目 录

Stellaris 外设驱动库——SSI

1.1 SSI 总体特性

Stellaris 系列 ARM 的 SSI（Synchronous Serial Interface，同步串行接口）是与具有 Freescale SPI（飞思卡尔半导体）、MicroWire（美国国家半导体）、Texas Instruments（德州仪器，TI）同步串行接口的外设器件进行同步串行通信的主机或从机接口。SSI 接口是 Stellaris 系列 ARM 都支持的标准外设，也是流行的外部串行总线之一。SSI 具有以下主要特性：

- 主机或从机操作
- 时钟位速率和预分频可编程
- 独立的发送和接收 FIFO，16 位宽，8 个单元深
- 接口操作可编程，以实现 Freescale SPI、MicroWire 或 TI 的串行接口
- 数据帧大小可编程，范围 4～16 位
- 内部回环测试模式，可进行诊断/调试测试

1.2 SSI 通信协议

对于 Freescale SPI、MICROWIRE、Texas Instruments3 种帧格式，当 SSI 空闲时串行时钟（SSICLK）都保持不活动状态，只有当数据发送或接收时处于活动状态，SSICLK 才在设置好的频率下工作。利用 SSICLK 的空闲状态可提供接收超时指示。如果一个超时周期之后接收 FIFO 仍含有数据，则产生超时指示。

对于 Freescale SPI 和 MICROWIRE 这两种帧格式，串行帧（SSIFss）管脚为低电平有效，并在整个帧的传输过程中保持有效（被下拉）。

而对于 Texas Instruments 同步串行帧格式，在发送每帧之前，每遇到 SSICLK 的上升沿开始的串行时钟周期时，SSIFss 管脚就跳动一次。在这种帧格式中，SSI 和片外从器件在 SSICLK 的上升沿驱动各自的输出数据，并在下降沿锁存来自另一个器件的数据。

不同于其它两种全双工传输的帧格式，在半双工下工作的 MICROWIRE 格式使用特殊的主-从消息技术。在该模式中，帧开始时向片外从机发送 8 位控制消息。在发送过程中，SSI 没有接收到输入的数据。在消息已发送之后，片外从机对消息进行译码，并在 8 位控制消息的最后一位也已发送出去之后等待一个串行时钟，之后以请求的数据来响应。返回的数据在长度上可以是 4～16 位，使得在任何地方整个帧长度为 13～25 位。

1.2.1 Texas Instruments 同步串行帧格式

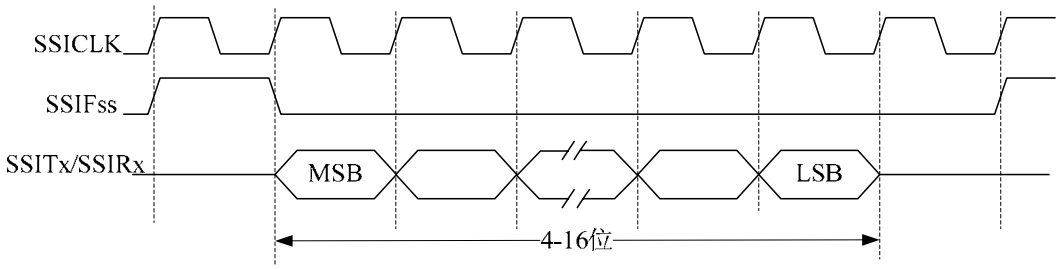

图 1　TI 同步串行帧格式（单次传输）

图1 显示了一次传输的 Texas Instruments 同步串行帧格式。

在该模式中，任何时候当 SSI 空闲时，SSICLK 和 SSIFss 被强制为低电平，发送数据线 SSITx 为三态。一旦发送 FIFO 的底部入口包含数据，SSIFss 变为高电平并持续一个 SSICLK 周期。即将发送的值也从发送 FIFO 传输到发送逻辑的串行移位寄存器中。在 SSICLK 的下一个上升沿，4～16 位数据帧的 MSB 从 SSITx 管脚移出。同样地，接收数据的 MSB 也通过片外串行从器件移到 SSIRx 管脚上。

然后，SSI 和片外串行从器件都提供时钟，供每个数据位在每个 SSICLK 的下降沿进入各自的串行移位器中。在已锁存 LSB 之后的第一个 SSICLK 上升沿上，接收数据从串行移位器传输到接收 FIFO。

