标志 3-0 引脚作为从设备选择信号，而次要的 SPI 端口有四个专用的 SPIFLGB3-0 信号可以传送到任意一个 DAI 引脚。

对于 ADSP-21367/8/9 和 ADSP-2137x 处理器，两个 SPI 端口都可以通过信号路由单元 2(SRU2)传送到处理器的数字外设接口(DPI)。ADSP-21367/8/9 和 ADSP-2137x 处理器上的两个 SPI 端口都有专用的 SPIFLG3–0 信号在 SPI 作为主设备时用作从设备选择信号。

SHARC 处理器上的 DAI 引脚和 DPI 引脚有一个内部上拉电阻。默认情况下，这个上拉电阻被使能，可以通过写相应的上拉使能寄存器使其失效。当 SPI 信号传送到 DAI 或 DPI 引脚时，可能就不需要再加上拉电阻。

对于 ADSP-21367/8/9 和 ADSP-2137x 处理器，当 SPI 被用作主设备时，利用 SRU2 传送 SPICLK 信号时必须注意应用中的以下几个方面。当 CLKPL=0 时，传送 SPICLK 的 DPI 引脚的使能信号必须配置如下：

```
SRU(SPI_CLK_O,DPI_PB03_I);
SRU(HIGH,DPI_PBEN03_I);
```

当 CLKPL=1 时，传送 SPICLK 的 DPI 引脚使能信号必须配置如下：

```
SRU(SPI_CLK_O,DPI_PB03_I);
SRU(SPI_CLK_PBEN_O,DPI_PBEN03_I);
```

SPI 初始化

SHARC 处理器的 SPI 端口用以下控制寄存器和状态寄存器对 SPI 进行配置：

- SPICTL
- SPIFLG
- SPIBAUD
- SPISTAT

SPICTL 寄存器

SPICTL 寄存器有用于选择 SPI 配置和使能 SPI 的控制位。包括：

- 字长选择
- 主设备或从设备操作
- 数据格式(小的或大的字节存储次序)
- 时钟相位和极性
- 数据引脚的开路漏极输出
- 包使能

对 SPICTL 寄存器的完整描述请参阅处理器的 HRM。

SPIFLG 寄存器

SPIFLG 寄存器只有在 SPI 配置为主设备时才启用。这个寄存器不用于从设备模式的操作。它允许您选择一个或者全部 SPIFLG3-0 信号作为从选择信号。设置 SPIFLG 寄存器的 DSxEN 位可以选择适当的 SPIFLGx 信号作为从选择信号。

当 CPHASE=0 时，从选择信号将由硬件自动生成。SPIFLGx 信号将在每个数据传输之前设定为低，而在连续的传输过程中设为高。

当 CPHASE=1 时，从选择信号只能通过手动方式由用户应用程序生成。可以利用 SPIFLGx 寄存器位实现这个过程。在这种模式下，设置或清除 SPIFLG 寄存器上的 SPIFLGx 位将会对 SPIFLGx 信号产生影响。在 SPI 使能之前，通过设置 SPIFLG 寄存器上的 SPIFLGx 位来确保这个引脚为高电平。传输一开始，在启动传输之前清除 SPIFLG 寄存器上的 SPIFLGx 位。

SPIBAUD 寄存器

SPIBAUD 寄存器只用于 SPI 配置为主设备时。这个寄存器不用于从设备模式的操作。它允许选择 SPI 主控设备的波特率。BAUDR 位(15-1)可用来配置 SPI 波特率。SPI 波特率是由处理器的内核时钟计算得来的。对于 ADSP-2136x 和 ADSP-2137x 处理器，SPI 波特率的计算如下：

SPI 波特率=CCLK / (8 *(BAUDR-1))

对于 ADSP-2126x 处理器，SPI 波特率的计算如下：

SPI 波特率= CCLK / (4 *(BAUDR-1))

SPISTAT 寄存器

SPISTAT 寄存器是只读寄存器，它提供发送和接收缓存器的 FIFO 状态，以及当前传输完成与否。它还有说明发送/接收错误和多主设备错误的数据位。这个寄存器上的数据位都是写 1 清除该位。

SPI DMA 初始化

SPI 有以下用于初始化 DMA 模式操作的寄存器：

- SPIDMAC
- IISPI
- IMSPI
- CSPI
- CPSPI

IISPI，IMSPI，CSPI 和 CPSPI 寄存器是 DMA 参数寄存器。IISPI，IMSPI 和 CSPI 寄存器分别存储用于数据传输的内存储器地址值，地址修正值

