

Avalon[®] 接口规范

针对 Intel[®] Quartus[®] Prime 设计套件的更新: **18.1**



MNL-AVABUSREF | 2018.09.26

官网最新文档: [PDF](#) | [HTML](#)

内容

1. Avalon® 接口规范简介	4
1.1. Avalon 属性和参数	4
1.2. 信号角色(Signal Roles)	5
1.3. 接口时序(Interface Timing)	5
1.4. 示例：系统设计中的 Avalon 接口	5
2. Avalon 时钟和复位接口	8
2.1. Avalon Clock Sink 信号角色	8
2.2. Clock Sink 属性	8
2.3. 相关联的时钟接口	9
2.4. Avalon Clock Source 信号角色	9
2.5. Clock Source 属性	9
2.6. Reset Sink	9
2.7. Reset Sink 接口属性	10
2.8. 相关联的复位接口	10
2.9. 复位源(Reset Source)	10
2.10. Reset Source 接口属性	10
3. Avalon 存储器映射的接口(Avalon Memory-Mapped Interface)	11
3.1. Avalon 存储器映射的接口简介	11
3.2. Avalon 存储器映射的接口信号角色	13
3.3. 接口属性	16
3.4. 时序(timing)	18
3.5. 传输(transfer)	18
3.5.1. 典型的读传输和写传输	18
3.5.2. 使用 waitrequestAllowance 属性进行传输	20
3.5.3. 固定等待状态的读和写传输(Read and Write Transfers with Fixed Wait-States)	23
3.5.4. 流水线传输(Pipelined Transfers)	23
3.5.5. 突发传输(Burst Transfers)	26
3.5.6. 读和写响应	29
3.6. 地址对齐(Address Alignment)	30
3.7. Avalon -MM Slave 寻址	31
4. Avalon 中断接口	32
4.1. 中断发送器(Interrupt Sender)	32
4.1.1. Avalon 中断发送器信号角色	32
4.1.2. 中断发送器属性(Interrupt Sender Properties)	32
4.2. 中断接收器(Interrupt Receiver)	32
4.2.1. Avalon 中断接收器信号角色	33
4.2.2. 中断接收器属性(Interrupt Receiver Properties)	33
4.2.3. 中断时序	33
5. Avalon Streaming 接口	34
5.1. 术语和概念	35
5.2. Avalon Streaming 接口信号角色	35
5.3. 信号排序和时序(Signal Sequencing and Timing)	36

5.3.1. 同步接口.....	36
5.3.2. 时钟使能(Clock Enables).....	36
5.4. Avalon -ST 接口属性.....	36
5.5. 典型的数据传输.....	37
5.6. 信号详情.....	37
5.7. 数据布局(Data Layout).....	38
5.8. 无背压的数据传输.....	39
5.9. 有背压的数据传输.....	39
5.9.1. 使用 readyLatency 和 readyAllowance 的数据传输.....	40
5.9.2. 使用 readyLatency 的数据传输.....	42
5.10. 数据包数据传输(Packet Data Transfers).....	43
5.11. 信号详情.....	44
5.12. 协议详情.....	45
6. Avalon Conduit 接口.....	46
6.1. Avalon 管道(Conduit)信号角色.....	47
6.2. 管道角色(Conduit Properties)	47
7. Avalon 三态管道接口(Avalon Tristate Conduit Interface).....	48
7.1. Avalon 三态管道(Conduit)信号角色.....	49
7.2. 三态管道属性(Tristate Conduit Properties).....	50
7.3. 三态管道时序(Tristate Conduit Timing).....	50
A. 已弃用的信号.....	51
B. Avalon 接口规范的文档修订历史.....	52



1. Avalon® 接口规范简介

Avalon® 接口使您能够轻松连接 Intel® FPGA 中的各个组件，从而简化了系统设计。Avalon 接口系列对应用于流式传输高速数据，读写寄存器和存储器以及控制片外器件的接口进行了定义。Platform Designer 中的组件采用这些标准接口。此外，您可以在自定义组件中使用 Avalon 接口，以增强设计的互操作性。

本规范定义了所有的 Avalon 接口。阅读本规范后，您应该了解哪些接口适合您的组件以及哪些信号角色用于特定行为。本规范定义了以下七个接口：

- **Avalon Streaming Interface (Avalon -ST)**—支持单向数据流的接口，包括多路复用流，数据包和 DSP 数据。 **流媒体接口**
- **Avalon Memory Mapped Interface (Avalon -MM)**—一种基于地址的读/写接口，是主 - 从连接的典型接口。 **映射**
- **Avalon Conduit Interface**—一种接口类型，适用于那些不适合任何其他 Avalon 类型的单个信号或信号组。您可以在一个 Platform Designer 系统内部连接管道接口(conduit interface)。或者，您可以将它们导出以连接到设计中的其他模块或者连接到 FPGA 管脚。
- **Avalon Tri-State Conduit Interface (Avalon -TC)**—支持与片外(off-chip)连接的接口。多个外设可以通过信号多路复用(signal multiplexing)来共享管脚，从而减少 FPGA 的管脚数和 PCB 上的走线数量。
- **Avalon Interrupt Interface**—允许组件向其他组件发送事件信号的接口。
- **Avalon Clock Interface**—驱动或接收时钟的接口。
- **Avalon Reset Interface**—提供复位连接的接口。

一个组件可以包括任意数量的这些接口，并且还可以包括相同接口类型的多个实例。

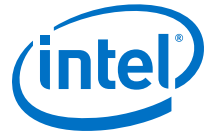
注意: Avalon 接口是一个开放的标准。对使用或基于 Avalon 接口的产品进行开发和销售时不需要许可(license)和版税(royalty)。

相关链接

- [Intel FPGA IP 核的简介](#)
提供有关所有 Intel FPGA IP 核的一般信息，包括参数化、生成、更新和仿真 IP 核。
- [Generating a Combined Simulator Setup Script](#)
创建无需对软件进行手动更新和不需要 IP 版本升级的仿真脚本。
- [Project Management Best Practices](#)
提供关于您的工程和 IP 文件的高效管理和可移植性指南。

1.1. Avalon 属性和参数

Avalon 接口通过属性描述其行为。每种接口类型的规范定义了所有接口属性和默认值。例如，Avalon -ST 接口的 maxChannel 属性可指定接口支持的通道数量。Avalon Clock 接口的 clockRate 属性提供时钟信号的频率。



1.2. 信号角色(Signal Roles)

每个 Avalon 接口定义了信号角色及其行为。很多信号角色都是可选的。您可以只选择用于实现所需功能的信号角色。例如，Avalon -MM 接口包括可选的 `beginbursttransfer` 和 `burstcount` 信号角色，用于那些支持突发(bursting)的组件。Avalon -ST 接口包括可选的 `startofpacket` 和 `endofpacket` 信号角色，用于那些支持数据包(packets)的接口。

除了 Avalon Conduit 接口，每个接口只可包括每种信号角色的一个信号。很多信号角色都支持低电平有效(active-low)信号。在本文档中，通常使用高电平有效(active-high)信号。

1.3. 接口时序(Interface Timing)

本文档的后续章节包括描述单独接口(individual interface)类型传输的时序信息。对于这些中的任何接口都没有保证的性能。实际性能取决于众多因素，包括组件设计和系统实现。

大多数 Avalon 接口不得对时钟和复位以外的信号是边沿敏感的。其他信号在稳定之前可能会跳变多次。时钟边沿之间的信号的确切时序取决于所选择的 Intel FPGA 的特性。本规范未指定电气特性。关于电气规格，请参考相应的器件文档。

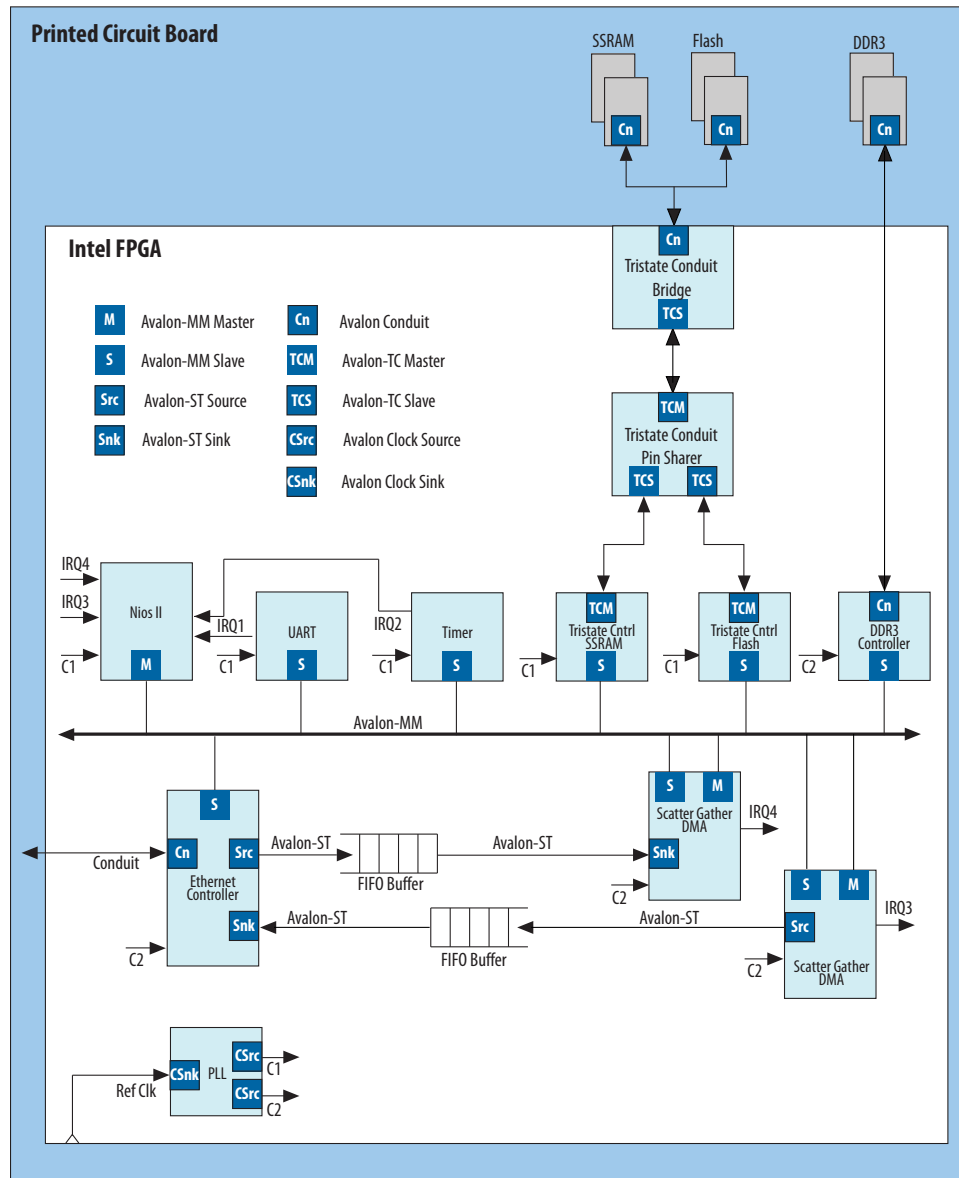
1.4. 示例：系统设计中的 Avalon 接口

在此示例中，Ethernet Controller 包含六个不同的接口类型：

- Avalon -MM
- Avalon -ST
- Avalon Conduit
- Avalon -TC
- Avalon Interrupt
- Avalon Clock

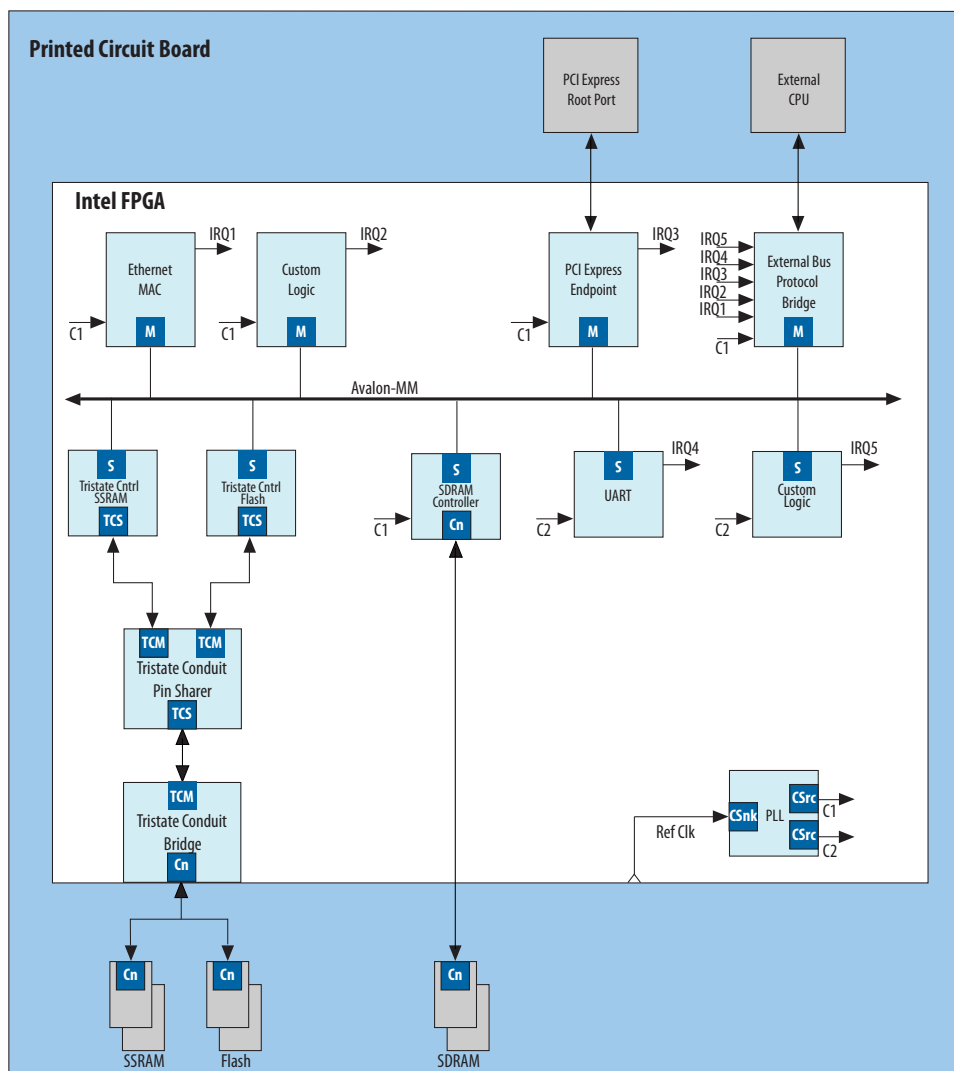
Nios® II 处理器通过 Avalon -MM 接口对片上组件的控制和状态寄存器进行访问。分散集合 DMA (scatter gather DMA)通过 Avalon -ST 接口发送和接收数据。四个组件包含由运行在 Nios II 处理器上的软件提供服务的中断接口。PLL 通过 Avalon Clock Sink 接口接收时钟并通过两个时钟源。两个组件包含 Avalon -TC 接口来访问片外存储器。最后，DDR3 控制器通过 Avalon Conduit 接口访问外部 DDR3 存储器。

图 1. 包含分散集合 DMA 控制器(Scatter Gather DMA Controller)和 Nios II 处理器的系统设计中的 Avalon 接口



在下图中，外部处理器通过与 Avalon -MM 接口的外部总线桥接来访问片上组件的控制和状态寄存器。PCI Express Root Port 通过驱动包含 Avalon -MM 主接口的片上 PCI Express Endpoint 来控制印刷电路板上的器件和 FPGA 的其他组件。外部处理器处理来自五个组件的中断。PLL 通过 Avalon Clock sink 接口接收参考时钟并提供两个时钟源。闪存和 SRAM 存储器通过 Avalon -TC 接口共享 FPGA 管脚。最后，SDRAM 控制器通过 Avalon Conduit 接口对外部 SDRAM 存储器进行访问。

图 2. 包含 PCI Express Endpoint 和外部处理器的系统设计中的 Avalon 接口

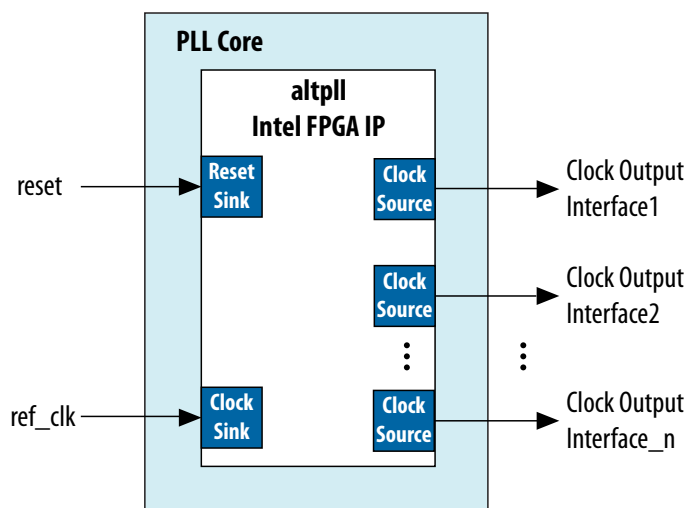


2. Avalon 时钟和复位接口

Avalon Clock 接口对一个组件使用的时钟进行定义。组件可以有时钟输入，时钟输出或两者。锁相环(PLL)是一个包括时钟输入以及时钟输出的组件的示例。

下图是一个简化图，显示了一个 PLL 组件的最重要的输入和输出。

图 3. PLL Core 时钟输出和输入



2.1. Avalon Clock Sink 信号角色

clock sink 对其他接口和内部逻辑提供时序参考。

表 1. Clock Sink 信号角色

信号角色	宽度	方向	是否需要	描述
clk	1	Input	Yes	一个时钟信号。对内部逻辑和其他接口提供同步。

2.2. Clock Sink 属性

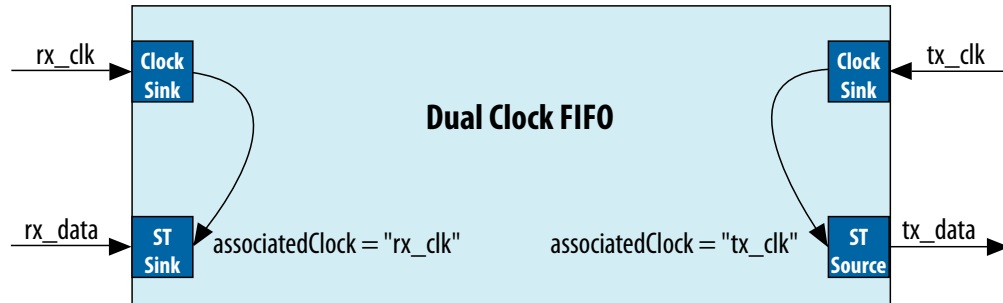
表 2. Clock Sink 属性

名称	默认值	合法值	描述
clockRate	0	$0 - 2^{32} - 1$	表示 clock sink 接口的频率(Hz)。如果为 0，那么时钟速率支持任何频率。如果为非 0，那么当连接的 clock source 不是指定的频率时 Platform Designer 会发出一个警告消息。