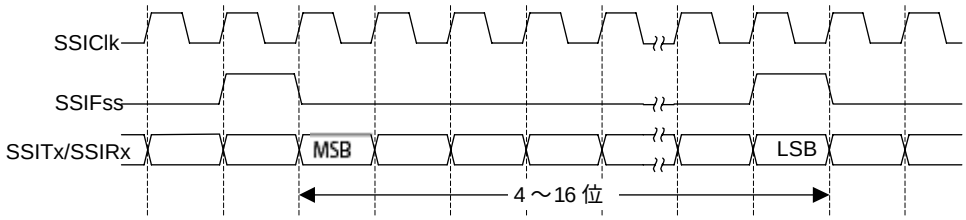

图 2　TI 同步串行帧格式（连续传输）

图2 显示了背对背（back-to-back）传输时的 Texas Instruments 同步串行帧格式。

1.2.2　Freescale SPI 帧格式

Freescale SPI（Motorala SPI）接口是一个 4 线接口，其中 SSIFss 信号用作从机选择。Freescale SPI 格式的主要特性为：SSICLK 信号的不活动状态和相位可以通过 **SSISCR0** 控制寄存器中的 SPO 和 SPH 位来设置。

- SPO 时钟极性位

当 SPO 时钟极性控制位为 0 时，在没有数据传输时 SSICLK 管脚上将产生稳定的低电平。如果 SPO 位为 1，则在没有进行数据传输时在 SSICLK 管脚上产生稳定的高电平。

- SPH 相位控制位

SPH 相位控制位选择捕获数据以及允许数据改变状态的时钟边沿。通过在第一个数据捕获边沿之前允许或不允许时钟转换，从而在第一个被传输的位上产生极大的影响。当 SPH 相位控制位为 0 时，在第一个时钟边沿转换时捕获数据。如果 SPH 位为 1，则在第二个时钟边沿转换时捕获数据。

1．　SPO=0 和 SPH=0 时的 Freescale SPI 帧格式

SPO=0 和 SPH=0 时，Freescale SPI 帧格式的单次和连续传输信号序列如*图3* 和*图4* 所示。

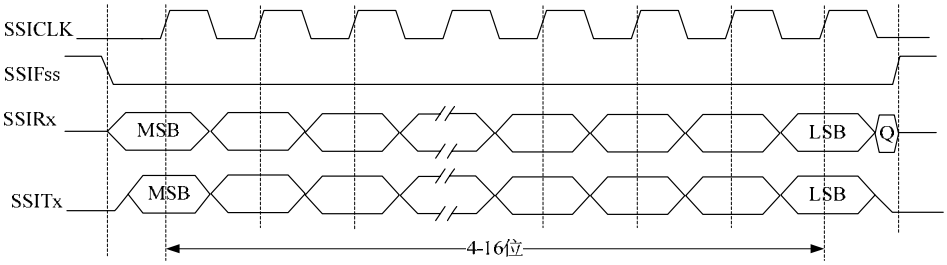

图 3　SPO=0 和 SPH=0 时的 Freescale SPI 帧格式（单次传输）

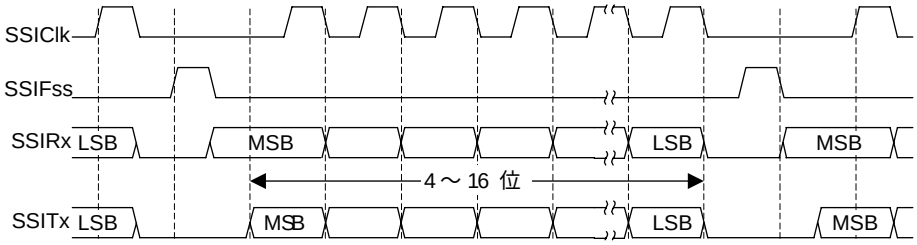

图 4　SPO=0 和 SPH=0 时的 Freescale SPI 帧格式（连续传输）

在上述配置中，SSI 处于空闲周期时：

（1）SSICLK 强制为低电平

（2）SSIFss 强制为高电平

（3）发送数据线 SSITx 强制为低电平

（4）当 SSI 配置为主机时，使能 SSICLK 端口

（5）当 SSI 配置为从机时，禁止 SSICLK 端口

如果 SSI 使能并且在发送 FIFO 中含有有效的数据，则通过将 SSIFss 主机信号驱动为低电平表示发送操作开始。这使得从机数据能够放在主机的 SSIRx 输入线上，主机 SSITx 输出端口使能。在半个 SSICLK 周期之后，有效的主机数据传输到 SSITx 管脚。既然主机和从机数据都已设置好，则在下面的半个 SSICLK 周期之后，SSICLK 主机时钟管脚变为高电平。在 SSICLK 的上升沿捕获数据，该操作延续到 SSICLK 信号的下降沿。

如果是传输一个字，则在数据字的所有位都已传输完之后，在捕获到最后一个位之后的一个 SSICLK 周期后，SSIFss 线返回到其空闲的高电平状态。

在连续的背对背传输中，数据字的每次传输之间 SSIFss 信号必须变为高电平。这是因为如果 SPH 位为逻辑 0，则从机选择管脚将其串行外设寄存器中的数据固定，不允许修改。因此，主器件必须在每次数据传输之间将从器件的 SSIFss 管脚拉高，来使能串行外设的数据写操作。当连续传输完成时，在捕获到最后一个位之后的一个 SSICLK 周期后，SSIFss 管脚返回到其空闲状态。