和计数值。CPSPI 寄存器在 DMA 链使能时，存储下一个 DMA 序列。

SPIDMAC 寄存器有用于配置发送或接收 DMA，使能中断，使能 DMA 和 DMA 链的控制位。它还有显示当前 DMA 传输状态和一些错误情况的状态位。SPI DMA 通过设置 SPIDMAC 寄存器的 SPIDEN 位来进行初始化。

SPI 传输启动

SPI 传输可以在内核模式或 DMA 模式下进行。SPI 控制寄存器上的 TIOMOD1-0 位选择数据传输模式。在内核模式下，发送缓冲(TXSPI)和接收缓冲(RXSPI)寄存器由内核直接访问。在 DMA 模式下，DMA 控制器对接收缓存或发送缓存访问从而完成数据传输。

当 TIMOD-0 位被配置为“00”时，在使能 SPI 主设备后对 RXSPI 缓存器的读操作会启动一次数据传输。当这两位被配置为“01”时，在使能 SPI 主设备后对 TXSPI 缓存器的写操作会启动一次数据传输。TIMOD1-0 位在 DMA 操作模式下被配置为“10”。根据 DMA 的方向(发送或接收)，DMA 控制器通过写入发送缓存器或从接收缓存器读取操作启动一次数据传输。

DMA控制器有深度为4的FIFO，用于DMA数据传输。对于FIFO上的每组4个数据字会生成一个内部DMA请求。如果DMA计数不是4的倍数，每组的四个数据字都将生成一个请求，最后为剩下的数据字再生成一个请求。例如，如果DMA计数为7，DMA控制器生成两个请求(一个是对4个DMA计数，另一个是对剩下的数据字)。

所有情况下，主控设备为从设备提供 SPI 时钟。从设备选择信号根据 CPHASE 位的配置进行自动或手动生成。

SPI 传输和中断服务

处理器内核驱动的传输可以处于中断驱动模式或轮询检测模式。在轮询检测模式下，用户程序必须不断地通过对 SPITSTAT 寄存器的读操作来检测发送缓存器或接收缓存器的状态。当发送缓存器为空或接收缓存器中有数据时，应用程序可以向发送缓存器中写入数据或从接收缓存器中读出数据。

在中断驱动模式下，中断在发送缓存器为空或接收缓存器有一个完整的数据字时自动生成，这取决于 TIMOD1-0 位的配置。在中断服务程序内部，用户应用程序必须写入发送缓存器或从接收缓存器中读出数据。

总的来说，用户应用程序可以在需要全双工通信的情况下可以利用轮询检测和中断驱动模式的结合。

对于 DMA 驱动的传输操作，在模块传输的结束端会生成一个中断。对于 DMA 链模式，中断可以在每个序列末端或全部序列的末端生成。DMA 中断能够处理通过 SPI 接收的数据块或用于启动一次新的传输。

当SPI主控设备配置为在处理器内核模式下通过对接收缓存器的读操作来开始一次数据传输时，要初始化主设备就必须进行一次假读。这个假读操作产生从设备第一个传输数据的时钟。要接收N个数据字，主控设备就必须在处理器内核模式下完成N+1次数据传输。对于SPI接收DMA操作，由DMA控制器来完成这一假读。

SPI 传输终止

SPI 传输在 SPI 失效时被终止。SPI 可以在中断服务程序内部失效。对于内核模式的操作，在发送或接收完所需数目的数据字后，SPI 可以通过清除 SPICTL 寄存器的内容使其失效。SPISTAT 寄存器的 SPIFE 位必须在使 SPI 失效之前查询，这个位指示的当前传输是否完成。需要确保在 SPI 失效之前当前数据传输已经完成。

对于 DMA 模式的操作，在 ISR 内部，SPIDMAC 寄存器的 SPIDMAS 位必须检测，这一位指示 DMA 的完成状态并在 DMA 传输正在进行时设置(一旦传输完成就被清除)。必须查询 SPISTAT 寄存器的 SPIFE 位以保证最后一个数据传输的完成，然后可以通过清除 SPICTL 和 SPIDMAC 寄存器来禁用 SPI 和 SPIDMA 寄存器。

对于 DMA 链模式，可以通过向链指针寄存器写入 0 来终止 SPI 传输。在写入 0 之后，轮流对 SPIDMAC 寄存器的 SPIDMAS 位和 SPICTL 寄存器的 SPIFE 位进行查询。一旦当前数据传输完成，就可以通过清除 SPICTL 寄存器和 SPIDMAC 寄存器的内容使 SPI 和 SPIDMA 寄存器失效。