2.3. 相关联的时钟接口

所有同步接口都有一个 `associatedClock` 属性，该属性指定组件上的哪个时钟源(clock source)用作接口的同步参考。此属性如下图所示。

图 4. `associatedClock` 属性



2.4. Avalon Clock Source 信号角色

Avalon Clock source 接口从一个组件驱动出一个时钟信号。

表 3. `Clock Source` 信号角色

信号角色	宽度	方向	是否需要	描述
clk	1	Output	Yes	一个输出时钟信号。

2.5. Clock Source 属性

表 4. `Clock Source` 属性

名称	默认值	合法值	描述
<code>associatedDirectClock</code>	N/A	输入时钟名	直接驱动此时钟输出的时钟输入的名称(如果有)。
<code>clockRate</code>	0	$0 - 2^{32} - 1$	表示驱动时钟输出的频率(Hz)。
<code>clockRateKnown</code>	false	true, false	指示时钟频率是否已知。如果时钟频率已知，那么可以自定义系统中的其他组件。

2.6. Reset Sink

表 5. `Reset Input` 信号角色

`reset_req` 信号是一个可选的信号，通过在异步复位置位之前执行复位握手来防止存储器数据内容损坏。

信号角色	宽度	方向	是否需要	描述
<code>reset</code> , <code>reset_n</code>	1	Input	Yes	将接口或组件的内部逻辑复位成用户定义的状态。复位的同步属性由 <code>synchronousEdges</code> 参数定义。
<code>reset_req</code>	1	input	No	复位信号的早期指示。此信号用作 ROM 原语的待定复位的一个周期警告。使用 <code>reset_req</code> 来禁止时钟使能或屏蔽片上存储器的地址总线，在异步复位输入置位时防止地址转变。

2.7. Reset Sink 接口属性

表 6. Reset Input 信号角色

名称	默认值	合法值	描述
associatedClock	N/A	时钟名称	与此接口同步的时钟的名称。如果 synchronousEdges 的值为 DEASSERT 或者 BOTH，那么需要使用此属性。
synchronous-Edges	DEASSERT	NONE DEASSERT BOTH	表明复位输入所需要的同步类型。以下值定义为： <ul style="list-style-type: none"> NONE - 不需要同步，因为组件包含用于复位信号内部同步的逻辑。 DEASSERT - 复位置位是异步的，置低是同步的。 BOTH - 复位置位和置低是同步的。

2.8. 相关联的复位接口

所有同步接口都有一个 associatedReset 属性，用于指定哪个复位信号对接口逻辑进行复位。

2.9. 复位源(Reset Source)

表 7. Reset Output 信号角色

reset_req 信号是一个可选的信号，通过在异步复位置位之前执行复位握手来防止存储器数据内容损坏。

信号角色(Signal Roles)	宽度	方向	是否需要	描述
reset reset_n	1	Output	Yes	将接口或组件的内部逻辑复位成用户定义的状态。
reset_req	1	Output	Optional	使能复位请求生成，这是一个早期信号，在复位置位之前被置位。一旦置位此信号，在复位完成之前不能置低此信号。

2.10. Reset Source 接口属性

表 8. 复位接口属性

名称	默认值	合法值	描述
associatedClock	N/A	时钟名称	与此接口同步的时钟的名称。如果 synchronousEdges 的值为 DEASSERT 或者 BOTH，那么需要使用此属性。
associatedDirectReset	N/A	复位名称	复位输入的名称，此复位输入通过 one-to-one 链路直接驱动此复位源(reset source)。
associatedResetSinks	N/A	复位名称	指定复位输入，使复位源对复位进行复位。例如，reset synchronizer，执行与多个复位输入的 OR 操作来生成一个复位输出。
synchronousEdges	DEASSERT	NONE DEASSERT BOTH	表示复位输出的同步。以下值定义为： <ul style="list-style-type: none"> NONE - 复位接口是异步的。 DEASSERT - 复位置位是异步的，置低是同步的。 BOTH - 复位置位和置低是同步的。



3. Avalon 存储器映射的接口(Avalon Memory-Mapped Interface)

3.1. Avalon 存储器映射的接口简介

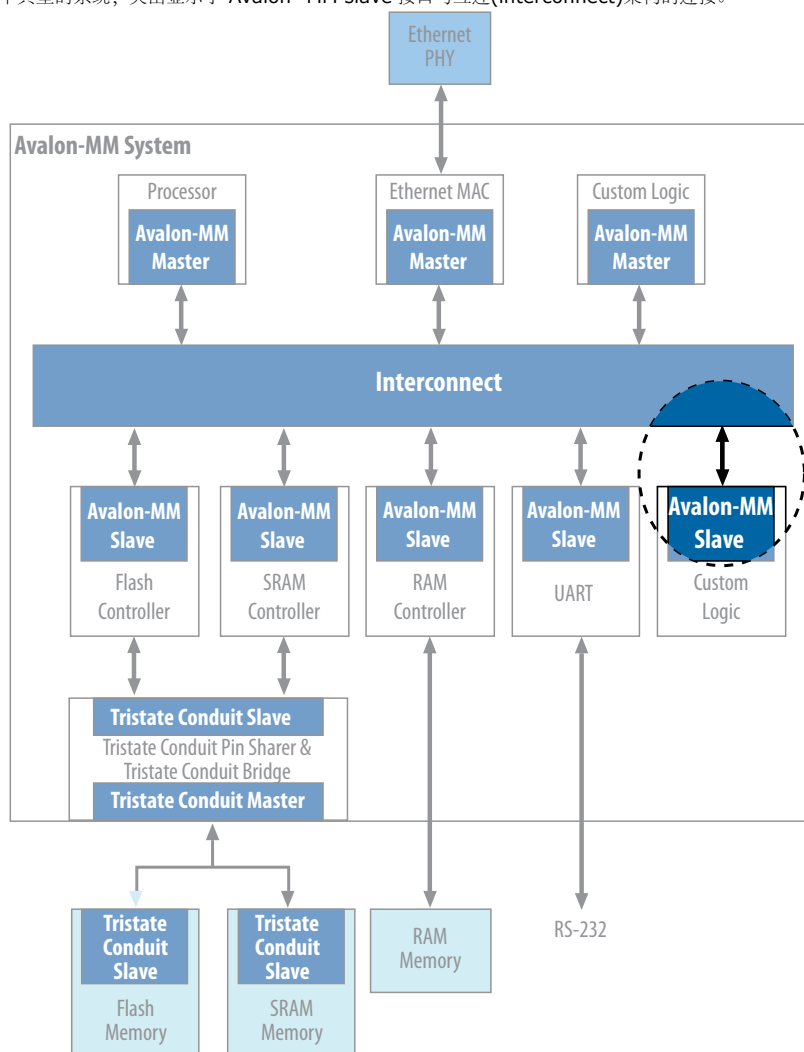
您可以使用 Avalon Memory-Mapped (Avalon -MM)接口实现主从组件的读写接口。以下是通常包含存储器映射接口的组件示例:

- 微处理器
- 存储器
- UART
- DMA
- 计时器(Timer)

Avalon -MM 接口有简单的也有有复杂的。例如, SRAM 接口有固定周期的读写传输, 具有简单的 Avalon -MM 接口。能够进行突发传输的流水线接口(pipelined interface)很复杂。

图 5. Avalon -MM Slave 传输

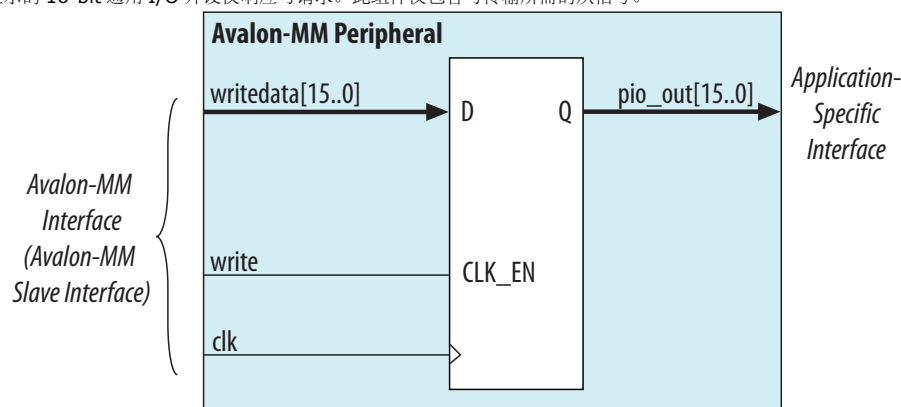
下图显示了一个典型的系统，突出显示了 Avalon -MM slave 接口与互连(interconnect)架构的连接。



Avalon -MM 组件通常仅包括组件逻辑所需的信号。

图 6. 从组件示例(Example Slave Component)

下图中显示的 16-bit 通用 I/O 外设仅响应写请求。此组件仅包含写传输所需的从信号。



Avalon -MM slave 中的每个信号对应于一个 Avalon -MM 信号角色。一个 Avalon -MM 接口仅使用每个信号角色的一个实例。

3.2. Avalon 存储器映射的接口信号角色

信号角色定义了 Avalon -MM 主端口和从端口支持的信号类型。

此规范不要求所有信号存在于一个 Avalon -MM 接口中。没有一个信号是始终需要的。对一个 Avalon -MM 接口的最低要求是用于只读接口的 `readdata`，或者用于只写接口的 `writedata` 和 `write`。

下表列出了 Avalon -MM 接口的信号角色：

表 9. Avalon -MM 信号角色

某些 Avalon -MM 信号可以是高电平有效(active high)或低电平有效(active low)。当低电平有效时，信号名称以 `_n` 结尾。

信号角色(Signal Roles)	宽度	方向	是否需要	描述
基本信号				
address	1 - 64	Master → Slave	No	<p>Masters: 默认情况下，address 信号代表一个字节地址。地址的值必须与数据宽度对齐。要写入一个数据字中的特定字节，master 必须使用 <code>byteenable</code> 信号。请参考 <code>addressUnits</code> 接口属性来了解字寻址。</p> <p>Slaves: 默认情况下，互连(interconnect)将字节地址转换成 slave 的地址空间中的一个字地址。从 slave 的角度来看，每个 slave 访问都是针对一个数据字。</p> <p>例如，<code>address = 0</code> 选择 slave 的第一个字。<code>address = 1</code> 选择 slave 的第二个字。请参考 <code>addressUnits</code> 接口属性来了解字节寻址。</p>
byteenable byteenable_n	2, 4, 8, 16, 32, 64, 128	Master → Slave	No	<p>使能在宽度大于 8 比特的接口上进行传输过程中一个或者多个特定字节通道。byteenable 中的每个比特对应于 writedata 和 readdata 中的一个字节。byteenable 的 master bit <n> 表明是否写入 byte <n>。写操作期间，byteenables 指定写入哪些字节。其他字节应该被 slave 忽略。读操作期间，byteenables 表明 master 读取哪些字节。只是返回 readdata 而无副作用的 slave 可以在读操作期间忽略 byteenables。如果一个接口没有 byteenable 信号，那么传输继续，就好像所有 byteenables 都被置位一样。</p> <p>当置位 byteenable 信号的一个以上比特时，所有置位的通道都是毗邻的。</p>

继续...



3. Avalon 存储器映射的接口(Avalon Memory-Mapped Interface)

MNL-AVABUSREF | 2018.09.26

信号角色(Signal Roles)	宽度	方向	是否需要	描述
debugaccess	1	Master → Slave	No	置位时, 允许 Nios II 处理器对配置成 ROM 的片上存储器进行写操作。
read read_n	1	Master → Slave	No	置位表明一个 read 传输。如果存在, 则需要 readdata。
readdata	8, 16, 32, 64, 128, 256, 512, 1024	Slave → Master	No	readdata 从 slave 驱动到 master, 以响应 read 传输。对那些支持读操作的接口是需要的。
response [1:0]	2	Slave → Master	No	<p>response 信号一个承载响应状态的可选信号。</p> <p>注意: 由于信号是共享的, 因此接口不能在同一时钟周期内发出或接受写响应和读响应。</p> <ul style="list-style-type: none">00: OKAY—对一个传输的成功响应。01: RESERVED—编码被保留。10: SLAVEERROR—来自 endpoint slave 的错误。表明一个失败的传输。11: DECODEERROR—表明对一个未定义位置的尝试访问。 <p>对于读响应:</p> <ul style="list-style-type: none">一个响应是通过每个 readdata 发送的。值为 N 的读突发长度会导致 N 个响应。即便在出现错误的情况下, 更少的响应也是无效的。对于突发中的每个 readdata, 响应信号值可以是不同的。接口必须要有读控制信号。通过使用 readdatavalid 信号可支持流水线(pipeline)。在读错误上, 对应的 readdata 是"don't care"。 <p>对于写响应:</p> <ul style="list-style-type: none">必须对每个写命令发送一个写响应。一个写突发仅产生一个响应, 必须在接受突发中的最后写传输之后发送。如果存在 writeresponsevalid, 那么必须使用写响应完成所有写命令。
write write_n	1	Master → Slave	No	置位表明一个 write 传输。如果存在, 则需要 writedata。
writedata	8, 16, 32, 64, 128, 256, 512, 1024	Master → Slave	No	写传输的数据。宽度必须与 readdata 的宽度相同(如果两者都存在)。对那些支持写操作的接口是必需的。
等待状态信号(Wait-State Signals)				
lock	1	Master → Slave	No	<p>lock 确保一旦 master 赢得仲裁, 此 master 就会对多个传输维持对 slave 的访问。Lock 与传输的锁定序列的第一个 read 或 write 同时置位。Lock 在传输的锁定序列的最后传输时置低。lock 的置位并不保证赢得仲裁。lock-asserting master 被授予之后, 此 master 保留授权, 直到 lock 置低。</p> <p>具备 lock 的 master 不能是一个 burst master。lock-equipped master 的仲裁优先级值将被忽略。</p> <p>lock 对于 read-modify-write (RMW)操作是特别有用的。典型的 read-modify-write 操作包括以下几个步骤:</p> <ol style="list-style-type: none">1. Master A 置位 lock, 并读取具有多个比特域的 32 比特数据。2. Master A 置低 lock, 修改一个比特域, 并写回 32 比特数据。 <p>lock 阻止 master B 在 master A 的读与写之间执行写操作。</p>
继续...				