2．　SPO=0 和 SPH=1 时的 Freescale SPI 帧格式

SPO=0 和 SPH=1 时，Freescale SPI 帧格式的传输信号序列如*图 5* 所示，该图涵盖了单次和连续传输这两种情况。

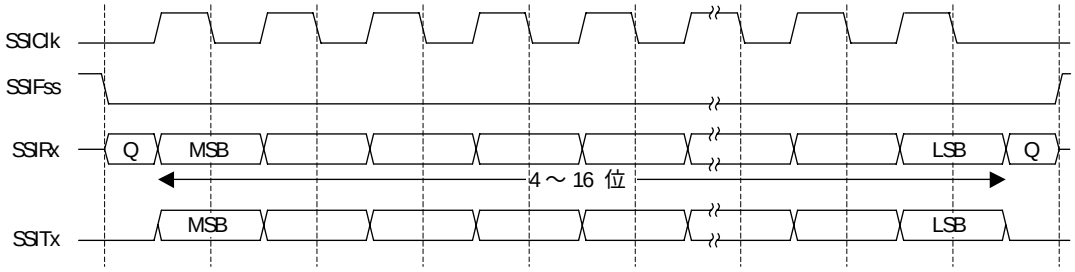

图 5　SPO=0 和 SPH=1 时的 Freescale SPI 帧格式

在该配置中，SSI 处于空闲周期时：

（1）SSICLK 强制为低电平

（2）SSIFss 强制为高电平

（3）发送数据线 SSITx 强制为低电平

（4）当 SSI 配置为主机时，使能 SSICLK 端口

（5）当 SSI 配置为从机时，禁止 SSICLK 端口

如果 SSI 使能并且在发送 FIFO 中含有有效的数据，则通过将 SSIFss 主机信号驱动为低电平表示发送操作开始。主机 SSITx 输出使能。在下面的半个 SSICLK 周期之后，主机和从机有效数据能够放在各自的传输线上。同时，利用一个上升沿转换来使能 SSICLK。然后，在 SSICLK 的下降沿捕获数据，该操作一直延续到 SSICLK 信号的上升沿。