SHARC 处理器 SPI 实例

该应用笔记所提供的例子是基于 ADSP-21369 处理器的。SPI 主设备代码将 ADSP-21369 处理器的初始 SPI 设定为 DMA 模式下的主设备。SPIBAUD 寄存器的除数在运行过程中计算，它取决于处理器内核时钟和所选的 SPI 波特率。SPI 从设备代码将 ADSP-21369 处理器的初始 SPI 设定为 DMA 模式下的从设备。对于两个代码实例，DMA 传输的方向可以利用宏指令进行选择。宏指令还可以用于选择时钟相位和极性的配置。

该代码可以直接用于 ADSP-21367/8 和 ADSP-2137x 处理器，而不用进行修改。对于 ADSP-21362/3/4/5/6 和 ADSP-2126x 处理器，SPI 配置代码也可以使用，但是 InitSRU 函数不必调用。对 SPIBAUD 波特率的计算必须换算到 ADSP-2126x 上。

总结

该应用笔记讨论了在 Blackfin 处理器和 SHARC 处理器上的 SPI 编程模式。文件中还提供了在 Blackfin 处理器和 SHARC 处理器之间通过 SPI 进行连续通信的代码范例。这些例程已经在 ADSP-21369 SHARC 处理器和 ADSPBF537 Blackfin 处理器的 SPI 端口之间进行了测试。

附录

与该文件结合的.ZIP 文件包含以下代码范例：

[1] 对于Blackfin处理器作为SPI主设备的代码范例

[2] 对于Blackfin处理器作为SPI从设备的代码范例

[3] 对于SHARC处理器作为SPI主设备的代码范例

[4] 对于SHARC处理器作为SPI从设备的代码范例

参考文献

[1] *ADSP-BF53x/BF56x Blackfin Processor Programming Reference*. Rev. 1.0, June 2005. Analog Devices, Inc.

[2] *ADSP-BF537 Blackfin Processor Hardware Reference.* Rev. 2.0, December 2005. Analog Devices, Inc.

[3] *ADSP-BF534/ADSP-BF536/ADSP-BF537 Blackfin Embedded Processor Data Sheet.* Rev. C, February 2007.Analog Devices, Inc.

[4] *Using the Blackfin Processor SPORT to Emulate a SPI Interface (EE-304)*. Rev. 1, November 10, 2006. Analog Devices, Inc.

[5] *ADSP-21368 SHARC Processor Hardware Reference*. Rev. 1.0, September 2006. Analog Devices, Inc.

[6] *ADSP-2136x SHARC Processor Hardware Reference for ADSP-21362/3/4/5/6 Processors*. Rev. 1.0, October 2005. Analog Devices, Inc.

[7] *ADSP-2126x SHARC Processor Peripherals Manual*. Rev 3.0, December 2005. Analog Devices, Inc.

[8] *ADSP-21371: SHARC Processor Data Sheet*. Rev. 0, July 2007 .Analog Devices, Inc.

[9] *ADSP-21375: 266 MHz High Performance SHARC Processor Preliminary Data Sheet.* Rev. PrB, December 2005. Analog Devices, Inc.

[10] *ADSP-21367/ADSP-21368/ADSP-21369 SHARC Processors Data Sheet.* Rev. A, August 2006. Analog Devices, Inc.

[11] *ADSP-21362/ADSP-21363/ADSP-21364/ADSP-21365/ADSP-21366 SHARC Processors Data Sheet.* Rev. B, June 2007. Analog Devices, Inc.

[12] *ADSP-21261: 3rd Generation, Low-Cost, 150 MHz SHARC Processor Data Sheet.* Rev 0, April 2006. Analog Devices, Inc.

[13] *ADSP-21262: 3rd Generation Low Cost 32-Bit Floating-Point SHARC Processor Data Sheet.* Rev. B, October 2005. Analog Devices, Inc.

[14] *ADSP-21266: High Performance SHARC Audio Processor Data Sheet.* Rev. B, May 2005. Analog Devices, Inc.

[15] *SHARC SPI Slave Booting (EE-177).* Rev. 3, January 19, 2007. Analog Devices, Inc.

文档记录

Revision	Description
Rev 1 – July 8, 2008 *by Jeyanthi Jegadeesan and Andreas Pellkofer*	Initial Release