3. Avalon 存储器映射的接口(Avalon Memory-Mapped Interface)

MNL-AVABUSREF | 2018.09.26



信号角色(Signal Roles)	宽度	方向	是否需要	描述
waitrequest waitrequest_n	1	Slave → Master	No	<p>当 slave 无法响应 read 或者 write 请求时会置位 waitrequest。强制 master 等待，直到互连(interconnect)准备好继续传输。在所有传输开始时，master 启动传输并等待，直到 waitrequest 置低。当 master 处于空闲状态时一定不要对 waitrequest 的置位状态做出假设：waitrequest 可能是高电平或者低电平，这取决于系统属性。</p> <p>当 waitrequest 置位时，除了 beginbursttransfer，对 slave 的 master 控制信号必须保持不变。关于显示 beginbursttransfer 信号的时序图，请参考 <i>Read Bursts</i>。</p> <p>Avalon -MM slave 可以在空闲周期内置位 waitrequest。Avalon -MM master 可以在 waitrequest 置位时启动一个传输，并等待此信号置低。为避免系统锁定，从器件(slave device)在复位状态时应该置位 waitrequest。</p>
流水线信号(Pipeline Signals)				
readdatavalid readdatavalid_n	1	Slave → Master	No	<p>用于可变延迟，流水线化的 read 传输。置位时表明 readdata 信号包有效数据。对于一个 burstcount 值为 <n> 的读突发，readdatavalid 信号必须置位 <n> 次，每个 readdata item 一次。在 read 的接受与 readdatavalid 的置位之间必须至少有一个周期的延迟。关于显示 readdatavalid 信号的时序图，请参考 <i>Pipelined Read Transfer with Variable Latency</i>。</p> <p>slave 可以通过置位 readdatavalid 将数据传输到 master，而不用考虑 slave 是否使用 waitrequest 来停止一个新命令。</p> <p>如果 master 支持流水线化读取(pipelined read)，则需要此信号。具有读功能的突发 master 必须包含 readdatavalid 信号。</p>
writeresponsevalid	1	Slave → Master	No	<p>可选信号。如果存在，那么接口对写命令发出写响应。置位时，响应信号上的值是一个有效的写响应。Writeresponsevalid 仅在接受写命令后的一个时钟周期或更多周期后置位。从命令接受到 writeresponsevalid 的置位至少有一个时钟周期延迟。</p>
突发信号(Burst Signals)				
burstcount	1 - 11	Master → Slave	No	<p>由突发 master 使用以指示每个突发中的传输数量。最大 burstcount 参数的值必须是 2 的幂。一个宽度为 <n> 的 burstcount 接口能够解码一个尺寸为 $2^{(\<n>-1)}$ 的最大突发。例如，一个 4-bit burstcount 信号能够支持的最大突发数为 8。最小 burstcount 为 1。</p> <p>constantBurstBehavior 属性控制 burstcount 信号的时序。具有读功能的突发 master 必须包含 readdatavalid 信号。</p> <p>对于使用字节地址的 master 和 slave，对地址宽度有以下限制：</p> <div style="background-color: #f0f0f0; padding: 5px; margin: 5px 0;"> $\langle address_w \rangle \geq \langle burstcount_w \rangle + \log_2(\langle symbols_per_word_of_interface \rangle)$ </div> <p>对于使用字地址的 master 和 slave，上面的 \log_2 项被省略。</p>
beginbursttransfer	1	Interconnect → Slave	No	<p>置位一个突发的第一个周期来指示何时开始一个突发传输。此信号在一个周期后置低，而不管 waitrequest 的值如何。关于显示 beginbursttransfer 的时序图，请参考 <i>Read Bursts</i> 中的图。</p> <p>beginbursttransfer 是可选的。slave 总是能够通过计算数据传输来从内部计算下一个写突发传输的开始。</p> <p>警告: do not 使用此信号。此信号存在用于支持传统的存储控制器。</p>



3.3. 接口属性

表 10. Avalon -MM 接口属性

名称	默认值	合法值	描述
addressUnits	Master - symbols Slave - words	words, symbols	指定地址的单位。一个符号(symbol)通常是一个字节。请参考 <i>Avalon Memory-Mapped Interface Signal Types</i> 表中的 address 的定义来了解此属性的典型使用。
alwaysBurstMaxBurst	false	true, false	为 true 时表明 master 总是发出最大长度的突发。最大突发长度为 $2^{\text{burstcount_width} - 1}$ 。此参数对 Avalon -MM slave 接口无效。
burstcountUnits	words	words, symbols	此属性指定 burstcount 信号的单位。对于符号(symbol), burstcount 值被解释为突发中的符号(字节)数量。对于字(word), burstcount 值被解释为突发中的字传输的数量。
burstOnBurstBoundariesOnly	false	true, false	如果为 true, 那么在此接口上出现的突发传输开始于突发大小(单位为字节)的倍数的地址。
constantBurstBehavior	Master - false Slave - false	true, false	Masters: 为 true 时, 声明 master 在整个突发传输中保持地址和突发计数(burstcount)常量。如果为 false (默认值), 那么声明 master 仅为突发的第一个 beat 保持地址和突发计数常量。 Slaves: 为 true 时, 声明 slave 期望地址和 burstcount 在整个突发中保持不变。如果为 false (默认值), 那么声明 slave 仅在突发的第一个 beat 上对地址和突发计数进行采样。
holdTime(1)	0	0 - 1000 cycles	指定 write 的置低与 address 和 data 的置低之间的时间(timingUnits)。(仅应用于写传输)
linewrapBursts	false	true, false	某些存储器件实现封装突发(wrapping burst), 而不是递增突发(incrementing burst)。当一个封装突发到达突发边界时, 地址回装到之前的突发边界。地址计数只需要低位比特(low-order bits)。例如, 在 32-bit 接口上每隔 32 个字节的突发边界对地址 0xC 的一个封装突发写入到以下地址: <ul style="list-style-type: none"> • 0xC • 0x10 • 0x14 • 0x18 • 0x1C • 0x0 • 0x4 • 0x8
maximumPendingReadTransactions (1)	1(2)	1 - 64	Slaves: 此参数是 slave 能够排序的最大数量的待读取操作。对于任何具有 readdatavalid 信号的 slave, 此值必须非零。 请参考 <i>Pipelined Read Transfer with Variable Latency</i> 来了解显示此属性的时序图和关于使用 waitrequest 和 readdatavalid 及多个未完成的读取操作的附加信息。 Masters: 此属性是 master 能够生成的最大数量的未完成的读传输(outstanding read transactions)。 注 请不要将此参数设为 0。(为向后兼容, 软件支持值为 0 的参数设置。然而, 不应该在新的设计中使用值为 0 的参数设置)。
maximumPendingWriteTransactions	0	1 - 64	slave 能够接受或者 master 能够发出的最大数量的待发布写操作(pending non-posted writes)。一旦互连(interconnect)达到此限制, 并且 master 停止发出命令, slave 就会置位 waitrequest。

继续...

3. Avalon 存储器映射的接口(Avalon Memory-Mapped Interface)

MNL-AVABUSREF | 2018.09.26



名称	默认值	合法值	描述
			默认值为 0，对一个支持写响应的 master 允许无限制的待写传输。支持写响应的 slave 必须将其设为非零值。
minimumResponseLatency	1		对于支持 readdatavalid 或者 writeresponsevalid 的接口，在读或写命令与对命令的响应之间指定最小数量的周期。
readLatency(1)	0	0 - 63	固定延迟 Avalon -MM slave 的读延迟。关于使用固定延迟读取的时序图，请参考 <i>Pipelined Read Transfers with Fixed Latency</i> 。 固定延迟的 Avalon -MM slave 必须对此接口属性提供一个值。可变延迟的 Avalon -MM slave 使用 readdatavalid 信号来指定有效数据。
readWaitTime(1)	1	0 - 1000 cycles	适用于不使用 waitrequest 信号的接口。readWaitTime 在 slave 接受一个读命令前以 timingUnits 表示时序。此时序就好像 slave 对 waitrequest 置位了 readWaitTime 周期。
setupTime(1)	0	0 - 1000 cycles	指定 address 和 data 的置位与 read 或 write 的置位之间的时间(timingUnits)。
timingUnits(1)	cycles	cycles, nanosecond s	指定 setupTime, holdTime, writeWaitTime 和 readWaitTime 的单位。同步器件使用周期(cycles)为单位，异步器件使用纳秒(nanoseconds)为单位。差不多所有的 Avalon -MM slave 器件都是同步的。 一个桥接 Avalon -MM slave 接口和片外器件的 Avalon -MM 组件可以是异步的。此片外器件可能有一个总线转向(bus turnaround)的固定建立时间(fixed settling time)。
waitrequestAllowance	0		指定 waitrequest 置位后可发出或接受的传输数量。 当 waitrequestAllowance 为 0 时，write, read 和 waitrequest 信号保持其现有行为，如 <i>Avalon -MM Signal Roles</i> 表中所描述。 当 waitrequestAllowance 大于 0 时，在 write 或 read 置位的每个时钟周期都算作一个命令传输。一旦 waitrequest 置位，只有 waitrequestAllowance 更多的命令传输是合法的，同时 waitrequest 保持置位。达到 waitrequestAllowance 后，只要 waitrequest 置位，write 和 read 就必须保持置低。 一旦 waitrequest 置低，在无任何限制的情况下，传输可随时继续，直到 waitrequest 再次置位。此时，waitrequestAllowance 更多的传输可能完成，而 waitrequest 仍保持置位。
writeWaitTime(1)	0	0 - 1000 Cycles	适用于不使用 waitrequest 信号的接口，writeWaitTime 指定 slave 接受写操作前的时间 (timingUnits)。此时序就好像 slave 对 waitrequest 置位 writeWaitTime 周期或者纳秒。 关于使用 writeWaitTime 的时序图，请参考 <i>Read and Write Transfers with Fixed Wait-States</i> 。
接口关系属性			
associatedClock	N/A	N/A	与此 Avalon -MM 接口同步的时钟接口的名称。
继续...			



名称	默认值	合法值	描述
associatedReset	N/A	N/A	对此 Avalon -MM 接口上的逻辑进行复位的复位接口的名称。
bridgesToMaster	0	Avalon -MM Master name on the same component	一个 Avalon -MM 桥接由一个 slave 和一个 master 组成，并具有以下属性：对请求一个或多个字节的 slave 的访问会导致 master 请求相同的字节。Platform Designer 组件库中的 Avalon -MM Pipeline Bridge 实现此功能。
注释： <ol style="list-style-type: none">尽管此属性是 slave 器件的一个特征，但 master 也可以声明此属性来使能匹配 master 和 slave 接口之间的直接连接。如果 slave 接口接受的读传输数量多于所允许的数量，那么互连待读 FIFO 可能会上溢，并产生不可预测的结果。slave 可能会失去 readdata 或者将 readdata 路由到错误的 master 接口。或者，系统可能会锁定。slave 接口必须置位 waitrequest 来防止出现上溢。			

相关链接

- [Avalon 存储器映射的接口信号角色](#) (第 13 页)
- [读和写响应](#) (第 29 页)
- [可变延迟的流水线读传输\(Pipelined Read Transfer with Variable Latency\)](#) (第 24 页)
- [固定延迟的流水线读传输\(Pipelined Read Transfer with Fixed Latency\)](#) (第 25 页)
- [Read and Write Responses](#)

In Platform Designer User Guide: Intel Quartus® Prime Pro Edition

3.4. 时序(timing)

Avalon -MM 接口是同步的。每个 Avalon -MM 接口都与一个相关联的时钟接口同步。如果信号是从与时钟信号同步的寄存器的输出驱动的，那么这些信号可以是组合的。此规范没有规定信号在时钟边沿之间如何或何时跳变。时序图没有显示细粒度的时序信息。

3.5. 传输(transfer)

本节在介绍传输类型之前定义了两个基本概念：

- **传输(Transfer)**—传输是一个字或者一个或多个符号的数据的读或写操作。传输发生在 Avalon -MM 接口与互连之间。传输需要一个或多个时钟周期才能完成。

master 和 slave 都是传输的一部分。Avalon -MM master 启动传输，Avalon -MM slave 作出响应。

- **Master-slave pair**—此术语是指在一个传输中涉及到 master 接口和 slave 接口。在一个传输过程中，master 接口控制和数据信号遍历互连架构，并与 slave 接口进行交互。

3.5.1. 典型的读传输和写传输

本节介绍了一个典型的 Avalon -MM 接口，通过 slave 控制的 waitrequest 来支持读写传输。通过置位 waitrequest 信号，slave 能够将互连暂停所需的周期。如果一个 slave 将 waitrequest 用于读传输或者写传输，那么该 slave 必须对两者都使用 waitrequest。

slave 通常在时钟的上升沿之后接收 address, byteenable, read 或者 write 和 writedata。slave 通过在上升时钟沿之前置位 waitrequest 来暂停传输。当 slave 置位 waitrequest 时，传输延迟。waitrequest 置位期间，地址和其他控制信号保持不变。slave 接口置低 waitrequest 之后，在第一个 clk 的上升沿完成传输。

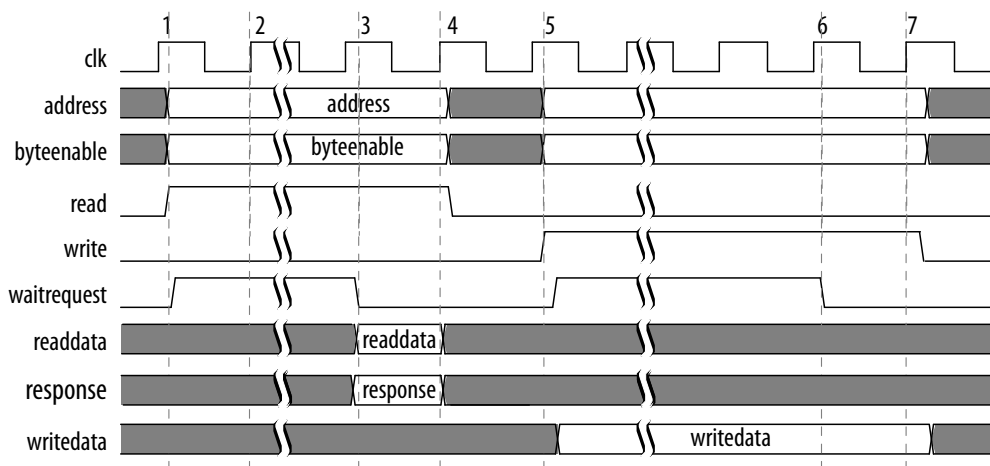


对 slave 接口暂停的时间长短没有限制。因此，您必须确保 slave 接口不要无限期地置位 waitrequest。下图显示了使用 waitrequest 的 read 和 write 传输。

注意:

waitrequest 能够从 read 和 write 请求信号中解耦。waitrequest 可以在空闲周期置位。当 waitrequest 置位时，Avalon -MM master 可以启动一个传输，并等待此信号置低。将 waitrequest 从 read 和 write 进行解耦可以提高系统时序。解耦消除了包含 read, write 和 waitrequest 信号的组合环路(combination loop)。如果要求更多的解耦，那么使用 waitrequestAllowance 属性。从 Quartus® Prime Pro v17.1 Stratix® 10 ES Editions 发布开始就包含了 waitrequestAllowance 属性。

图 7. 包括 Waitrequest 的读和写传输



此时序图中的数字标记以下转换:

1. address, byteenable 和 read 在 clk 的上升沿后置位。slave 置位 waitrequest, 暂停传输。
2. waitrequest 被采样。由于 waitrequest 置位，因此周期变成一个等待状态(wait-state)。address, read, write 和 byteenable 保持不变。
3. slave 在 clk 的上升沿之后置低 waitrequest。slave 置位 readdata 和 response。
4. master 对 readdata, response 和置低的 waitrequest(完成传输)的进行采样。
5. address, writedata, byteenable 和 write 信号在 clk 的上升沿之后置位。slave 置位 waitrequest, 暂停传输。
6. slave 在 clk 的上升沿之后置低 waitrequest。
7. slave 采集结束传输的写数据。

3.5.2. 使用 waitrequestAllowance 属性进行传输

waitrequestAllowance 属性指定 waitrequest 信号置位后一个 Avalon -MM master 能够发出的或者一个 Avalon -MM slave 必须接受的传输数量。从 Intel Quartus Prime 17.1 软件发布开始就包含了 waitrequestAllowance 属性。

waitrequestAllowance 的默认值为 0，对应于 *Typical Read and Write Transfers* 中描述的行为，其中 waitrequest 置位停止发出或接受当前传输。

一个 waitrequestAllowance 大于 0 的 Avalon -MM，当它的内部缓存在变满前仅能接受 waitrequestAllowance 更多的条目(entries)时通常会置位 waitrequest。
waitrequestAllowance 大于 0 的 Avalon -MM master 有 waitrequestAllowance 额外的周期来停止发送传输，从而支持 master 逻辑中更多的 pipelining。当 waitrequestallowance 用完时，master 必须置低 read 或者 write 信号。

置大于 0 的 waitrequestAllowance 支持高速设计，在此设计中背压的立即形成可能会导致最大操作频率(F_{MAX})的下降，这通常是由控制路径中的组合逻辑导致的。Avalon -MM slave 必须支持对其 waitrequestAllowance 值合法的所有可能的传输时序。例如，一个 waitrequestAllowance = 2 的 slave 必须能够接受下例中显示的任何 master 传输波形。

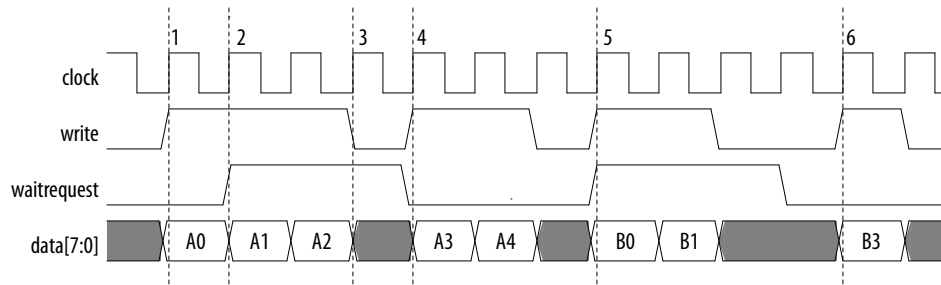
相关链接

典型的读传输和写传输 (第 18 页)

3.5.2.1. waitrequestAllowance 等于二

下图显示了一个 Avalon -MM> master 的时序，在 Avalon -MM> slave 置低或者置位 waitrequest 之后，该 Avalon -MM> master 有两个时钟周期来开始和停止发送传输。

图 8. Master write: waitrequestAllowance 等于两个时钟周期



此图中的标记标识了以下事件：

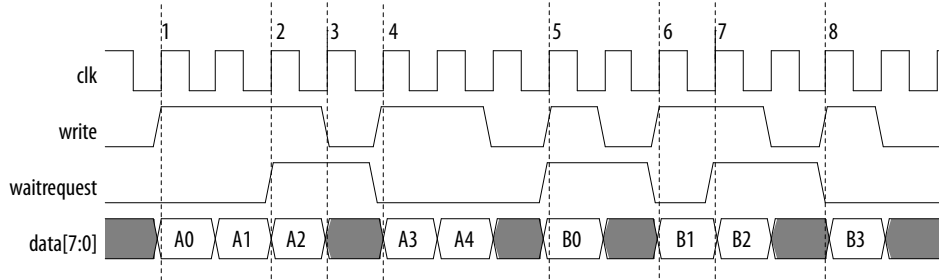
当waitrequestAllowance为0时，一旦waitrequest拉高，就不能再继续传输数据了。

1. Avalon -MM> master 驱动 write 和 data。
2. Avalon -MM> slave 置位 waitrequest。由于 waitrequestAllowance 为 2，因此 master 能够完成 2 个额外的数据传输。
3. master 按照要求置低 write，因为 slave 置位 waitrequest 第三个周期。
4. Avalon -MM> master 驱动 write 和 data。slave 没有置位 waitrequest。写操作完成。
5. 即便 slave 置位 waitrequest，Avalon master 也驱动 write 和 data。由于 waitrequestAllowance 是 2 个周期，因此写操作完成。
6. Avalon master 驱动 write 和 data。slave 没有置位 waitrequest。写操作完成。

3.5.2.2. waitrequestAllowance 等于一

下图显示了一个 Avalon -MM master 的时序，在 Avalon -MM slave 置低或者置位 waitrequest 之后，该 Avalon -MM master 有一个时钟周期来开始和停止发送传输。