如果是传输一个字，则在所有位传输完之后，在捕获到最后一个位之后的一个 SSICLK 周期，SSIFss 线返回到其空闲的高电平状态。

如果是背对背（back-to-back）传输，则在两次连续的数据字传输之间 SSIFss 管脚保持低电平，连续传输的结束情况与单个字传输相同。

3．　SPO=1 和 SPH=0 时的 Freescale SPI 帧格式

SPO=1 和 SPH=0 时，Freescale SPI 帧格式的单次和连续传输信号序列如*图 6* 和*图 7* 所示。

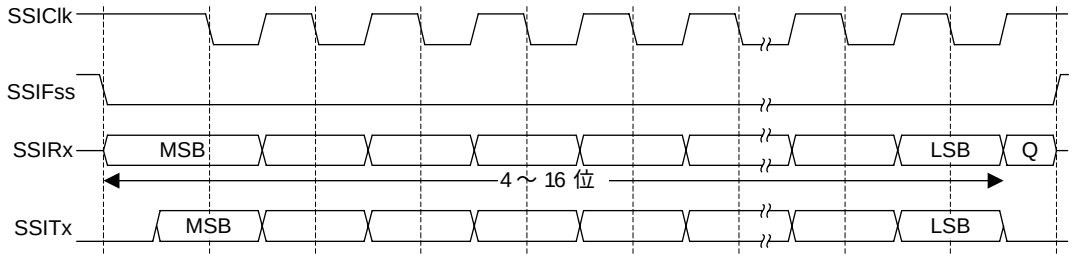

图 6　SPO=1 和 SPH=0 时的 Freescale SPI 帧格式（单次传输）

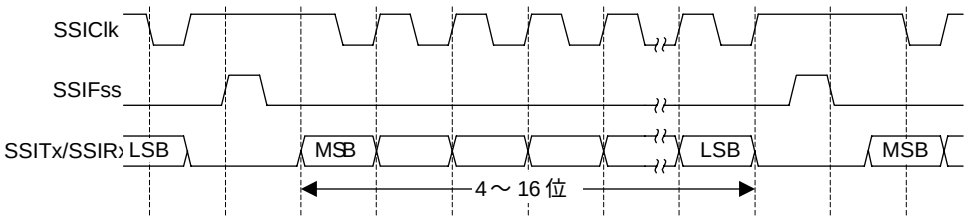

图 7　SPO=1 和 SPH=0 时的 Freescale SPI 帧格式（连续传输）

在该配置中，SSI 处于空闲周期时：

（1）SSICLK 强制为高电平

（2）SSIFss 强制为高电平

（3）发送数据线 SSITx 强制为低电平

（4）SSI 配置为主机时，使能 SSICLK 引脚

（5）SSI 配置为从机时，禁止 SSICLK 引脚

如果 SSI 使能并且在发送 FIFO 中含有有效的数据，则通过将 SSIFss 主机信号驱动为低电平表示传输操作开始，这可使从机数据立即传输到主机的 SSIRx 线上。主机 SSITx 输出引脚使能。半个周期之后，有效的主机数据传输到 SSITx 线上。既然主机和从机的有效数据都已设置好，则在下面的半个 SSICLK 周期之后，SSICLK 主机时钟管脚变为低电平。这表示数据在下降沿被捕获并且该操作延续到 SSICLK 信号的上升沿。

如果是单个字传输，则在数据字的所有位传输完之后，在最后一个位传输完之后的一个 SSICLK 周期，SSIFss 线返回到其空闲的高电平状态。

而在连续的背对背（back-to-back）传输中，每次数据字传输之间 SSIFss 信号必须变为高电平。这是因为如果 SPH 位为逻辑 0，则从机选择管脚使其串行外设寄存器中的数据固定，不允许修改。因此，每次数据传输之间，主器件必须将从器件的 SSIFss 管脚拉为高电平来使能串行外设的数据写操作。在连续传输完成时，最后一个位被捕获之后的一个 SSICLK 周期，SSIFss 管脚返回其空闲状态。

4．　SPO=1 和 SPH=1 时的 Freescale SPI 帧格式

SPO=1 和 SPH=1 时，Freescale SPI 帧格式的传输信号序列如*图 8* 所示。该图涵盖了单次和连续传输两种情况。

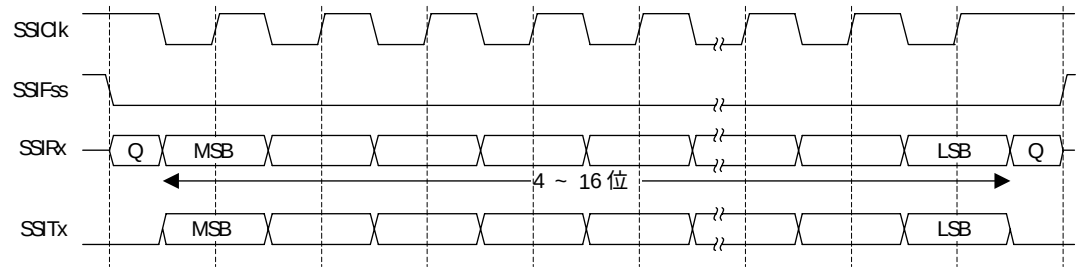

图 8　SPO=1 和 SPH=1 时的 Freescale SPI 帧格式

（注：图中的 Q 表示未定义）

在该配置中，SSI 处于空闲周期时：

（1）SSICLK 强制为高电平

（2）SSIFss 强制为高电平

（3）发送数据线 SSIFss 强制为低电平

（4）当 SSI 配置为主机时，使能 SSICLK 引脚

（5）当 SSI 配置为从机时，禁止 SSICLK 引脚

如果 SSI 使能并且在发送 FIFO 中含有有效的数据，则通过将 SSIFss 主机信号驱动为低电平表示发送操作开始。主机 SSITx 输出引脚使能。在下面的半个 SSICLK 周期之后，主机和从机数据都能够放在各自的传输线上。同时，利用 SSICLK 的下降沿转换来使能 SSICLK。然后在上升沿捕获数据，并且该操作延续到 SSICLK 信号的下降沿。

在所有位传输完之后，如果是单个字传输，则在最后一个位捕获完之后的一个 SSICLK 周期中，SSIFss 线返回到其空闲的高电平状态。

而对于连续的背对背(back-to-back)传输，$SSIF_{SS}$管脚保持其有效的低电平状态，直至最后一个字的最后一位捕获完，再返回其上述的空闲状态。

而对于连续的背对背(back-to-back)传输，在两次连续的数据字传输之间 SSIFss 管脚保持低电平，连续传输的结束情况与单个字传输相同。

1.2.3 MICROWIRE 帧格式

图9 显示了单次传输的 MICROWIRE 帧格式，而*图10* 为该格式的背对背（back-to-back）传输情况。

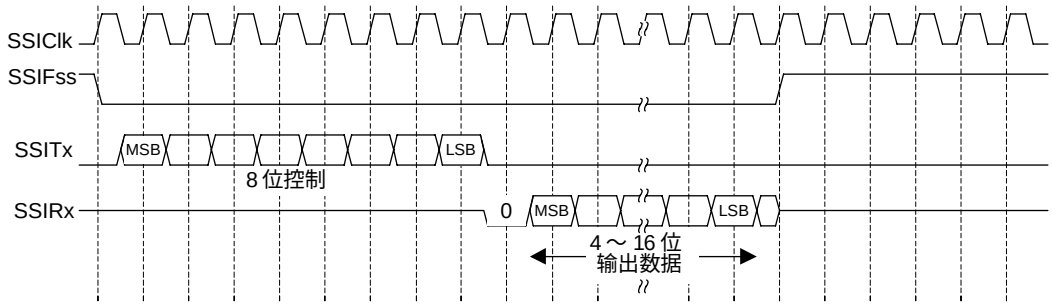

图 9　MICROWIRE 帧格式（单次传输）

MICROWIRE 格式与 SPI 格式非常类似，只是 MICROWIRE 为半双工而不是全双工，使用主-从消息传递技术。每次串行传输都由 SSI 向片外从器件发送 8 位控制字开始。在此传输过程中，SSI 没有接收到输入的数据。在消息已发送完之后，片外从机对消息进行译码，SSI 在将 8 位控制消息的最后一位发送完之后等待一个串行时钟，之后以请求的数据来响应。返回的数据在长度上为 4～16 位，使得任何地方总的帧长度为 13～25 位。

在该配置中，SSI 处于空闲状态时：

（1）SSICLK 强制为低电平

（2）SSIFss 强制为高电平

（3）数据线 SSITx 强制为低电平。

通过向发送 FIFO 写入一个控制字节来触发一次传输。SSIFss 的下降沿使得包含在发送 FIFO 底部入口的值能够传输到发送逻辑的串行移位寄存器中，并且 8 位控制帧的 MSB 移出

到 SSITx 管脚上。在该控制帧传输期间 SSIFss 保持低电平，SSIRx 管脚保持三态。

片外串行从器件在 SSICLK 的上升沿时将每个控制位锁存到其串行移位器中。在将最后一位锁存之后，从器件在一个时钟的等待状态中对控制字节进行译码，并且从机通过将数据发送回 SSI 来响应。数据的每个位在 SSICLK 的下降沿时驱动到 SSIRx 线上。SSI 在 SSICLK 的上升沿依次将每个位锁存。在帧传输结束时，对于单次传输，在最后一位已锁存到接收串行移位器之后的一个时钟周期，SSIFss 信号被拉为高电平。这使得数据传输到接收 FIFO 中。**注** :在接收移位器已将 LSB 锁存之后的 SSICLK 的下降沿上或在 SSIFss 管脚变为高电平时，片外从器件能够将接收线置为三态。

对于连续传输，数据传输的开始与结束与单次传输相同。但 SSIFss 线持续有效（保持低电平）并且数据传输以背对背(back-to-back)方式产生。在接收到当前帧的接收数据的 LSB 之后，立即跟随下一帧的控制字节。在当前帧的 LSB 已锁存到 SSI 之后，接收数据的每个位在 SSICLK 的下降沿从接收移位器中进行传输。

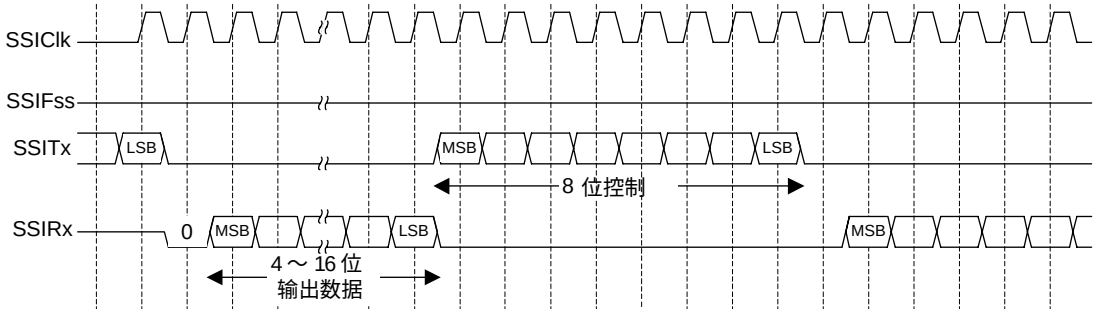

图 10　MICROWIRE 帧格式（连续传输）

在 MICROWIRE 模式中，SSIFss 变为低电平之后的 SSICLK 上升沿上，SSI 从机对接收数据的第一个位进行采样。驱动自由运行 SSICLK 的主机必须确保 SSIFss 信号相对于 SSICLK 的上升沿具有足够的建立时间和保持时间裕量（setup and hold margins）。

图 11 阐明了建立和保持时间要求。相对于 SSICLK 的上升沿（在该上升沿上，SSI 从机将对接收数据的第一个位进行采样），SSIFss 的建立时间至少必须是 SSI 进行操作的 SSICLK 周期的两倍。相对于该边沿之前的 SSICLK 上升沿，SSIFss 至少必须具有一个 SSICLK 周期的保持时间。