图 9. Master Write: waitrequestAllowance 等于一个时钟周期



此图中的编号标识了以下事件：

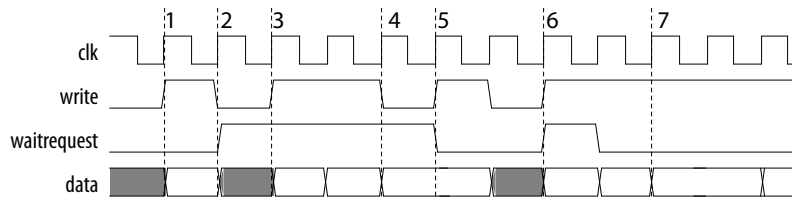
1. Avalon -MM master 驱动 write 和 data。
2. Avalon -MM slave 置位 waitrequest。由于 waitrequestAllowance 为 1，因此 master 能够写操作。
3. master 置低 write，因为 slave 置位 waitrequest 第二个周期。
4. Avalon -MM master 驱动 write 和 data。slave 没有置位 waitrequest。写操作完成。
5. slave 置位 waitrequest。由于 waitrequestAllowance 为 1 个周期，因此写操作完成。
6. Avalon -MM master 驱动 write 和 data。slave 没有置位 waitrequest。写操作完成。
7. Avalon -MM slave 置位 waitrequest。由于 waitrequestAllowance 为 1，因此 master 能够完成 1 个额外的数据传输。
8. Avalon master 驱动 write 和 data。slave 没有置位 waitrequest。写操作完成。

3.5.2.3. waitrequestAllowance 等于二-不推荐

下图显示了一个 Avalon -MM master 在 waitrequest 置位后能够发送两个传输的时序。

此时序是合法的，但不推荐。在此示例中，master 计算传输的数量，而不是计算时钟周期数。这种方法需要一个计数器，使实现变得更复杂，并可能影响时序收敛。当 master 确定何时通过 waitrequest 信号和恒定周期数来驱动传输时，master 会根据寄存的信号启动或停止传输。

图 10. waitrequestAllowance 等于二个传输



此图中的编号标识了以下事件：

1. Avalon -MM> master 置位 write 并驱动 data。
2. Avalon -MM> slave 置位 waitrequest。
3. Avalon -MM> master 驱动 write 和 data。由于 waitrequestAllowance 为 2，master 在 2 个连续周期内驱动数据。
4. Avalon -MM> master 置低 write，因为 master 已经使用了 2-transfer waitrequestAllowance。
5. waitrequest 一置低，Avalon -MM> master 就发出一个写操作。
6. Avalon -MM> master 驱动 write 和 data。slave 置位 waitrequest 一个周期。
7. 作为 waitrequest 的响应，Avalon -MM> master 将数据保持 2 个周期。

3.5.2.4. Avalon -MM Master 和 Slave 接口的 waitrequestAllowance 兼容性

支持 waitrequest 信号的 Avalon -MM master 和 slave 支持背压(backpressure)。有背压的 master 总是能够连接到没有背压的 slave。没有背压的 master 不能连接到有背压的 slave。

表 11. Avalon-MM Master 和 Slave 的 waitrequestAllowance 兼容性

Master and Slave waitrequestAllowance	兼容性
master = 0 slave = 0	遵循与标准 Avalon-MM 接口相同的兼容性规则。
master = 0 slave > 0	无法直接连接。 对于具有 waitrequest 信号的 master 的情况，需要一个简单的适配(adaptation)。如果 master 不支持 waitrequest 信号，那么就无法进行连接。
master > 0 slave = 0	无法直接连接。 当与一个具有 waitrequest 信号或者固定等待状态的 slave 连接时，适配(缓存)是需要的。
master > 0 slave > 0	如果 master' s allowance <= slave' s allowance，那么就不需要适配(adaptation)。 如果 master allowance < slave allowance，那么可以插入流水线寄存器(pipeline registers)。 对于点到点连接，您可以在命令信号或者 waitrequest 信号上添加流水线寄存器。最多可插入<d>个寄存器阶段(register stages)，其中<d>是限额(allowances)之间的差异。 连接到一个具有比 slave 所需缓冲更高的 waitrequestAllowance 的 master。

3.5.2.5. waitrequestAllowance 错误条件

如果一个 Avalon -MM 接口违反了 waitrequest 限额规范，那么行为是不可预测的。

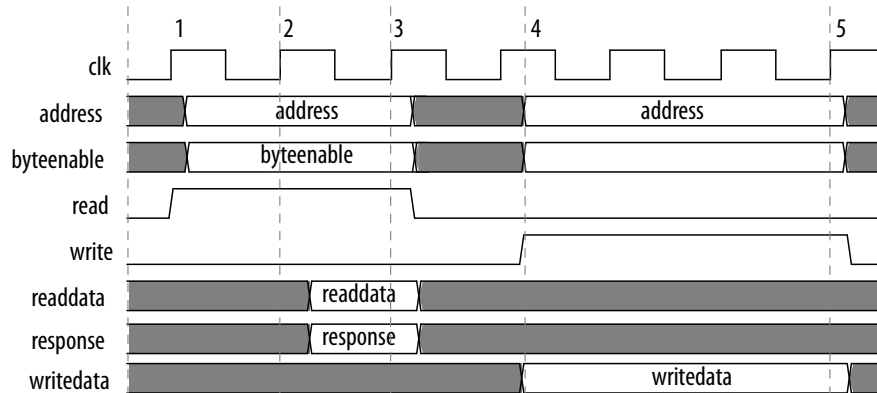
- 如果一个 master 违反了 waitrequestAllowance = <n> 规范，发送大于<n>的传输，那么传输可能会被取消，或者出现数据损坏。
- 如果一个 slave 显示一个比可能值更大的 waitrequestAllowance，那么一些传输可能会被取消，或者发生数据损坏。

3.5.3. 固定等待状态的读和写传输(Read and Write Transfers with Fixed Wait-States)

slave 可以使用 readWaitTime 和 writeWaitTime 属性来指定固定等待状态。使用固定等待状态是使用 waitrequest 来停止传输的一个替代方法。地址和控制信号 byteenable, read 和 write)在传输期间保持不变。将 readWaitTime 或 writeWaitTime 设置为<n>相当于将 waitrequest 置位<n>个周期每传输。

在下图中, slave 有 writeWaitTime = 2 和 readWaitTime = 1。

图 11. slave 接口上固定等待状态的读和写传输(Read and Write Transfer with Fixed Wait-States at the Slave Interface)



此图中的编号标识了以下转换:

1. master 在 clk 的上升沿置位 address 和 read。
2. clk 的下一个上升沿标记第一个也是唯一一个等待状态(wait-state)周期的结束。readWaitTime 为 1。
3. slave 在 clk 的上升沿置位 readdata 和 response。读传输结束。
4. writedata, address, byteenable 和 write 信号可用于 slave。
5. 写传输在 2 个等待状态(wait-state)周期后结束。

单一等待状态的传输通常用于多周期片外(off-chip)外设。 外设 在 clk 的上升沿采集地址和控制信号。外设有一个完整的周期来返回数据。

零等待状态的组件是允许的。然而, 零等待状态的组件可能会降低可实现的频率。零等待状态要求组件在出现请求的同一周期中生成响应。

3.5.4. 流水线传输(Pipelined Transfers)

Avalon -MM 流水线读传输增加同步从器件的吞吐量, 这些同步从器件需要几个周期才能返回第一次访问的数据。这样的器件通常可以在一段时间后每个周期返回一个数据值。新流水线读传输可以在先前传输的 readdata 返回之前启动。

流水线读传输有一个地址阶段(address phase)和一个数据阶段(data phase)。 master 通过在地 址阶段期间提供地址来启动一个传输。slave 通过在数据阶段期间传送数据来完成传输。一个新传输 (或者多个传输)的地址阶段可以在先前传输的数据阶段完成之前开始。此延迟称为流水线延迟 (pipeline latency)。流水线延迟是从地址阶段结束到数据阶段开始的持续时间。

等待状态和流水线延迟的传输时序有以下主要差异：

- 等待状态(wait-states)——等待状态决定地址阶段的长度。等待状态限制一个端口的最大吞吐量。如果 slave 需要一个等待状态来响应一个传输请求，那么端口需要每个传输两个时钟周期。
- 流水线延迟(Pipeline Latency)——流水线延迟决定独立于地址阶段数据返回的时间。无等待状态的流水线 slave 可以维持每个周期一个传输。但是，slave 可能需要几个周期的延迟才能返回数据的第一个单元。

可以同时支持等待状态和流水线读取。流水线延迟可以是固定的也可以是可变的。

3.5.4.1. 可变延迟的流水线读传输(Pipelined Read Transfer with Variable Latency)

采集地址和控制信号后，Avalon-MM 流水线 slave 需要一个或多个周期来生成数据。流水线可以在任何给定时间有多个待定读传输。

可变延迟流水线读传输：

- 需要一个额外信号，readdatavalid，当读数据有效时指示。
- 包括与非流水线读传输相同的一组信号。

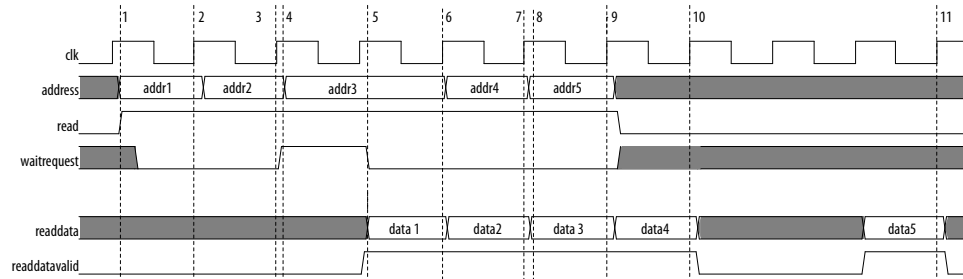
在可变延迟流水线读传输中，使用 readdatavalid 的 slave 外设被认为是通过可变延迟进行流水线化的。对应于一个读命令的 readdata 和 readdatavalid 信号最早可以在此读命令置位后的周期中置位。

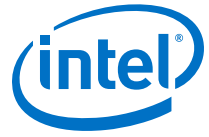
slave 必须按照接受读命令的相同顺序返回 readdata。具有可变延迟的流水线 slave 端口必须使用 waitrequest。slave 可以通过置位 waitrequest 来停止传输，以保持一个可接受数量的待定传输。slave 可以通过置位 readdatavalid 将数据传输到 master，而不用考虑 slave 是否通过 waitrequest 来暂停一个新命令。

注意：待定传输的最大数量是从接口(slave interface)的一个属性。互连架构使用此数量来构建逻辑，将 readdata 布线到请求的 master。从接口(而非互连架构)必须跟踪待定读取的数量。slave 必须置位 waitrequest 以防止待定读取的数量超过最大数量。如果 slave 有 waitrequestAllowance > 0，那么 slave 必须尽早地置位 waitrequest，以便待定传输总数(包括在 waitrequest 置位期间接受的那些数量)不超过指定的待定传输的最大数量。

图 12. 可变延迟的流水线读传输(Pipelined Read Transfers with Variable Latency)

下图显示了几个 slave 读传输。通过可变延迟对 slave 进行流水线化。在此图中，slave 最多可接受两个待定传输。slave 使用 waitrequest 来避免超过这个最大值。





此图中的编号标识了以下转换：

1. master 置位 address 和 read，启动一个读传输。
2. slave 采集 addr1。
3. slave 采集 addr2。
4. 由于 slave 已经接受 2 个最大数量的待定读取，因此 slave 置位 waitrequest，从而导致第三个传输停止。
5. slave 置位 data1，即对 addr1 的响应。slave 置低 waitrequest。
6. slave 采集 addr3。interconnect 采集 data1。
7. slave 采集 addr4。interconnect 采集 data2。
8. slave 驱动 readdatavalid 和 readdata 以响应第三次读传输。
9. slave 采集 addr5。interconnect 采集 data3。read 信号被置低。waitrequest 的值不再是相关的。
10. interconnect 采集 data4。
11. slave 驱动 data5，并置位 readdatavalid，完成最后待定读传输的数据阶段。

如果 slave 在处理待定读传输时无法处理写传输，那么 slave 必须置位 waitrequest，并且停止写操作，直到完成待定读传输。在 slave 接受对与当前待定读传输相同地址的写传输的情况下，Avalon-MM 规范没有定义 readdata 的值。

3.5.4.2. 固定延迟的流水线读传输(Pipelined Read Transfer with Fixed Latency)

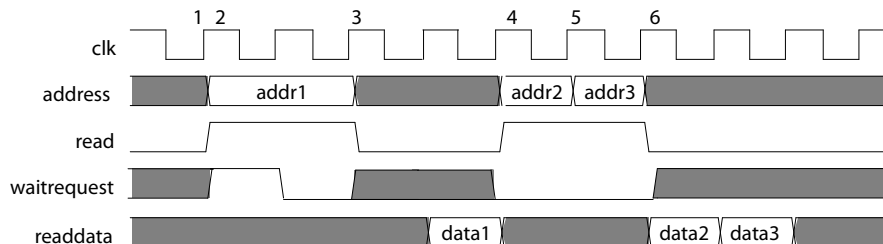
固定延迟读传输的地址阶段与可变延迟情况相同。在地址阶段之后，具有固定读延迟的流水线 slave 需要固定数量的时钟周期才能返回有效的 readdata。readWaitTime 属性指定返回有效 readdata 的时钟周期数。interconnect 在相应的上升时钟边沿采集 readdata，结束数据阶段。

在地址阶段，slave 通过置位 waitrequest 来阻止传输。或者，slave 对固定数量的等待状态指定 readWaitTime。在等待状态(如果有)之后，地址阶段在 clk 的下一个上升沿结束。

在数据阶段，slave 在一个固定延迟后驱动 readdata。对于一个值为 $<n>$ 的读延迟，slave 在地址阶段结束后必须在 clk 的 $<nth>$ 上升沿呈现有效的 readdata。

图 13. 具有两个周期的固定延迟的流水线读传输(Pipelined Read Transfer with Fixed Latency of Two Cycles)

下图显示了一个 master 与一个 pipelined slave 之间的多个数据传输。slave 驱动 waitrequest 来停止传输，并且有一个 2 个周期的固定读延迟。



此时序图中的编号标识了以下转换：

1. master 通过置位 read 和 addr1 启动一个读传输。
2. slave 置位 waitrequest 以使传输暂停一个周期。
3. slave 在 clk 的上升沿采集 addr1。地址阶段在此结束。
4. slave 在 2 个周期后呈现有效的 readdata，传输结束。
5. 对一个新的读传输置位 addr2 和 read。
6. 在返回先前传输的数据之前，master 在下一个周期内启动第三次读传输。

3.5.5. 突发传输(Burst Transfers)

突发(burst)将多个传输作为一个单元进行执行，而不是独立地处理每个字。突发可以增加 slave 端口的吞吐量，从而在一次处理多个字时实现更高的效率，例如 SDRAM。突发的净效应是锁定突发持续时间的仲裁。支持读写操作的突发 Avalon -MM 接口一定支持读写突发。

突发 Avalon -MM 接口包含一个 burstcount 输出信号。如果 slave 有一个 burstcount 输入，那么 slave 具有突发能力。

burstcount 信号的行为如下：

- 在突发的开始，burstcount 表示突发中有序传输的数量。
- 对于 burstcount 的宽度 $\langle n \rangle$ ，最大突发长度为 $2^{\langle n \rangle - 1}$ 。最小法定突发长度为 1。

如要支持 slave 读突发，slave 也必须支持：

- 使用 waitrequest 信号的等待状态。
- 使用 readdatavalid 信号的可变延迟的流水线传输。

在突发的开始，slave 看到 address 和 burstcount 上的突发长度值。对于一个地址为 $\langle a \rangle$ 和 burstcount 值为 $\langle b \rangle$ 的突发，slave 必须从地址 $\langle a \rangle$ 开始执行 $\langle b \rangle$ 个连续传输。在 slave 接收(写)或返回(读) $\langle b^{\text{th}} \rangle$ 字的数据后完成突发。突发 slave 必须对每个突发采集一次 address 和 burstcount。slave 逻辑必须对突发中的所有传输(但不包括第一个传输)推断地址。slave 也可以使用输入信号 beginbursttransfer，interconnect 在每个突发的第一个周期对此信号进行置位。

3.5.5.1. 写突发(Write Bursts)

这些规则应用于一个写突发开始于大于 1 的 burstcount 的情况：

- 当值为 $\langle n \rangle$ 的 burstcount 出现在突发的开始时，slave 必须接受 $\langle n \rangle$ 个连续单元的 writedata 以完成突发。master-slave pair 之间的仲裁保持锁定，直到突发完成。此锁定保证在写突发完成之前没有其他 master 能够在 slave 上执行传输。
- slave 必须在 write 置位时才能采集 writedata。在突发期间，master 通过置低 write 来表明 writedata 是无效的。置低 write 不会终止突发。write 的置低会延迟突发，并且没有其他 master 可以访问 slave，从而降低了传输效率。
- slave 通过置位 waitrequest，强制 writedata，write，burstcount 和 byteenable 保持不变来延迟一个传输。