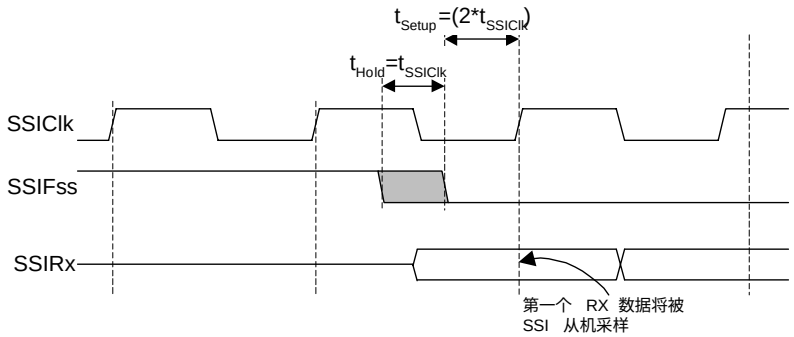

图 11　MICROWIRE 帧格式，SSIFss 输入建立和保持时间要求

1.3 SSI 功能概述

SSI 对从外设器件接收到的数据执行串行到并行转换。CPU 可以访问 SSI 数据寄存器来发送和获得数据。发送和接收路径利用内部 FIFO 存储单元进行缓冲，以允许最多 8 个 16 位的值在发送和接收模式中独立地存储。

1.3.1 位速率和帧格式

SSI 包含一个可编程的位速率时钟分频器和预分频器来生成串行输出时钟。尽管最大位速率由外设器件决定，但 1.5MHz 及更高的位速率仍是支持的。

串行位速率是通过对输入的系统时钟进行分频来获得。虽然理论上 SSICLK 发送时钟可达到 25MHz，但模块可能不能在该速率下工作。发送操作时，系统时钟速率至少必须时 SSICLK 的两倍。接收操作时，系统时钟速率指导必须时 SSICLK 的 12 倍。

SSI 通信的帧格式有 3 种：Texas Instruments 同步串行数据帧、Freescal SPI 数据帧、MICROWIRE 串行数据帧。

根据已设置的数据大小，每个数据帧长度在 4～16 位之间，并采用 MSB 在前的方式发送。

1.3.2 FIFO 操作

对 FIFO 的访问是通过 SSI 数据寄存器（SSIDR）中写入与读出数据来实现的，SSIDR 为 16 为宽的数据寄存器，可以对它进行读写操作，SSIDR 实际对应两个不同的物理地址，以分别完成对发送 FIFO 和接收 FIFO 的操作。

SSIDR 的读操作即是对接收 FIFO 的入口（由当前 FIFO 读指针来指向）进行访问。当 SSI 接收逻辑将数据从输入的数据帧中转移出来后，将它们放入接收 FIFO 的入口（由当前 FIFO 写指针来指向）。

SSIDR 的写操作即是将数据写入发送 FIFO 的入口（由写指针来指向）。每次，发送逻辑将发送 FIFO 中的数值转移出来一个，装入发送串行移位器，然后在设置的位速率下串行溢出到 SSITx 管脚。

当所选的数据长度小于 16 位时，用户必须正确调整写入发送 FIFO 的数据，发送逻辑忽略高位未使用的位。小于 16 位的结合搜数据在记诶手传冲去中自动调整。

当 SSI 设置为 MICROWIRE 帧格式时，发送数据的默认大小为 8 为（最高有效字节忽略），接收数据的大小由程序员控制。即使当 SSICR1 寄存器的 SSE 位设置为 0 时（禁止 SSI 端口），也可以不将发送 FIFO 和接收 FIFO 清零。这样可在使能 SSI 之前使用软件来填充发送 FIFO。

- 发送 FIFO

通用发送 FIFO 是 16 位宽、8 单元深、先进先出的存储缓冲区。CPU 通过写 SSI 数据寄存器 SSIDR 来将数据写入发送 FIFO，数据在由发送逻辑读出之前一直保存在发送 FIFO 中。

当 SSI 配置为主机或从机时，并行数据先写入发送 FIFO，再转换成串行数据并通过 SSITx 管脚分别发送到相关的从机或主机。

- 接收 FIFO

通用接收 FIFO 是一个 16 位宽、8 单元深、先进先出的存储缓冲区。从串行接口接收到的数据在由 CPU 读出之前一直保存在缓冲区中，CPU 通过读 SSIDR 寄存器来访问读 FIFO。