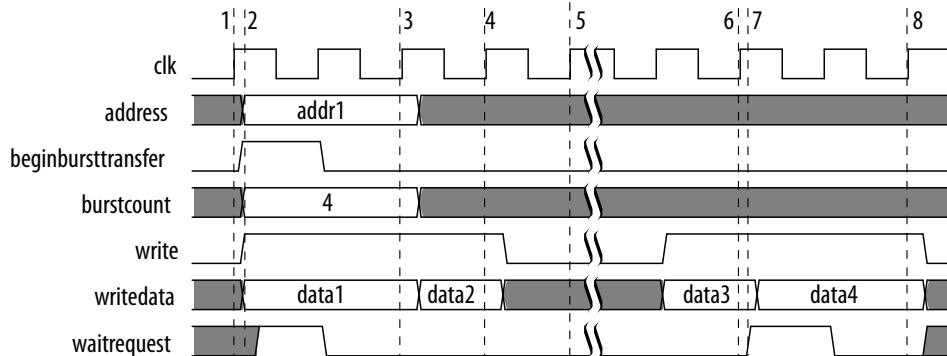


- byteenable 信号的功能对于突发和非突发 slave 是相同的。对一个 64-bit slave 的 32-bit master 突发写操作(burst-writing), 从字节地址 4 开始, slave 看到的第一个写传输在它的地址 0 上, byteenable = 8'b11110000。byteenables 可以根据突发的不同字进行改变。
- byteenable 信号不必都置位。一个写入部分字的 burst master 能够使用 byteenable 信号来识别正被写入的数据。
- Writes with 通过全部为 0 的 byteenable 信号的写操作作为有效传输传递到 Avalon-MM slave。
- constantBurstBehavior 属性指定突发信号的行为。
 - 当 constantBurstBehavior 对于一个 master 为真(true)时, 此 master 在整个突发期间保持 address 和 burstcount 稳定。当对一个 slave 为真(true)时, constantBurstBehavior 声明此 slave 期望 address 和 burstcount 在整个突发期间保持稳定。
 - 当 constantBurstBehavior 为假(false)时, master 仅对突发的第一个传输保持 address 和 burstcount 稳定。当 constantBurstBehavior 为假(false)时, slave 仅在突发的第一个传输上采集 address 和 burstcount。

图 14.

constantBurstBehavior 对 Master 和 Slave 设置为 False 的写突发

下图显示了一个长度为 4 的 slave 写突发。在此实例中, slave 置位 waitrequest 两次来延迟突发。I



此时序图中的编号标识了以下转换:

1. master 置位 address, burstcount, write 并驱动 writedata 的第一个单元。
2. slave 立即置位 waitrequest, 表明 slave 尚未准备好继续传输。
3. waitrequest 为低电平。slave 采集 addr1, burstcount 和 writedata 的第一个单元。在随后的传输周期中, address 和 burstcount 被忽略。
4. slave 在 clk 的上升沿采集数据的第二个单元。
5. write 置低期间突发暂停。
6. slave 在 clk 的上升沿采集数据的第三个单元。
7. slave 置位 waitrequest。作为响应, 所有输出在另一个时钟周期内保持不变。
8. slave 在 clk 的上升沿采集数据的最后一个单元。slave 写突发结束。

在上图中, beginbursttransfer 信号在突发的第一个时钟周期置位, 在下一个时钟周期置低。即便 slave 置位 waitrequest, beginbursttransfer 信号也仅在第一个时钟周期置位。

相关链接

接口属性 (第 16 页)

3.5.5.2. 读突发(Read Bursts)

读突发类似于具有可变延迟的流水线读传输。读突发有不同的地址和数据阶段。当 slave 呈现有效的 readdata 时 readdatavalid 指示。与流水线读传输不同，单个读突发地址会导致多个数据传输。

这些规则应用于读突发：

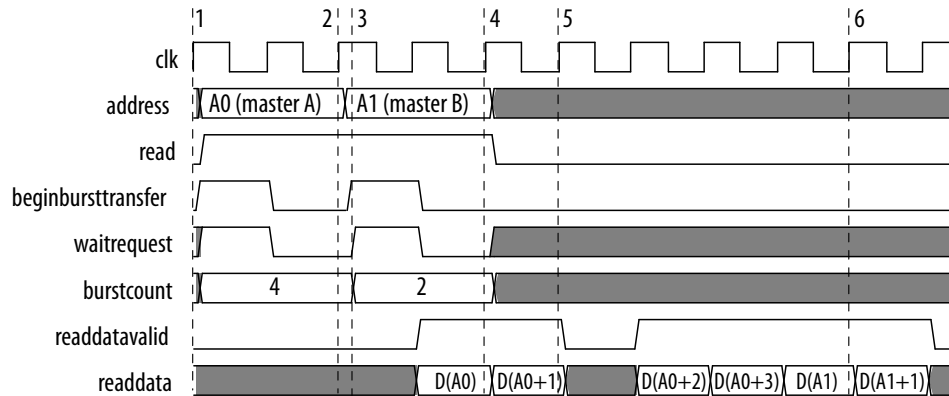
- 当一个 master 直接连接到一个 slave 时，值为 $<n>$ 的 burstcount 意味着此 slave 必须返回 $<n>$ 个字的 readdata 才能完成突发。在 interconnect 链接到 master 和 slave 对的情况下，interconnect 可以抑制从 master 发送到 slave 的读命令。例如，如果 master 发送一个读命令，其中 byteenable 的值为 0，那么 interconnect 可能会抑制此读操作。因此，slave 不会响应读命令。
- slave 通过提供 readdata 和置位 readdatavalid 一个周期来呈现每个字。readdatavalid 的置低会延迟突发数据阶段，但不会终止突发数据阶段。
- 对于 burstcount > 1 的读操作，Intel 建议置位所有 byteenables。

注意:

Intel 建议具有突发功能的 slave 没有读取副作用。(此规范不保证一个 master 从 slave 读取多少个字节才能满足一个请求。)

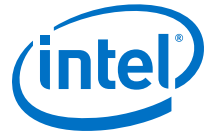
图 15. 读突发(Read Burst)

下图显示了一个系统，其中两个突发 master 访问一个 slave。注意：在数据返回给 Master A 之前，Master B 能够驱动一个读请求。



此时序图中的编号标识了以下转换：

1. Master A 在 clk 的上升沿之后置位 address (A0)，burstcount 和 read。slave 置位 waitrequest，导致除了 beginbursttransfer 之外的所有输入在另一个时钟周期内保持不变。
2. slave 在 clk 的上升沿采集 A0 和 burstcount。一个新的传输可能在下个周期开始。
3. Master B 驱动 address (A1)，burstcount 和 read。slave 置位 waitrequest，导致除了 beginbursttransfer 之外的所有输入保持不变。



4. slave 呈现有效的 readdata 并置位 readdatavalid, 传输 master A 的数据的第一个字。
5. 传输 master A 的第二个字。slave 置低 readdatavalid, 暂停读突发。slave 端口能够保持 readdatavalid 置低任意数量的时钟周期。
6. 返回 master B 的第一个字。

3.5.5.3. 换行突发(Line – Wrapped Bursts)

具有指令缓存的处理器通过使用换行突发(line-wrapped bursts)来提高效率。当处理器请求不在缓存中的数据时, 缓存控制器必须重新填充整个缓存行。对于高速缓存行大小为 64 字节的处理器, 一个高速缓存缺失(cache miss)会导致从内存中读取 64 个字节。如果处理器在出现高速缓存缺失时从地址 0xC 读取, 那么一个低效高速缓存控制器可能会在地址 0 上发出突发, 从而产生来自读取地址 0x0, 0x4, 0x8, 0xC, 0x10, 0x14, 0x18, ... 0x3C 的数据。在第四次读取之前, 请求的数据不可用。对于换行突发, 地址顺序为 0xC, 0x10, 0x14, 0x18, ... 0x3C, 0x0, 0x4 和 0x8。首先返回请求的数据。整个缓存行最终从内存中重新填充。

3.5.6. 读和写响应

对于任何 Avalon -MM slave, 必须以无危险的方式对命令进行处理。读和写响应按接受命令的顺序发出。

3.5.6.1. Avalon -MM 读和写响应的传输顺序(Masters and Slaves)

对于任何 Avalon -MM master:

- *Avalon Interface Specifications* 保证对同一 slave 发出的命令以命令发出的顺序到达 slave, slave 以命令发出的顺序进行响应。
- 不同的 slave 可以以 master 发布命令不同的顺序接收和响应命令。成功后 slave 以命令发出的顺序进行响应。
- 响应(如果存在)以命令发出的顺序进行返回, 不管读或写命令是针对同一 slave 还是不同的 slave。
- *Avalon Interface Specifications* 对不同 master 之间的传输顺序不予保证。

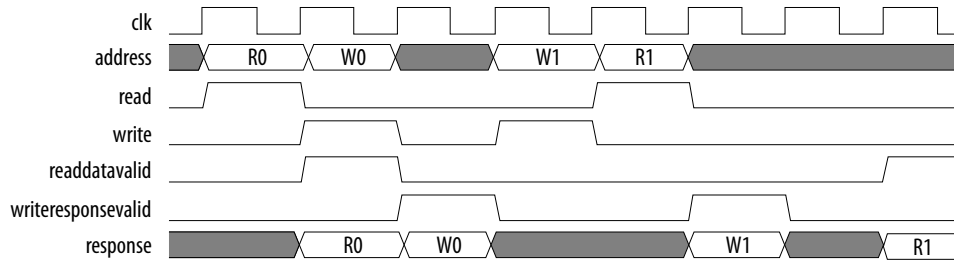
3.5.6.2. Avalon -MM 读和写响应时序图

下图显示了 Avalon -MM 读和写响应的命令接受和命令发送的顺序。由于读和写接口共享 response 信号, 因此一个接口不能够在同一时钟周期中发出或接受一个写响应和一个读响应。

读响应, 对每个 readdata 发送一个响应。一个为 <N> 的读突发长度会产生 <N> 个响应。

写响应, 对每个写命令发送一个响应。一个写突发仅产生一个响应。从接口(slave interface)在接收到突发中的最后写传输后发送响应。当一个接口包括 writeresponsevalid 信号时, 所有写命令必须通过写响应来完成。

图 16. Avalon -MM 读和写响应时序图



3.5.6.2.1. minimumResponseLatency 时序图(包含 readdatavalid 或 writeresponsevalid)

对于 readdatavalid 或者 writeresponsevalid 的接口，默认的一个周期 minimumResponseLatency 能导致 Avalon-MM master 上时序收敛的困难。

下面时序图显示了 1 或 2 个周期的 minimumResponseLatency 的行为。请注意：实际的响应延迟也可以大于这些时序图中显示的最小允许值。

图 17. minimumResponseLatency 等于一个周期

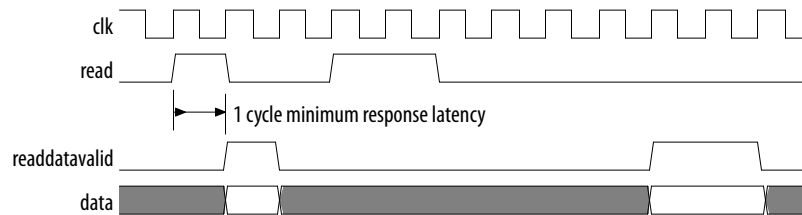
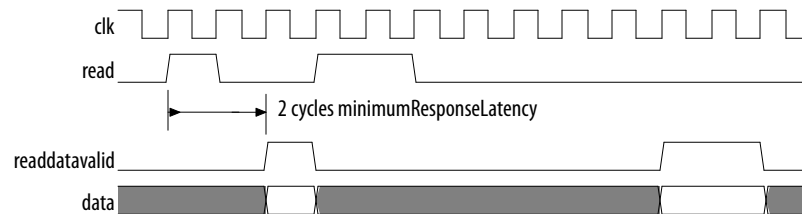


图 18. minimumResponseLatency 等于两个周期



兼容性

具有相同 minimumResponseLatency 的接口可以互操作，而无需任何调整。如果 master 的 minimumResponseLatency 高于 slave，那么需使用流水线寄存器对差异进行补偿。流水线寄存器应该从 slave 延迟 readdata。如果 slave 的 minimumResponseLatency 高于 master，那么接口可以互操作，而无需任何调整。

3.6. 地址对齐(Address Alignment)

互连(interconnect)仅支持对齐的访问。master 只能以符号形式发送其数据宽度倍数的地址。master 可以通过置低 byteenables 以写入部分字。例如，地址 2 上的 2 字节写入的 byteenables 是 4' b1100。



3.7. Avalon -MM Slave 寻址

动态总线大小调整(dynamic bus sizing)在不同数据宽度的 master-slave 对之间的传输期间管理数据。从数据(slave data)在主地址空间(master address space)中以连续字节对齐。

如果主数据宽度大于从数据宽度，那么主地址空间中的字映射到从地址空间中的多个位置。例如，从 16-bit slave 的 32-bit master read 会在 slave 一端产生两次读取传输。读取是连续的地址。

如果 master 比 slave 窄，那么 interconnect 管理 slave 字节通道。在 master 读传输期间，interconnect 仅向较窄的 master 呈现 slave 数据的相应字节通道。在 master 写传输期间，interconnect 自动置位 byteenable 信号，只将数据写入指定的 slave 字节通道。

Slave 的数据宽度必须是 8, 16, 32, 64, 128, 256, 512 或者 1024 bits。下表显示了在一个执行全字(full-word)访问的 32-bit master 中各种宽度的 slave 数据的对齐。在此表中，OFFSET[N]指的是从地址空间(slave address space)的从字尺寸偏移(slave word size offset)。

表 12. 动态总线大小调整 Master 到 Slave 地址映射(Dynamic Bus Sizing Master-to-Slave Address Mapping)

Master Byte Address (1)	访问	32-Bit Master Data		
		当访问一个 8-Bit 从接口	当访问一个 16-Bit 从接口	当访问一个 64-Bit 从接口
0x00	1	OFFSET[0] 7..0	OFFSET[0] 15..0 (2)	OFFSET[0] 31..0
	2	OFFSET[1] 7..0	OFFSET[1] 15..0	—
	3	OFFSET[2] 7..0	—	—
	4	OFFSET[3] 7..0	—	—
0x04	1	OFFSET[4] 7..0	OFFSET[2] 15..0	OFFSET[0] 63..32
	2	OFFSET[5] 7..0	OFFSET[3] 15..0	—
	3	OFFSET[6] 7..0	—	—
	4	OFFSET[7] 7..0	—	—
0x08	1	OFFSET[8] 7..0	OFFSET[4] 15..0	OFFSET[1] 31..0
	2	OFFSET[9] 7..0	OFFSET[5] 15..0	—
	3	OFFSET[10] 7..0	—	—
	4	OFFSET[11] 7..0	—	—
0x0C	1	OFFSET[12] 7..0	OFFSET[6] 15..0	OFFSET[1] 63..32
	2	OFFSET[13] 7..0	OFFSET[7] 15..0	—
	3	OFFSET[14] 7..0	—	—
	4	OFFSET[15] 7..0	—	—
And so on		And so on	And so on	And so on
注释： 1. 尽管 master 发出字节地址，但 master 访问全 32-bit 字。 2. 对于所有的 slave 入口，[<n>]是字偏移，下标值是字中的比特。				

4. Avalon 中断接口

Avalon 中断接口使从组件(slave components)能够向主组件(master components)发送事件信号。例如，DMA 控制器可以在完成一个 DMA 传输后中断处理器。

4.1. 中断发送器(Interrupt Sender)

中断发送器将单个中断信号驱动到中断接收器。irq 信号的时序必须与其相关联时钟的上升沿同步。irq 与任何其他接口上的任何传输无关。irq 必须一直置位，直到在相关联的确认 Avalon -MM 从接口上确认接收。

中断是特定于组件的。接收器通常通过从 Avalon -MM 从接口中读取中断状态寄存器来确定相应的响应。

4.1.1. Avalon 中断发送器信号角色

表 13. 中断发送器信号角色

信号角色	宽度	方向	是否需要	描述
irq irq_n	1-32	Output	Yes	中断请求。中断发送器将中断信号驱动到中断接收器。

4.1.2. 中断发送器属性(Interrupt Sender Properties)

表 14. 中断发送器属性(Interrupt Sender Properties)

属性名称	默认值	合法值	描述
associatedAddressablePoint	N/A	此组件上的 Avalon -MM slave 的名称。	Avalon -MM 从接口的名称，提供对寄存器的访问来服务中断。
associatedClock	N/A	此组件上的时钟接口的名称。	时钟接口的名称，中断发送器与此时钟接口同步。对于此属性，发送器和接收器可以有不同的值。
associatedReset	N/A	此组件上的复位接口的名称。	复位接口的名称，中断发送器与此复位接口同步。

4.2. 中断接收器(Interrupt Receiver)

中断接收器接口接收来自中断发送器接口的中断。包含 Avalon -MM 主接口的组件可以包含一个中断接收器，用于检测由包含中断发送器接口的从组件置位的中断。中断接收器接受来自每个中断发送器(作为一个单独的比特)的中断请求。



4.2.1. Avalon 中断接收器信号角色

表 15. 中断接收器信号角色

信号角色	宽度	方向	是否需要	描述
irq	1 - 32	Input	Yes	irq 是一个 is an $<n>$ -bit 向量，其中每个比特直接对应一个 IRQ sender，没有固有的优先级假设。

4.2.2. 中断接收器属性(Interrupt Receiver Properties)

表 16. 中断接收器属性(Interrupt Receiver Properties)

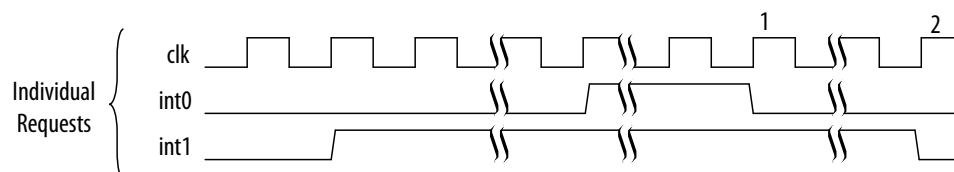
属性名称	默认值	合法值	描述
associatedAddressable Point	N/A	Avalon -MM 主接口的名称	Avalon -MM 主接口的名称，用于对此接口上接收到的中断进行服务。
associatedClock	N/A	Avalon 时钟接口的名称	Avalon 时钟接口的名称，中断接收器与此时钟接口同步。对于此属性，发送器和接收器可以有不同的值。
associatedReset	N/A	Avalon 复位接口的名称	复位接口的名称，中断接收器与此复位接口同步。