当 SSI 配置位主机或从机时，通过 SSIRx 管脚接收到的串行数据转换成并行数据后装载到相关的从机或主机接收 FIFO。

1.3.3 SSI 中断

SSI 在满足以下条件时能够产生中断：

- 发送 FIFO 服务
- 接收 FIFO 服务
- 接收 FIFO 超时
- 接收 FIFO 溢出

所有中断事件在发送到嵌套中断向量控制器之前先要执行“或”操作，因此，在任何给定的时刻 SSI 只能向中断控制器产生一个中断请求。通过对 SSI 中断屏蔽寄存器（SSIIM）中的对应位进行设置，你可以屏蔽 4 个单独屏蔽的中断里的任一个，将适当的屏蔽位置 1 可使能中断。

SSI 提供单独的输出和组合的中断输出，这样可允许全局中断服务程序或组合的逻辑驱动程序来处理中断。发送或接收动态数据流的中断已与状态中断分开，这样，根据 FIFO 出发点（trigger level）可以对数据执行读和写操作。各个中断源的状态可从 SSI 原始中断状态（SSIRIS）和 SSI 屏蔽后的中断状态寄存器（SSIMIS）中读出。

1.4 SSI 库函数参考

1.4.1 配置与控制

1． SSIConfigSetExpClk( )

表 1 函数 SSIConfigSetExpClk( )

功能	SSI 配置（需要提供明确的时钟速度）
原型	void SSIConfigSetExpClk(unsigned long ulBase , unsigned long ulSSIClk , unsigned long ulProtocol , unsigned long ulMode , unsigned long ulBitRate , unsigned long ulDataWidth)
参数	ulBase：SSI 模块的基址，应当取下列值之一： SSI_BASE // SSI 模块的基址（用于仅含有 1 个 SSI 模块的芯片） SSI0_BASE // SSI0 模块的基址（等同于 SSI_BASE） SSI1_BASE // SSI1 模块的基址 ulSSIClk：提供给 SSI 模块的时钟速度 ulProtocol：数据传输的协议，应当取下列值之一： SSI_FRF_MOTO_MODE_0 // Freescale（飞思卡尔半导体）格式，极性 0，相位 0 SSI_FRF_MOTO_MODE_1 // Freescale（飞思卡尔半导体）格式，极性 0，相位 1 SSI_FRF_MOTO_MODE_2 // Freescale（飞思卡尔半导体）格式，极性 1，相位 0 SSI_FRF_MOTO_MODE_3 // Freescale（飞思卡尔半导体）格式，极性 1，相位 1

	SSI_FRF_TI // TI（德州仪器）格式 SSI_FRF_NMW // National（美国国家半导体）MicroWire 格式 ulMode：SSI 模块的工作模式，应当取下列值之一： SSI_MODE_MASTER // SSI 主模式 SSI_MODE_SLAVE // SSI 从模式 SSI_MODE_SLAVE_OD // SSI 从模式（输出禁止） ulBitRate：SSI 的位速率，这个位速率必须满足下面的时钟比率标准： ulBitRate ≤ FSSI／2（主模式） ulBitRate ≤ FSSI／12（从模式） 其中 FSSI 是提供给 SSI 模块的时钟速率 ulDataWidth：数据宽度，取值 4～16
返回	无

2． SSIConfig( )

这是个宏函数，为了实际编程的方便，常常用来代替函数 SSIConfigSetExpClk( )。

表 2 宏函数 SSIConfig( )

功能	SSI 配置
原型	#define SSIConfig(a , b , c , d , e) SSIConfigSetExpClk(a , SysCtlClockGet( ) , b , c , d , e)
参数	详见函数 SSIConfigSetExpClk( )的描述
返回	无

3． SSIEnable( )

表 3 函数 SSIEnable( )

功能	使能 SSI 发送和接收
原型	void SSIEnable(unsigned long ulBase)
参数	ulBase：SSI 模块的基址，取值 SSI_BASE、SSI0_BASE 或 SSI1_BASE
返回	无

4． SSIDisable( )

表 4 函数 SSIDisable( )

功能	禁止 SSI 发送和接收
原型	void SSIDisable(unsigned long ulBase)
参数	ulBase：SSI 模块的基址，取值 SSI_BASE、SSI0_BASE 或 SSI1_BASE
返回	无

1.4.2 数据收发

1． SSIDataPutNonBlocking( )

表 5 函数 SSIDataPutNonBlocking( )

功能	将一个数据单元放入 SSI 的发送 FIFO 里（不等待）
原型	long SSIDataPutNonBlocking(unsigned long ulBase , unsigned long ulData)
参数	ulBase：SSI 模块的基址，取值 SSI_BASE、SSI0_BASE 或 SSI1_BASE ulData：要发送的数据单元（4～16 个有效位）
返回	返回写入发送 FIFO 的数据单元数量（如果发送 FIFO 里没有可用的空间则返回 0）

2． SSIDataGetNonBlocking( )

表 6 函数 SSIDataGetNonBlocking( )