4.2.3. 中断时序

Avalon -MM master 先服务优先级 0 中断，再服务优先级 1 中断。

图 19. 中断时序

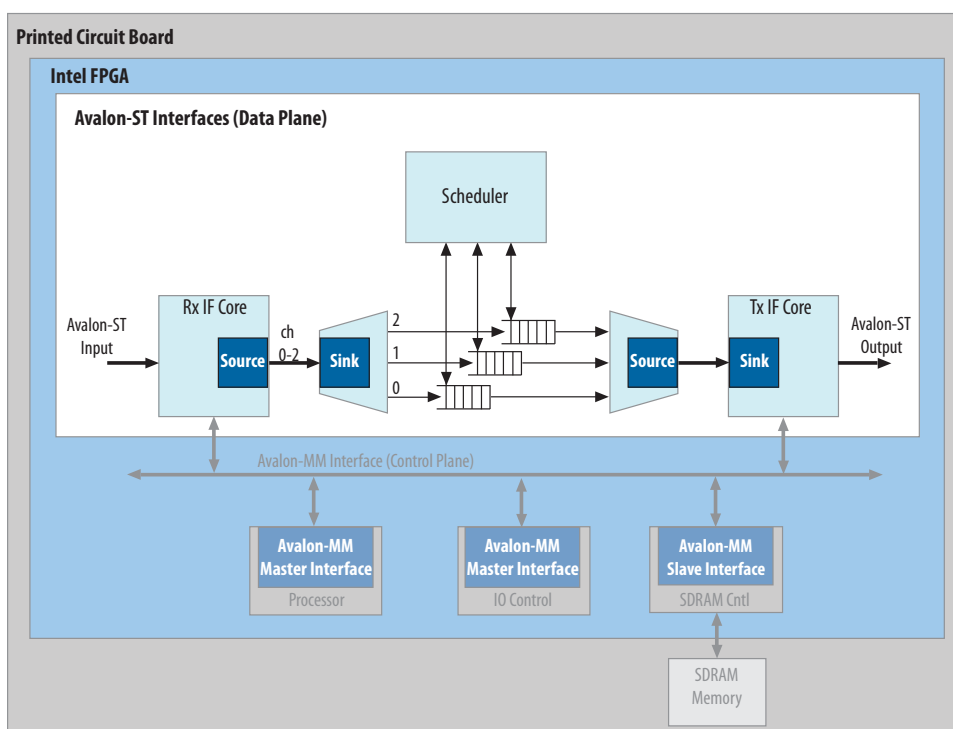
在下图中，中断 0 具有更高的优先级。当 int0 置位时，中断接收器正在处理 int1。int0 handler 被调用并完成。然后，int1 handler 继续执行。下图显示了 int0 在 time 1 上置低。int1 在 time 2 上置低。



5. Avalon Streaming 接口

Avalon Streaming (Avalon -ST)接口可用于那些驱动高带宽，低延迟，单向数据的组件。典型应用包括多路复用流(multiplexed streams)，数据包和 DSP 数据。Avalon -ST 接口信号可以描述支持单个数据流的传统流接口，而无需了解通道或数据包边界。此接口还支持更复杂的协议，能够在多个通道之间进行数据包交错的突发和数据包传输。

图 20. Avalon -ST 接口- Avalon -ST 接口的典型应用



Avalon -ST source 和 sink 接口都未必是可互操作的。然而，如果两个接口对同一应用空间提供兼容的功能，那么可以使用适配器(adapter)来实现它们之间的互操作。

Avalon -ST 接口支持要求如下功能的数据通路：

- 低延迟，高吞吐量的点对点数据传输
- 灵活数据包交错的多通道支持
- 通道的边带信号(sideband signaling)，错误和数据包描述的开始和结束
- 支持数据突发(data bursting)
- 自动接口适配(automatic interface adaptation)



5.1. 术语和概念

Avalon -ST 接口协议定义了以下术语和概念：

- **Avalon Streaming System**—一个 Avalon Streaming 系统包含一个或多个 Avalon -ST 连接，将数据从 source 接口传输到 sink 接口。上面显示的系统包含 Avalon -ST 接口，将数据从系统输入传输到系统输出。 Avalon -MM 控制和状态寄存器接口提供软件控制。
- **Avalon Streaming Components**—一个使用 Avalon -ST 接口的典型系统，结合了多个功能模块(称为组件)。系统设计人员配置组件，并将它们连接在一起来实现一个系统。
- **Source and Sink Interfaces and Connections**—当连接两个组件时，数据从 source 接口流向 sink 接口。 *Avalon Interface Specifications* 将一个与 sink 接口连接的 source 接口的结合称为 *connection*。
- **Backpressure**—背压使 sink 能够对 source 发出信号来停止发送数据。背压支持是可选的。 sink 使用背压来停止数据流的原因如下：
 - 当 sink FIFOs 已满时
 - 当其输出接口上出现拥塞时
- **Transfers and Ready Cycles**—一个传输导致从 source 接口到 sink 接口的数据和控制传播。对于数据接口，一个 ready 周期是 sink 能够接受一个传输的周期。
- **Symbol**—符号是最小的数据单位。对于大多数数据包接口，符号是一个字节。一个或多个符号构成一个周期中传输的单个数据单元。
- **Channel**—通道是一条物理或逻辑路径或链路，信息通过通道在两个端口之间进行传递。
- **Beat**—beat 是 source 与 sink 接口(由一个或多个符号组成)之间的单一周期传输。
- **Packet**—数据包是 source 同时发送的数据和控制信号的集合。数据包可以包含一个标头，以帮助路由器和网络设备将数据包指引到正确的目的地。应用程序定义数据包格式，而不是此规范。 Avalon -ST 数据包的长度可以变化，并且可以跨连接交错。通过使用 Avalon -ST 接口，数据包的使用是可选的。

5.2. Avalon Streaming 接口信号角色

一个 Avalon -ST source 或者 sink 接口中的每个信号对应一个 Avalon -ST 信号角色。一个 Avalon -ST 接口可以只包含每个信号角色的一个实例。所有的 Avalon -ST 信号角色都应用于 source 和 sink，并对 source 和 sink 有相同的含义。

表 17. Avalon -ST 接口信号

在下表中，所有信号角色都是高电平有效(active high)。

信号角色(Signal Roles)	宽度	方向	是否需要	描述
基本信号				
channel	1 - 128	Source → Sink	No	在当前周期上传输的数据的 channel 数。 如果一个接口支持通道信号，那么此接口也必须定义 maxChannel 参数。
data	1 - 4,096	Source → Sink	No	从 source 到 sink 的 data 信号，通常承载正在传输的大量信息。 参数进一步定义了 data 信号的内容和格式。
继续...				

信号角色(Signal Roles)	宽度	方向	是否需要	描述
error	1 - 256	Source → Sink	No	一个比特掩码，对影响当前周期中正在传输的数据的错误进行标记。单一比特的 error 信号对组件识别的每个错误进行屏蔽。errorDescriptor 定义了 error 信号属性。
ready	1	Sink → Source	No	置高表明 sink 能够接受数据。sink 在 cycle <n>上置位 ready，将 cycle <n + readyLatency >标记为一个 ready cycle。source 可仅置位 valid，在 ready 周期传输数据。 无 ready 输入的 source 不支持背压。无 ready 输出的 sink 从不需要背压。
valid	1	Source → Sink	No	source 置位此信号来限定所有其他 source 到 sink 信号。sink 在 ready 周期(valid 被置位)上采样数据和其他 source-to-sink 信号。所有其他周期被忽略。 无 valid 输出的 source 在 sink 没有置位背压的每个周期上提供有效数据。无 valid 输入的 sink 期望在它们没有背压的每个周期上有效数据。
数据包传输信号				
empty	1 - 5	Source → Sink	No	表示空符号的数量，也就是不代表有效数据。在每个 beat 一个符号的接口上不需要 empty 信号。
endofpacket	1	Source → Sink	No	由 source 置位以标记一个数据包的结束。
startofpacket	1	Source → Sink	No	由 source 置位以标记一个数据包的开始。

5.3. 信号排序和时序(Signal Sequencing and Timing)

5.3.1. 同步接口

一个 Avalon -ST 连接的所有传输与相关时钟信号的上升沿同步发生。从一个 source 接口到一个 sink 接口的所有输出(包括 data, channel 和 error 信号)都必须在时钟的上升沿寄存。sink 接口的输入不必寄存。在 source 上寄存信号有助于高频操作。

5.3.2. 时钟使能(Clock Enables)

Avalon -ST 组件通常不包括时钟使能输入。Avalon -ST 信号本身足以确定组件是否应该被启用的周期。Avalon -ST 兼容组件可以为其内部逻辑提供时钟使能输入。但是，使用时钟使能的组件必须确保接口的时序符合协议。

5.4. Avalon -ST 接口属性

表 18. Avalon -ST 接口属性

属性名称	默认值	合法值	描述
associatedClock	1	Clock interface	与此 Avalon -ST 接口同步的 Avalon 时钟接口的名称。
associatedReset	1	Reset interface	与此 Avalon -ST 接口同步的 Avalon 复位接口的名称。
beatsPerCycle	1	1,2,4,8	指定在单个周期中传输的 beat 的数量。此属性支持使用相同的 start_of_packet, end_of_packet, ready 和 valid 信号传输 2 个单独但相关的流(stream)。
继续...			



属性名称	默认值	合法值	描述
			beatsPerCycle 是一个很少使用的 Avalon -ST 协议的特性。
dataBitsPerSymbol	8	1 - 512	定义每个符号的比特数。例如，面向字节的接口有 8-bit 符号。此值不限于 2 的幂。
emptyWithinPacket	false	true, false	为 true 时，empty 对整个数据包有效。
errorDescriptor	0	List of strings	字符串列表，描述与错误信号的每个比特相关的错误。列表的长度必须与错误信号中的比特数相同。列表中的第一个字应用于最高位。例如，“crc, overflow”意思是 error 的 bit[1] 指示一个 CRC 错误。bit[0] 表示一个上溢错误。
firstSymbolInHighOrderBits	true	true, false	为 true 时，一阶符号(first-order symbol)被驱动到数据接口的最高有效位。最高阶符号(highest-order symbol)在此规范中标记为 D0。当此属性设为 false 时，第一个符号出现在低位上。D0 出现在 data[7:0]。对于 32-bit 总线，如果为 true，那么 D0 出现在 bits[31:24] 上。
maxChannel	0	0 - 255	数据接口能够支持的最大通道数量。
readyLatency	0	0 - 8	定义了 ready 信号的置位与 valid 信号的置位之间的关系。如果 readyLatency = <n>，其中 n > 0，那么 valid 在 ready 置位后仅能够置位 <n> 周期。
readyAllowance ⁽¹⁾	0	0 - 8	定义了 sink 在 ready 置低后能够接受的传输数量。当 readyAllowance = 0 时，sink 在 ready 置低后不能接受任何传输。如果 readyAllowance = <n>，其中 <n> > 0，那么 sink 在 ready 置低后最多能够接受 <n> 个传输。
symbolsPerBeat	1	1 - 32	在每个周期上传输的符号数量。

5.5. 典型的数据传输

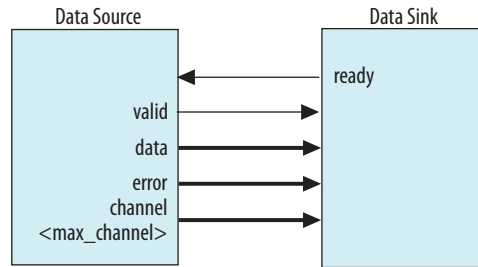
本节定义了一个 source 接口到 sink 接口的数据传输。在所有情况下，data source 和 data sink 都必须符合规范。data sink 不负责检测 source 协议错误。

5.6. 信号详情

下图显示了 Avalon -ST 接口通常包含的信号。一个典型的 Avalon -ST source 接口将 valid, data, error 和 channel 信号驱动到 sink。sink 可以使用 ready 信号来应用背压(backpressure)。

- (1)
- 如果 readyLatency = 0，那么 readyAllowance 可以为 0，也可以大于 0。
 - 如果 readyLatency > 0，那么 readyAllowance 必须等于或大于 readyLatency。
 - 如果 source 或者 sink 没有指定 readyAllowance 的值，那么 readyAllowance=readyLatency。设计不需要增添 readyAllowance，除非想让 source 或者 sink 利用此特性。

图 21. 典型的 Avalon -ST 接口信号



有关这些信号的更多详情如下：

- **ready**—在支持背压的接口上，sink 置位 ready 来标记可能发生传输的周期。如果 ready 在 cycle <n>上置位，那么 cycle <n + readyLatency>被认为是一个 ready cycle。
- **valid**—valid 信号限定任何从 source 到 sink 的数据传输的周期上的有效数据。在每个有效周期上，sink 对 data 信号和其他 source 到 sink 信号进行采样。
- **data**—data 信号承载着从 source 到 sink 传输的大量信息。数据信号由在每个时钟周期上传输的一个或多个符号组成。dataBitsPerSymbol 参数定义了数据信号如何分成符号。
- **error**—在 error 信号中，每个比特对应于一个可能的错误条件。任何周期上的一个 0 值都代表此周期上无错误数据。此规范没有定义当检测到一个错误时组件应该采取的操作。
- **channel**—source 驱动可选的 channel 信号来表明数据属于哪个通道。对一个特定接口，channel 的含义要取决于应用。在某些应用中，channel 表示接口数。在其他应用中，channel 表示页数或者时间段(timeslot)。当使用 channel 信号时，在每个活动周期中传输的所有数据都属于同一通道(channel)。source 可以在连续的活动周期中更改为一个不同的通道。使用 channel 信号的

接口必须定义 maxChannel 参数来表示最大通道数。如果一个接口支持的通道数量是动态改变的，那么 maxChannel 表示此接口能够支持的最大数量。

5.7. 数据布局(Data Layout)

图 22. 数据符号(Data Symbols)

下图显示了一个 64-bit 数据信号，dataBitsPerSymbol=16。symbol 0 是最高有效符号(most significant symbol)。

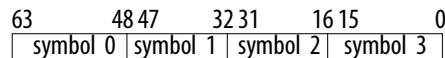
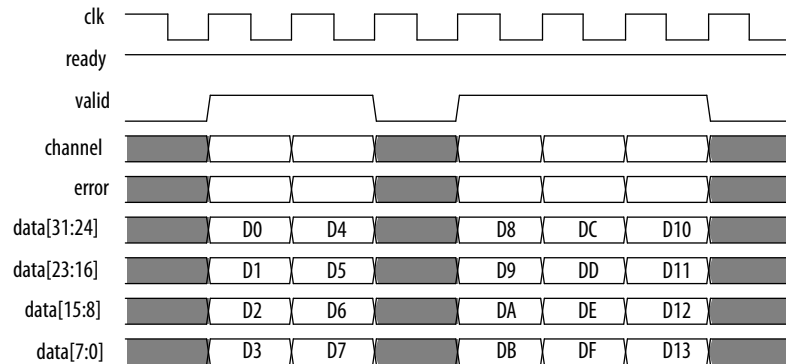




图 23. 数据的布局

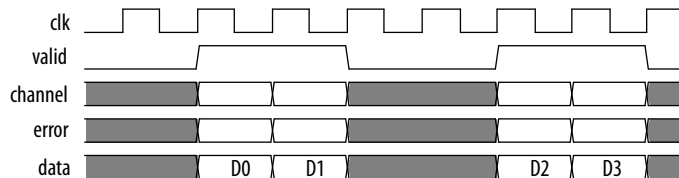
下面时序图显示了一个 32-bit 实例，其中 $\text{dataBitsPerSymbol}=8$ ， $\text{symbolsPerBeat}=4$ 和 $\text{beatsPerCycle}=1$ 。



5.8. 无背压的数据传输

无背压的数据传输是最基本的 Avalon -ST 数据传输。在任何特定的时钟周期上，source 接口驱动 data 和可选的 channel 和 error 信号，并置位 valid。如果 valid 置位，那么 sink 接口在参考时钟的上升沿采样这些信号。

图 24. 无背压的数据传输



5.9. 有背压的数据传输

sink 置位 ready 一个时钟周期来表明已对一个活动周期准备就绪。如果 sink 对数据已准备就绪，那么周期就是一个 ready 周期。在一个 ready 周期中，source 可以置位 valid 并提供数据给 sink。如果 source 没有要发送的数据，那么 source 置低 valid，并且能够将 data 驱动到任何值。

支持背压的接口定义 readyLatency 参数来表示从 ready 置位到能够驱动有效数据的周期数量。如果 readyLatency 为非零值，那么 $\text{cycle} < n + \text{readyLatency} >$ 是一个 ready cycle (如果 ready 在 $\text{cycle} < n >$ 上置位)。

当 $\text{readyLatency} = 0$ 时，只有在 ready 和 valid 在同一周期上置位时才会出现数据传输。在此模式下，source 在发送有效数据之前不会接收到 sink 的 ready 信号。source 提供数据，只要有有效数据就会置位 valid。source 等待 sink 采集数据并置位 ready。source 可随时改变数据。只有 ready 和 valid 都置位时，sink 才从 source 采集输入数据。

当 $\text{readyLatency} \geq 1$ 时，sink 在 ready 周期本身之前置位 ready。source 通过置位 valid 在相应的周期中能够作出响应。在不是 ready 周期的周期中，source 可以不置位 valid。

readyAllowance 定义了 ready 置低时 sink 能够采集的传输数量。当 readyAllowance = 0 时，sink 在 ready 置低后不能接受任何传输。如果 readyAllowance = <n>，其中 n > 0，那么 sink 在 ready 置低后最多能够接受<n>个传输。

5.9.1. 使用 readyLatency 和 readyAllowance 的数据传输

以下规则应用于使用 readyLatency 和 readyAllowance 传输数据时。

- 如果 readyLatency 为 0，那么 readyAllowance 可以大于或等于 0。
- 如果 readyLatency 大于 0，那么 readyAllowance 可大于或等于 readyLatency。