功能	从 SSI 的接收 FIFO 里读取一个数据单元（不等待）
原型	long SSIDataGetNonBlocking(unsigned long ulBase , unsigned long *pulData)
参数	ulBase：SSI 模块的基址，取值 SSI_BASE、SSI0_BASE 或 SSI1_BASE pulData：指针，指向保存读取到的数据单元地址
返回	返回从接收 FIFO 里读取到的数据单元数量（如果接收 FIFO 为空，则返回 0）

3． SSIDataNonBlockingPut( )

表 7 宏函数 SSIDataNonBlockingPut( )

功能	将一个数据单元放入 SSI 的发送 FIFO 里（不等待）
原型	#define SSIDataNonBlockingPut(a , b) SSIDataPutNonBlocking(a , b)
参数	参见函数 SSIDataPutNonBlocking( )的描述
返回	参见函数 SSIDataPutNonBlocking( )的描述

4． SSIDataNonBlockingGet( )

表 8 宏函数 SSIDataNonBlockingGet( )

功能	从 SSI 的接收 FIFO 里读取一个数据单元（不等待）
原型	#define SSIDataNonBlockingGet(a , b) SSIDataGetNonBlocking(a , b)
参数	参见函数 SSIDataGetNonBlocking( )的描述
返回	参见函数 SSIDataGetNonBlocking( )的描述

5． SSIDataPut( )

表 9 函数 SSIDataPut( )

功能	将一个数据单元放入 SSI 的发送 FIFO 里

原型	void SSIDataPut(unsigned long ulBase , unsigned long ulData)
参数	ulBase：SSI 模块的基址，取值 SSI_BASE、SSI0_BASE 或 SSI1_BASE ucData：要发送数据单元（4～16 个有效位）
返回	无

6． SSIDataGet( )

表 10 函数 SSIDataGet( )

功能	从 SSI 的接收 FIFO 里读取一个数据单元
原型	void SSIDataGet(unsigned long ulBase , unsigned long *pulData)
参数	ulBase：SSI 模块的基址，取值 SSI_BASE、SSI0_BASE 或 SSI1_BASE pulData：指针，指向保存读取到的数据单元地址
返回	无

1.4.3 中断控制

1． SSIIntEnable( )

表 11 函数 SSIIntEnable( )

功能	使能单独的（一个或多个）SSI 中断源
原型	void SSIIntEnable(unsigned long ulBase , unsigned long ulIntFlags)
参数	ulBase：SSI 模块的基址，取值 SSI_BASE、SSI0_BASE 或 SSI1_BASE ulIntFlags：指定的中断源，应当取下列值之一或者它们之间的任意“或运算”组合形式： SSI_TXFF // 发送 FIFO 半空或不足半空 SSI_RXFF // 接收 FIFO 半满或超过半满 SSI_RXTO // 接收超时（接收 FIFO 已有数据但未半满，而后续数据长时间不来） SSI_RXOR // 接收 FIFO 溢出
返回	无

2． SSIIntDisable( )

表 12 函数 SSIIntDisable( )

功能	禁止单独的（一个或多个）SSI 中断源
原型	void SSIIntDisable(unsigned long ulBase , unsigned long ulIntFlags)
参数	参见函数 SSIIntEnable( )的描述
返回	无

3． SSIIntStatus( )

表 13 函数 SSIIntStatus( )

功能	获取 SSI 当前的中断状态

原型	unsigned long SSIIntStatus(unsigned long ulBase , tBoolean bMasked)
参数	ulBase：SSI 模块的基址，取值 SSI_BASE、SSI0_BASE 或 SSI1_BASE bMasked：如果需要获取原始的中断状态，则取值 false 如果需要获取屏蔽的中断状态，则取值 true
返回	当前中断的状态，参见函数 SSIIntEnable( )里参数 ulIntFlags 的描述

4． SSIIntClear( )

表 14 函数 SSIIntClear( )

功能	清除 SSI 的中断
原型	void SSIIntClear(unsigned long ulBase , unsigned long ulIntFlags)
参数	参见函数 SSIIntEnable( )的描述
返回	无

5． SSIIntRegister( )

表 15 函数 SSIIntRegister( )

功能	注册一个 SSI 中断服务函数
原型	void SSIIntRegister(unsigned long ulBase , void (*pfnHandler)(void))
参数	ulBase：SSI 模块的基址，取值 SSI_BASE、SSI0_BASE 或 SSI1_BASE pfnHandler：指针，指向 SSI 中断出现时被调用的函数
返回	无

6． SSIIntUnregister( )

表 16 函数 SSIIntUnregister( )

功能	注销 SSI 的中断服务函数
原型	void SSIIntUnregister(unsigned long ulBase)
参数	ulBase：SSI 模块的基址，取值 SSI_BASE、SSI0_BASE 或 SSI1_BASE
返回	无