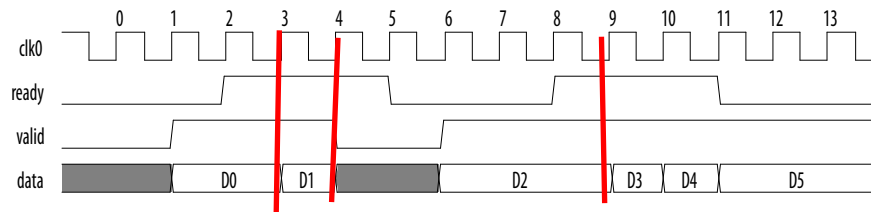
当 readyLatency = 0 和 readyAllowance = 0 时，只有在 ready 和 valid 都置位时数据才会传输。在此情况下，source 在发送有效数据之前不会接收到 sink 的 ready 信号。source 提供数据并尽可能置位 valid。source 等待 sink 采集数据并置位 ready。source 可随时改变数据。只有 ready 和 valid 都置位时，sink 才从 source 采集输入数据。

图 25. readyLatency = 0, readyAllowance = 0

当 readyLatency = 0 和 readyAllowance = 0 时，source 能够随时置位 valid。只有在 ready = 1 时，sink 才从 source 采集数据。

下图显示了这些事件：

1. 在 cycle 1 中，source 提供数据并置位 valid。
2. 在 cycle 2 中，sink 置位 ready 和 D0 传输。
3. 在 cycle 3 中，D1 传输。
4. 在 cycle 4 中，sink 置位 ready，但 source 不驱动有效数据。
5. source 在 cycle 6 上提供数据并置位 valid。
6. 在 cycle 8 中，sink 置位 ready，所以 D0 传输。
7. D3 在 cycle 9 上传输，D4 在 cycle 10 上传输。



ready和valid同时为高时，data才会被sink采集

图 26. readyLatency = 0, readyAllowance = 1

当 readyLatency = 0 和 readyAllowance = 1 时, sink 在 ready = 0 后能够在采集一次数据传输。

下图显示了这些事件:

1. 在 cycle 1 中, source 提供数据并置位 valid, 而 sink 置位 ready。D0 传输。
2. D1 在 cycle 2 中传输。
3. 在 cycle 3 中, ready 置低, 但是由于 readyAllowance = 1 允许再一次传输, 因此 D2 输出。
4. 在 cycle 5 中, valid 和 ready 都置位, 所以 D3 传输。
5. 在 cycle 6 中, source 置低 valid, 所以没有数据传输。
6. 在 cycle 7 中, valid 置位, ready 置低, 然而由于 readyAllowance = 1 允许再一次传输, 所以 D4 传输。

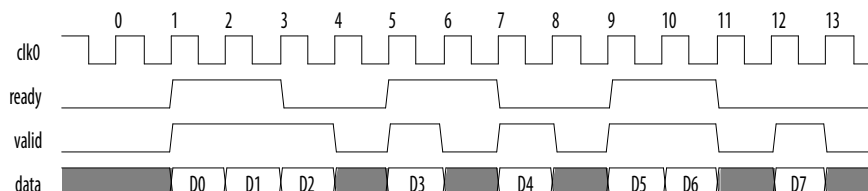
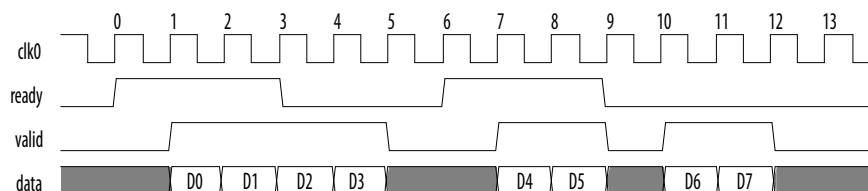


图 27. readyLatency = 1, readyAllowance = 2

当 readyLatency = 1 和 readyAllowance = 2 时, sink 能够在 ready 置位后的一个周期传输数据, 并且在 ready 置低后允许另外两个传输周期。

下图显示了这些事件:

1. 在 cycle 0 中, sink 置位 ready。
2. 在 cycle 1 中, source 提供数据并置位 valid。立即出现传输。
3. 在 cycle 3 中, sink 置低 ready, 但 source 仍然置位 valid 并驱动有效数据, 因为 sink 在 ready 置低后的两个周期能够采集数据。
4. 在 cycle 10 中, sink 已置低 ready, 但 source 置位 valid 并驱动有效数据, 因为 sink 在 ready 置低后的两个周期能够采集数据。



适应要求(Adaptation Requirements)

下表描述了 source 和 sink 接口是否要求适应。

表 19. Source/Sink 适应要求

readyLatency	readyAllowance	适应(Adaptation)
Source readyLatency = Sink readyLatency	Source readyAllowance = Sink readyAllowance	不要求适应: sink 能够采集所有传输。
	Source readyAllowance > Sink readyAllowance	要求适应: ready 置低后, source 能够发送比 sink 采集的更多的传输。

继续...

readyLatency	readyAllowance	适应(Adaptation)
	Source readyAllowance < Sink readyAllowance	不要求适应: ready 置低后, sink 能够采集比 source 发送的更多的传输。
Source readyLatency > Sink readyLatency	Source readyAllowance = Sink readyAllowance	不要求适应: ready 被置位, source 开始发送的时间晚于 sink 能够采集的时间。ready 置低后, source 能够发送与 sink 能够采集一样多的传输。
	Source readyAllowance > Sink readyAllowance	要求适应: ready 置低后, source 能够发送比 sink 采集的更多的传输。
	Source readyAllowance < Sink readyAllowance	不要求适应: ready 置低后, source 能够发送比 sink 采集的更少的传输。
Source readyLatency < Sink readyLatency	Source readyAllowance = Sink readyAllowance	要求适应: source 能够在 sink 采集之前开始发送传输。
	Source readyAllowance > Sink readyAllowance	要求适应: source 能够在 sink 采集之前开始发送传输。此外, 在 ready 置低后, source 能够发送比 sink 能够采集的更多的传输。
	Source readyAllowance < Sink readyAllowance	要求适应: source 能够在 sink 采集之前开始发送传输。

5.9.2. 使用 readyLatency 的数据传输

如果 source 或者 sink 没有指定 readyAllowance 的值, 那么 readyAllowance=readyLatency。使用 source 和 sink 的设计不需要增添 readyAllowance, 除非想让 source 或者 sink 利用此特性。

图 28. 使用背压的传输, readyLatency=0

下图显示了这些事件:

1. 尽管 sink 未准备好, source 也在 cycle 1 上提供数据并置位 valid。
2. 在进入下一个数据周期前, source 等待直到 cycle 2, 当 sink 置位 ready。
3. 在 cycle 3 中, source 在同一周期上驱动数据, sink 准备好接收数据。
4. 在 cycle 4 中, sink 置位 ready, 但 source 不驱动有效数据。

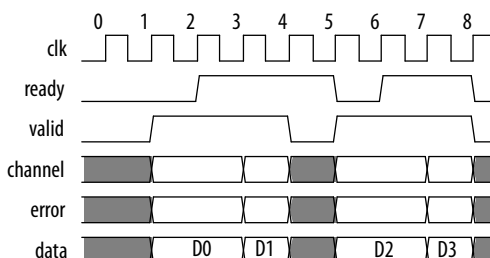


图 29. 使用背压的传输, readyLatency=1

下图分别显示了使用 `readyLatency=1` 和 `readyLatency=2` 的数据传输。在这两种情况下, `ready` 在 `ready cycle` 之前置位, `source` 通过提供数据并置位 `valid` 在 1 或 2 个周期后作出响应。当 `readyLatency` 不为 0 时, `source` 必须在 `non-ready` 周期上置低 `valid`。

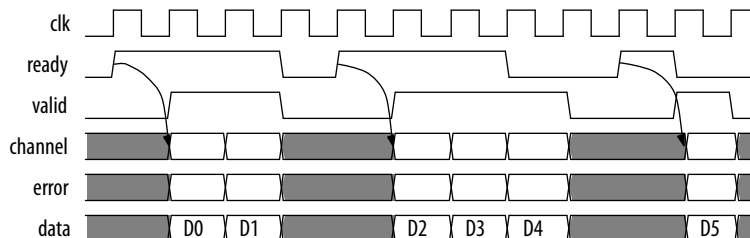
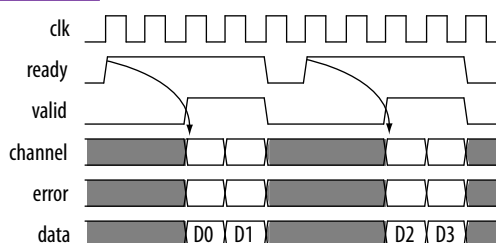


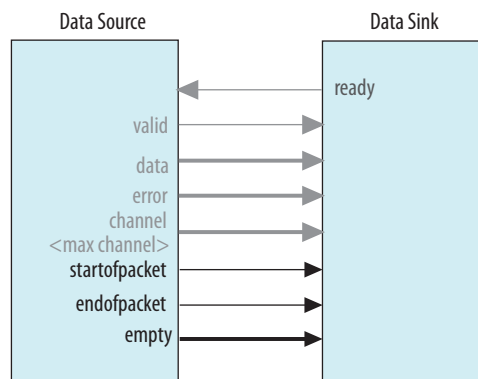
图 30. 使用背压的传输, readyLatency=2



5.10. 数据包数据传输(Packet Data Transfers)

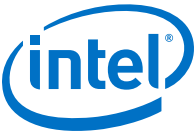
数据包传输属性增加了从 `source` 接口到 `sink` 接口的数据包传输支持。定义了三个附加信号以实现数据包传输。`source` 接口和 `sink` 接口都必须包含这些附加信号才能支持数据包。您只能使用匹配的数据包属性来连接 `source` 接口和 `sink` 接口。`Platform Designer` 不自动将 `startofpacket`, `endofpacket` 和 `empty` 信号添加到不包含这些信号的 `source` 或 `sink` 接口。

图 31. Avalon -ST 数据包接口信号



5.11. 信号详情

- `startofpacket`—支持数据包传输的所有接口都需要 `startofpacket` 信号。`startofpacket` 对包含数据包起始部分的活动周期作标记。只有 `valid` 置位时才会解释此信号才。
- `endofpacket`—支持数据包传输的所有接口都需要 `endofpacket` 信号。`endofpacket` 对包含数据包结束部分的活动周期作标记。只有 `valid` 置位时才会解释此信号。`endofpacket` 可以在同一周期置位。在数据包之间不需要空闲周期。`startofpacket` 信号可以紧跟在前一个 `endofpacket` 信号之后。
- `empty`—可选的 `empty` 信号表示在 `endofpacket` 周期中空符号数量。 `sink` 仅在 `endofpacket` 置位的活动周期中检查 `empty` 的值。空字符总是 `data` 中的最后字符，当 `firstSymbolInHighOrderBits = true` 时由低位比特(`low-order bits`)表示。在所有的数据包接口(这些数据包接口的 `data` 信号承载一个以上的符号数据并具有一个可变长度数据包格式)上都需要 `empty` 信号。`empty` 信号的大小(以比特为单位) $\text{ceil}[\log_2(<\text{symbols per cycle}>)]$ 。



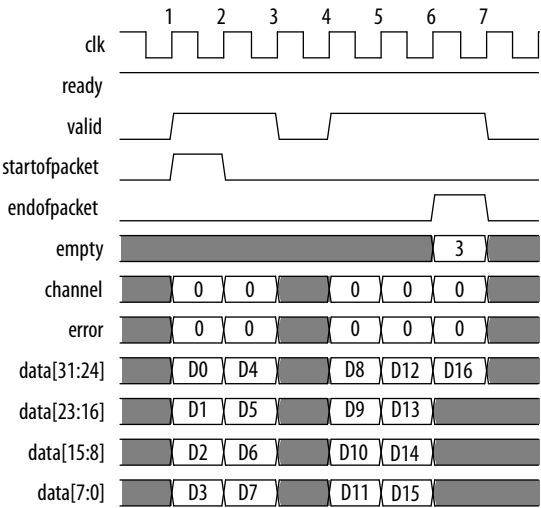
5.12. 协议详情

数据包数据传输遵循与典型数据传输相同的协议，增加了 startofpacket, endofpacket 和 empty。

图 32. 数据包传输

下图显示了一个从 source 接口到 sink 接口的 17-byte 数据包传输，其中 readyLatency=0。此时序图显示了以下事件：

- 1. 当 ready 和 valid 都置位时，数据传输出现在 cycles 1, 2, 4, 5 和 6 上。
- 2. 在 cycle 1 中，startofpacket 置位。传输数据包的前 4 个字节。
- 3. 在 cycle 6 中，endofpacket 置位。empty 的值为 3。此值表示数据包的结束，4 个符号中的 3 个为空。在 cycle 6 中，高位字节 data[31:24]驱动有效数据。



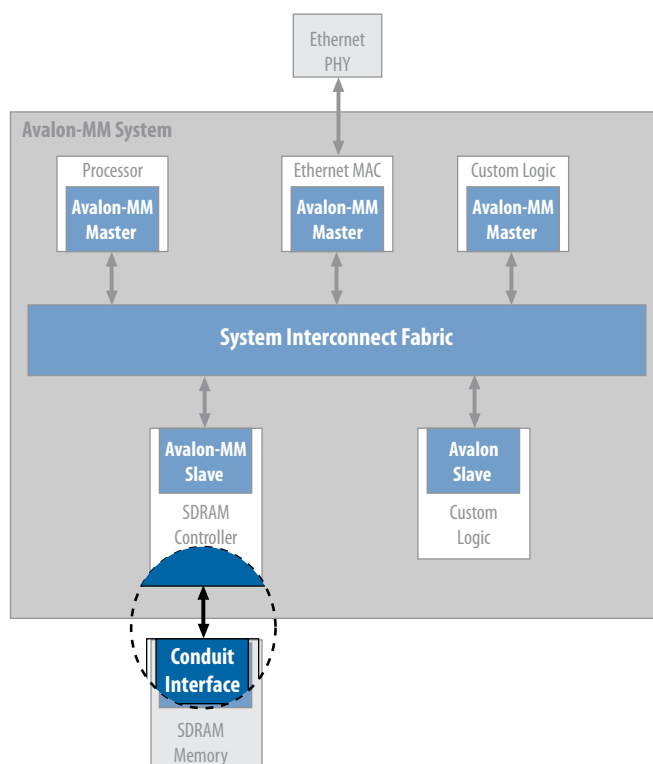
6. Avalon Conduit 接口

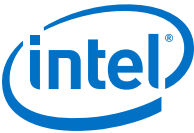
Avalon Conduit 接口对任意信号的集合进行组合。您可以对管道(conduit)信号指定任何角色。然而在连接管道时，角色和宽度必须匹配，方向必须相反。一个 Avalon Conduit 接口可以包括输入，输出和双向信号。一个模块可以有多个 Avalon Conduit 接口以提供逻辑信号分组。Conduit 接口可以声明相关的时钟。当连接的管道接口位于不同的时钟域中时，Platform Designer 会生成错误消息。

注意: 如果可能，您应该使用标准的 Avalon -MM 或 Avalon -ST 接口，而不是创建一个 Avalon Conduit 接口。Platform Designer 对这些接口提供验证和适配(adaptation)。Platform Designer 无能对 Avalon Conduit 接口提供验证或适配(adaptation)。

管道(conduit)接口通常用于驱动片外器件信号，例如 SDRAM 地址，数据和控制信号。

图 33. 管道接口(Coduit Interface)





6.1. Avalon 管道(Conduit)信号角色

表 20. 管道信号角色

信号角色(Signal Roles)	宽度	方向	描述
<any>	<n>	进，出或双向	一个管道接口由一个或多个任意宽度的输入，输出或双向信号组成。管道可以具有任何用户指定的角色。您可以在一个 Platform Designer (Standard)系统中连接兼容的 Conduit 接口，但前提是角色和宽度要匹配，方向要相反。

6.2. 管道角色(Conduit Properties)

管道接口没有属性。

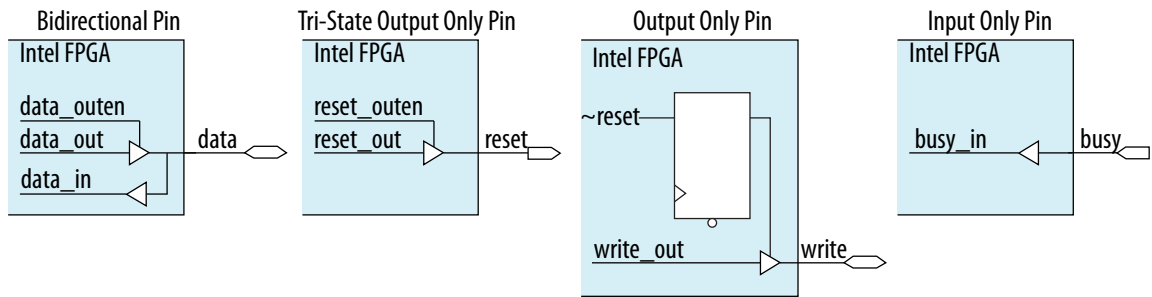
7. Avalon 三态管道接口(Avalon Tristate Conduit Interface)

Avalon Tristate Conduit Interface (Avalon -TC)是一种点对点接口，专为驱动片外组件的片上控制器而设计。此接口支持在多个三态器件之间共享数据，地址和控制管脚。在包含多个外部存储器件的系统中，共享可以保留管脚。

Avalon -TC 以两种方式对更多普通的 Avalon Conduit Interface 进行限制：

- Avalon -TC 需要 request 和 grant 信号。当多个 Tristate Conduit Masters (TCM)对一条共享总线请求访问时，这些信号将使能总线仲裁。
- 必须使用附加到信号角色的后缀来指定信号的管脚类型。三个后缀分别为：_out，_in 和 _outen 。匹配角色前缀识别共享相同 I/O 管脚的信号。以下显示了 Avalon -TC 共享管脚的命名约定。

图 34. 共享管脚类型

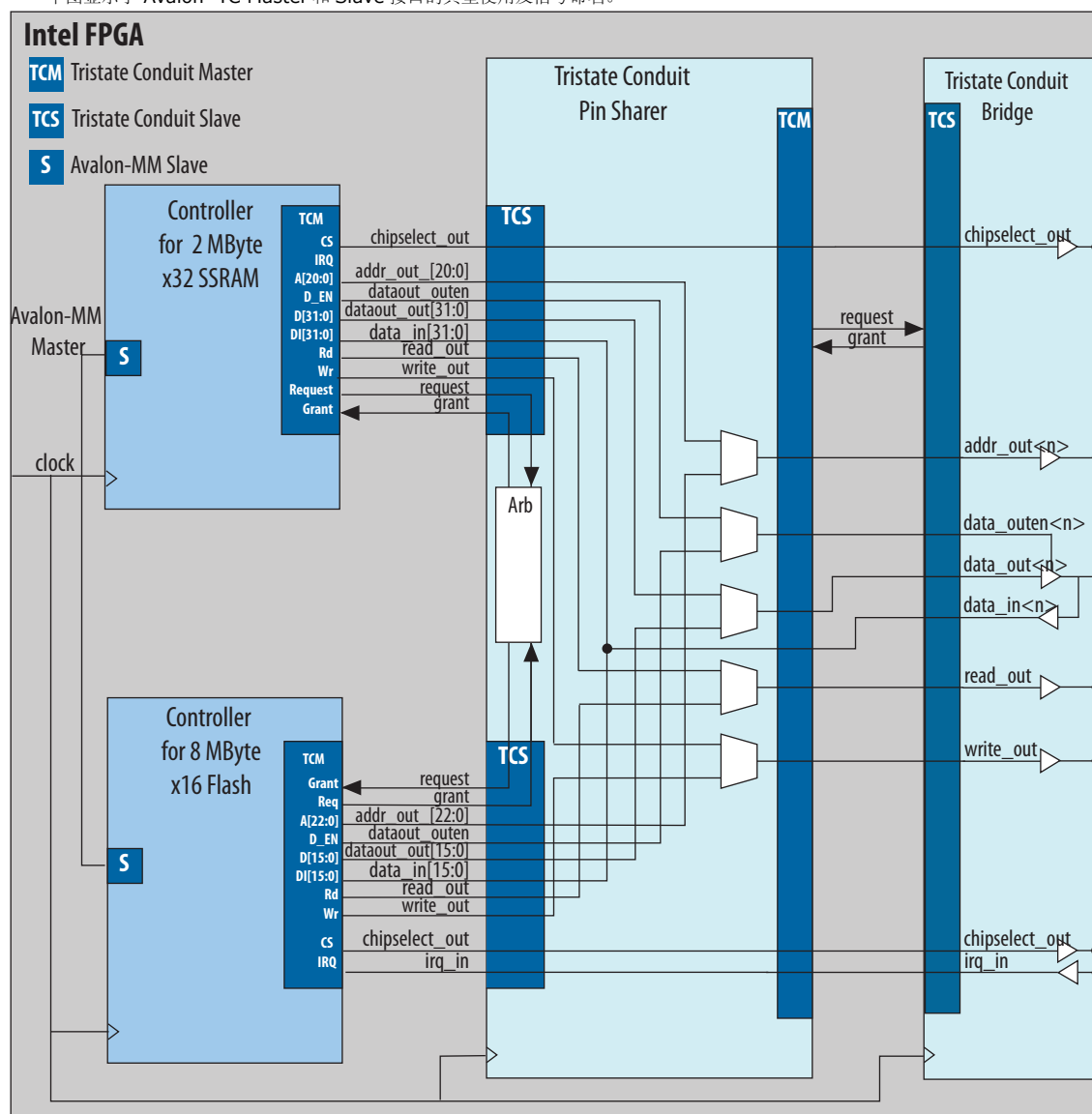


下图显示了使用 Avalon -TC 接口的管脚共享。

- 对于每个 Tristate Conduit Master，Tristate Conduit Pin Sharer 包含各自的 Tristate Conduit Slave Interfaces。每个 master 和 slave pair 都有各自的 request 和 grant 信号。
- Tristate Conduit Pin Sharer 将具有相同角色的信号识别成共享相同 FPGA 管脚的三态信号。在此实例中，以下信号是共享的：addr_out，data_out，data_in，read_out 和 write_out。
- Tristate Conduit Pin Sharer 将一条包含所有共享信号的单一总线驱动到 Tristate Conduit Bridge。如果共享信号的宽度不同，那么 Tristate Conduit Pin Sharer 将它们对齐在 0th bit 上。只要较小的信号控制总线，Tristate Conduit Pin Sharer 就会将高阶管脚(higher-order pins)驱动为 0。
- 未共享的信号直接通过 Tristate Conduit Pin Sharer 进行传播。在此实例中，以下信号没有被共享：chipselct0_out，irq0_out，chipselct1_out 和 irq1_out。
- 连接到同一 Tristate Conduit Pin Sharer 的所有 Avalon -TC 接口都必须位于同一时钟域中。

图 35. 三态管道接口(Tristate Conduit Interface)

下图显示了 Avalon -TC Master 和 Slave 接口的典型使用及信号命名。



关于 Generic Tristate Controller 和 Tristate Conduit Pin Sharer 的详细信息，请参考 *Avalon Tristate Conduit Components User Guide*。

[相关链接](#)

[Avalon Tristate Conduit Components User Guide](#)

7.1. Avalon 三态管道(Conduit)信号角色

下表列出了为 Avalon Tristate Conduit 接口而定义的信号。所有的 Avalon -TC 信号都应用于 masters 以及 slaves，并且对两者具有相同的含义。

表 21. 三态管道(Conduit)接口信号角色

信号角色(Signal Roles)	宽度	方向	是否需要	描述
request	1	Master → Slave	Yes	request 的含义取决于 grant 信号的状态，如下规则所示。 当 request 置位并且 grant 置低时，request 请求对当前周期的访问。 当 request 置位并且 grant 置位时，request 请求对下个周期的访问。因此，request 应该在访问的最后一个周期上置低。 request 信号在总线访问的最后一个周期中置低。 request 信号在传输的最后一个周期后可立即重新置位。如果没有其他 master 请求访问，那么此协议使再仲裁和连续总线访问成为可能。 一旦置位，request 就必须保持置位，直到被授予。因此，最短的总线访问是 2 个周期。关于仲裁时序的实例，请参考 Tristate Conduit Arbitration Timing。
grant	1	Slave → Master	Yes	置位时表明一个三态管道 master 具有执行传输的权限。 grant 信号置位以响应 request 信号。grant 信号保持置位，直到 request 置低后的一个周期。
<name>_in	1 - 1024	Slave → Master	No	一个逻辑三态信号的输入信号。
<name>_out	1 - 1024	Master → Slave	No	一个逻辑三态信号的输出信号。
<name>_outen	1	Master → Slave	No	一个逻辑三态信号的输出使能。

7.2. 三态管道属性(Tristate Conduit Properties)

对于 Avalon -TC 接口没有特殊的属性。

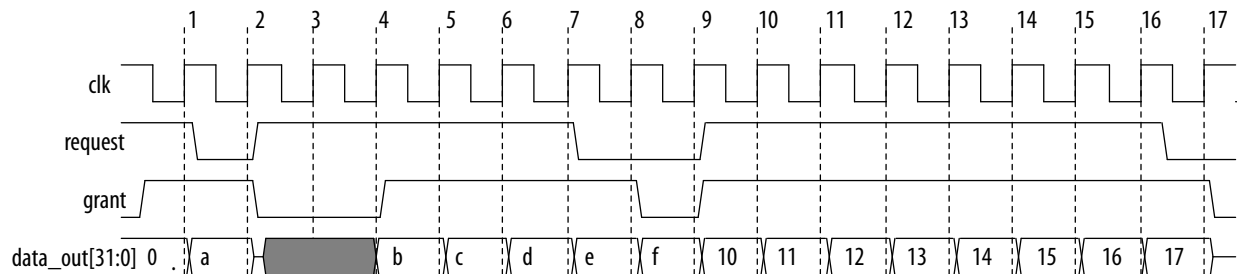
7.3. 三态管道时序(Tristate Conduit Timing)

下图显示了 Tristate Conduit Pin Sharer 的仲裁时序。请注意，器件能够在授予的周期中驱动或接收有效数据。

图 36. 三态管道仲裁时序(Tristate Conduit Conduit Timing)

此图显示了事件序列：

- 在 cycle 1 中，三态管道 slave 置位 grant。slave 在 cycle 1 和 2 中驱动有效数据。
- 在 cycle 4 中，三态管道 slave 置位 grant。slave 在 cycles 5 - 8 中驱动有效数据。
- 在 cycle 9 中，三态管道 slave 置位 grant。slave 在 cycles 10 - 17 中驱动有效数据。
- cycles 3, 4 和 9 不包含有效数据。





A. 已弃用的信号

已弃用的信号实现不再需要或已被取代的功能。

begintransfer

Avalon -MM masters 的一个输出。在一个传输的起始置位一个周期。此信号未被使用，不是必要的信号。

chipselct or chipselct_n

chipselct 或 chipselct_n: 下面描述的芯片选择信号在 Avalon Tristate Conduct (Avalon -TC)接口类型(包含一个芯片选择信号)发布时弃用。

以前 chipselct 是一个 Avalon Memory-Mapped (Avalon -MM)从接口(在一个读传输或写传输的开始发送信号)的 1-bit 输入。当前的 Platform Designer interconnect 根据地址和地址映射将读和写信号从 master 过滤掉。Platform Designer interconnect 仅驱动读和写信号到相应的 Avalon -MM slave, 从而使芯片选择不再是必要的。

此信号可以追溯到非常早期的微处理器设计。CPLD 解码微处理器地址并对外设(经常是异步的)生成芯片选择。通过使用同步系统就不需要此信号。

flush

从 *Avalon Interface Specifications* 的 1.2 版本中删除的信号。之前用于 master 对流水线读取清除待传输。

B. Avalon 接口规范的文档修订历史

文档版本	Intel Quartus Prime 版本	修订内容
2018.09.26	18.1	在写突发(Write Bursts)部分, 增添了一个陈述: 全部为 0 的 byteenables 的写操作被传递到 Avalon -MM slave, 作为有效传输。
2018.09.24	18.1	在 Avalon Memory-Mapped Interface Signal Roles 中增添了连续的字节使能支持。
2018.05.22	18.0	作了如下变更: <ul style="list-style-type: none"> 在 Avalon-ST Interface Properties 表中, 更正了 beatsPerCycle 的默认值。默认值为 1。 在 Avalon-ST Interface Properties 表中, 增添了 beatsPerCycle 的合法值。合法值为 1, 2, 4 和 8。 更正了次要错误和拼写错误。
2018.05.07	18.0	作了如下变更: <ul style="list-style-type: none"> 增添了对 readyAllowance 参数的支持。 更新了 Data Transfers with Backpressure 主题, 加入对 readyAllowance 参数的支持。 修复了次要错误和拼写错误。
2018.03.22	17.1	作了如下变更: <ul style="list-style-type: none"> 对 Read and Write Transfers with Waitrequest 时序图作了如下变更 <ul style="list-style-type: none"> 删除了 readdatavalid 信号, 当使用 waitrequest 时, 此信号是不相关的。 将编号 4, readdata 和 response 向前移到一个周期。 将 read 信号与编号 1 对齐。 扩展了 Transfers Using the waitrequestAllowance Property 部分。提供了更复杂的时序图。 更新了 Read Bursts 部分中的讨论。对于 burstcount > 1 的读操作, Intel 建议置位所有的 byteenables。 增强了 waitrequestAllowance Equals Two - Not Recommended 主题中的讨论。更正了时序图。从 clock cycle 11 开始, 数据必须被保持 2 个周期。
2017 年 11 月	17.1	作了如下变更: <ul style="list-style-type: none"> 更新了 Read Bursts 的讨论如下: <ul style="list-style-type: none"> 验证通过了陈述, "当一个 master 直接连接到一个 slave 时, 值为<n>的 burstcount 意味着此 slave 必须返回<n>个字的 readdata 才能完成突发。" 如果 master 直接连接到 slave, 那么此陈述为真。如果 interconnect 链接到 master 和 slave, 那么此陈述可以不为真。 从读突发的描述中删除了以下陈述: "通过 read burst 命令呈现的 byteenables 应用于突发的所有周期。" 此陈述不再为真。然而, Intel 建议使用 burstcount > 1 的读操作置位所有的 byteenables。 从 Pipelined Transfers 部分删除了以下陈述: 写传输不能被流水线化。您可以使用 writeresponsevalid 信号对写操作进行流水线化。 扩展了 Avalon -MM Read and Write Responses Timing Diagram 部分中的读和写响应的描述。 修改了 reset_req 信号的描述。 将 irq 的宽度从 1 bit 更改成 1-32 bits。Intel Quartus Prime Pro Edition 和 Intel Quartus Prime Standard Edition 软件都支持中断向量。

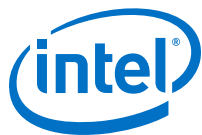
继续...

Intel Corporation. All rights reserved. Agilix, Altera, Arria, Cyclone, Enpirion, Intel, the Intel logo, MAX, Nios, Quartus and Stratix words and logos are trademarks of Intel Corporation or its subsidiaries in the U.S. and/or other countries. Intel warrants performance of its FPGA and semiconductor products to current specifications in accordance with Intel's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Intel assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Intel. Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.

*Other names and brands may be claimed as the property of others.



文档版本	Intel Quartus Prime 版本	修订内容
2017 年 5 月	Quartus Prime Pro v17.1 Stratix 10 ES Editions	作了如下变更： <ul style="list-style-type: none"> 增添了以下接口属性参数。 <ul style="list-style-type: none"> <code>waitrequestAllowance</code> 参数支持高速操作。此参数用于 Avalon-MM 接口。增添了使用此参数的时序图。 <code>minimumResponseLatency</code> 参数加速 Avalon-MM 接口的时序收敛。增添了使用此参数的时序图。
2015 年 12 月	15.1	作了以下变更： <ul style="list-style-type: none"> 将 <code>empty</code> 信号的宽度从最大 8 比特变更成最大 5 比特。 改进了 <code>reset_req</code> 信号的定义。 从 <i>Pipelined Read Transfer with Fixed Latency of Two Cycles</i> 时序图中删除了 <code>readdatavalid</code> 信号。对于固定延迟传输，此信号是无关的。 更正了定义 <code>empty</code> 信号的公式。 在 <i>Pipelined Read Transfers with Variable Latency</i> 时序图中作了如下变更： <ul style="list-style-type: none"> 将 <code>read</code> 信号的置低移到 cycle 9 在 cycle 9 中将 <code>waitrequest</code> 变更成 <code>don't care</code>。
2015 年 3 月	14.1	修复了 Figure 1-1 中的拼写错误。
2015 年 1 月	14.1	作了以下变更： <ul style="list-style-type: none"> 澄清了地址对齐实例。Avalon-MM master 和 slave 接口的宽度不同。 改进了 <i>Pipelined read Transfers with Variable Latency</i> 的讨论。更正了 <code>timing marker 2</code>，<code>timing marker 2</code> 应该正好在时钟的上升沿。 改进了 <i>Pipelined Read Transfer with Fixed Latency of Two Cycles</i> 的讨论。 澄清了 <code>beatsPerCycle</code> 属性的使用。 更正了 <code>line-wrapped bursts</code> 的地址范围。64-byte burst 的正确地址范围是 <code>0x0 - 0x3C</code>，而不是 <code>0x0 - 0x1C</code>。 在以下几个方面对 <i>Tristate Conduit Arbitration Timing</i> 图的描述作了更正： <ul style="list-style-type: none"> <code>tristate conduit slave</code> 置位 <code>grant</code>，而不是 <code>tristate conduit master</code> 置位 <code>grant</code>。 最后的 <code>grant</code> 出现在 cycle 9，而不是出现在 cycle 8。 增添了 <i>Deprecated Signals</i> 附录。 增添了 <code>read response</code> 信号。 改进了时钟和复位信号的定义。 更正了 <code>clock sink</code> 属性的定义。 更正了复位源接口的 <code>synchronousEdges</code> 的定义。 澄清了 Avalon-MM response 信号类型。 更新了 <code>empty</code> 的定义。此信号必须解释为 <code>emptyWithinPacket is true</code>。 针对清晰度和一致性进行的编辑。
2014 年 6 月	14.0	<ul style="list-style-type: none"> 更新了 Avalon-MM Signals 表，<code>begintransfer</code>，<code>readdatavalid</code> 和 <code>readdatavalid_n</code>。 更新了 Read and Write Transfers with Waitrequest 图： <ul style="list-style-type: none"> 将 <code>write</code> 的置低移到 cycle 6。 将 <code>readdatavalid</code> 和 <code>readdata</code> 的置位移到 cycle 4。 更新了 Pipelined Read Transfers with Variable Latency 图： <ul style="list-style-type: none"> 将 <code>data1</code> 的置位移到 cycle 5 之后，将 <code>data2</code> 的置位移到 cycle 6。 移动 <code>readdatavalid</code> 的置位以匹配 <code>data1</code> 和 <code>data2</code>。
2014 年 4 月	13.01	更正了 <i>Avalon Memory-Mapped Interfaces</i> 章节中的 <i>Read and Write Transfers with Waitrequest</i> 。
2013 年 5 月	13.0	进行了以下变更：
继续...		



文档版本	Intel Quartus Prime 版本	修订内容
		<ul style="list-style-type: none">• <i>Avalon Memory-Mapped Interfaces</i> 的次要更新。• <i>Avalon Streaming Interfaces</i> 的次要更新。• 更新了 <i>Avalon Conduit Interfaces</i> 来描述 Avalon 管道接口支持的信号角色。• 更新了 <i>Avalon Tristate Conduit Interface</i> 章节中的 <i>Shared Pin Types</i>。
2011 年 5 月	11.0	受 Qsys 支持的 <i>Avalon Interface Specifications</i> 的首次发布。