

本章节对 Altera® Qsys 系统集成工具生成的设计提供了优化系统互联性能的信息。

互联逻辑 (interconnect logic) 是所有大型系统的基础, 用于连接硬件模块或组件。创建互联逻辑容易出现的问题, 需要大量时间写入, 并且在设计要求变更时很难进行修改。Qsys 系统集成工具提供一个自动生成和优化的互联, 被设计成满足您的系统要求, 从而可以解决这些问题。

Qsys 支持标准的 Avalon®, AMBA® AXI3™ (1.0 版本), AMBA AXI4™ (2.0 版本) 和 AMBA APB™ 3 (1.0 版本) 接口。关于 Avalon 和 AMBA 接口的详细信息, 请参考 [Avalon Interface Specifications](#) 和 ARM® 网站上的 [AMBA Protocol Specifications](#)。不支持 AXI4-Lite。



关于确定使用哪种接口标准来创建您的 Qsys 设计的详细信息, 请参考 *Quartus II Handbook* 卷 1 中的 [Creating a System With Qsys](#) 章节。

本章中建议的以下设计实践有助于改善您的 Qsys 设计的时钟频率, 数据吞吐量, 逻辑使用或者功耗。当设计一个 Qsys 系统时, 除了 Qsys 提供的自动优化, 您还要根据您的设计意图和目标来进一步优化系统性能。

接下来的章节描述了对互联逻辑优化的 Qsys 支持:

- 第 10-1 页 “采用 Avalon 和 AXI 接口进行设计”
- 第 10-3 页 “在系统中使用层次结构”
- 第 10-4 页 “在存储器映射系统中使用并发机制”
- 第 10-8 页 “插入流水线级以增加系统频率”
- 第 10-9 页 “使用 Avalon 桥接”
- 第 10-20 页 “增加传输数据吞吐量”
- 第 10-25 页 “降低逻辑使用 (Reducing Logic Utilization)”
- 第 10-30 页 “降低功耗”
- 第 10-35 页 “设计实例”

采用 Avalon 和 AXI 接口进行设计

存储器映射接口 (memory-mapped interface) 的 Qsys Avalon 和 AXI 互联是连接主从接口的灵活的, 部分的交叉交换矩阵。

Avalon Streaming (Avalon-ST) 链路连接点到点单向接口, 通常用于数据流应用。每一对组件的连接都不需要数据源与数据接收 (data source and sink) 之间的仲裁。

由于 Qsys 支持复用的存储器映射和数据流连接, 因此您可以实现在单一设计中对控制使用复用逻辑和对数据使用数据流逻辑的系统。

关于设计数据流和存储器映射组件的详细信息，请参考 *Quartus II Handbook* 卷 1 中的 *Creating Qsys Components* 章节。

设计数据流组件

当设计数据流组件接口时，您必须考虑系统中每个组件的集成与通信。通常要考虑的一点是：从内部缓冲数据以适应组件间的延迟。例如，如果由于置低 `ready` 信号导致组件的 Avalon-ST 输出或者源的流数据被反压，那么此组件必须反压它的输入或者 sink 接口来避免上溢。

您可以在组件的输出侧使用 FIFO 来实现内部处理反压情况，这样即使输出端出现了反压，输入端仍能接受更多的数据。当 FIFO 有足够的剩余空间来满足内部延迟时，使用 FIFO 的 `almost full` 信号反压 sink 接口或输入数据。当数据可用时，通过 FIFO 的 `not empty` 标记驱动输出接口或源接口的数据有效信号。

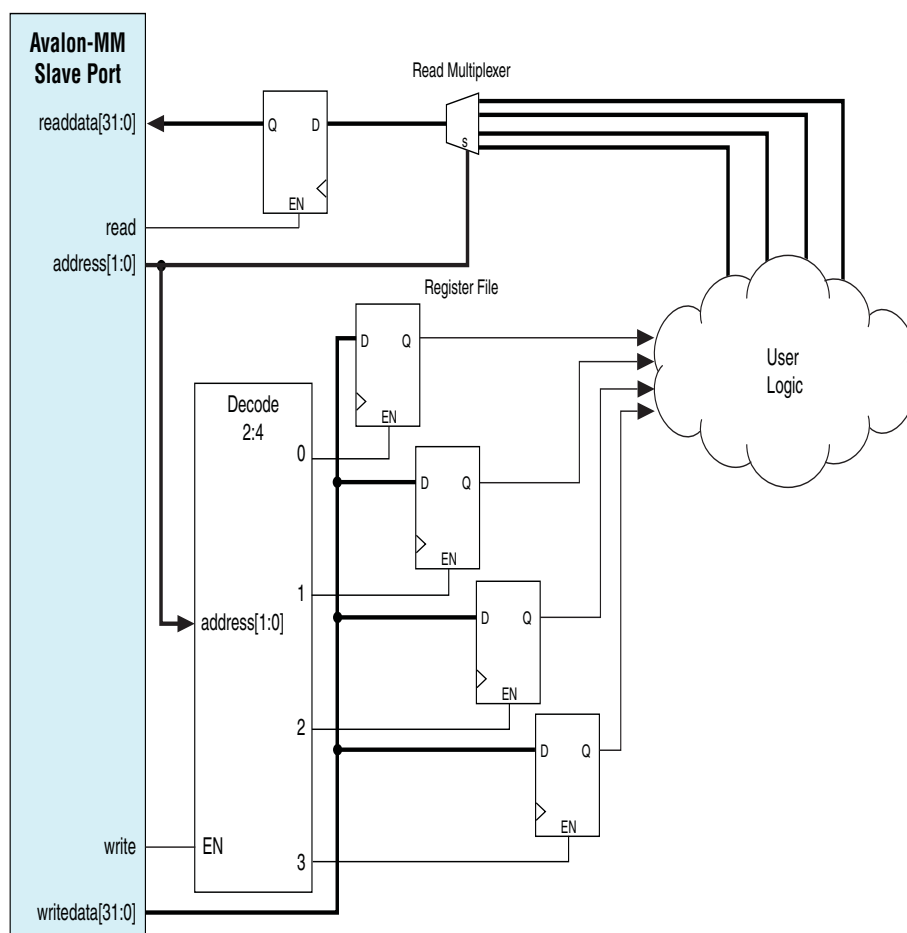
Quartus II 12.1 不提供 AXI 流和桥接组件。

设计存储器映射组件

采用存储器映射组件进行设计时，您可以使用第 10-2 页“从组件中控制和状态寄存器 (CSR) 的实例”实现任何包含映射到存储器位置的多个寄存器的组件。实现读写存储器映射传输的组件需要三个主要构建模块：地址解码器，寄存器文件和读复用器。

图 10-1 显示如何实现一组四个输出寄存器来支持从逻辑的软件读回。

图 10-1. 从组件中控制和状态寄存器 (CSR) 的实例



解码器使能相应的 32-bit 或 64-bit 寄存器用于写操作。对于读操作，地址位驱动复用器选择位。read 信号寄存来自复用器的数据，添加流水线阶段，使组件达到更高的时钟频率。此组件有写等待状态和一个读等待状态。或者，如果要想实现较高的数据吞吐量，那么需要将读和写等待状态都设为零，然后指定读延迟为 1，因为组件也支持流水线读操作。

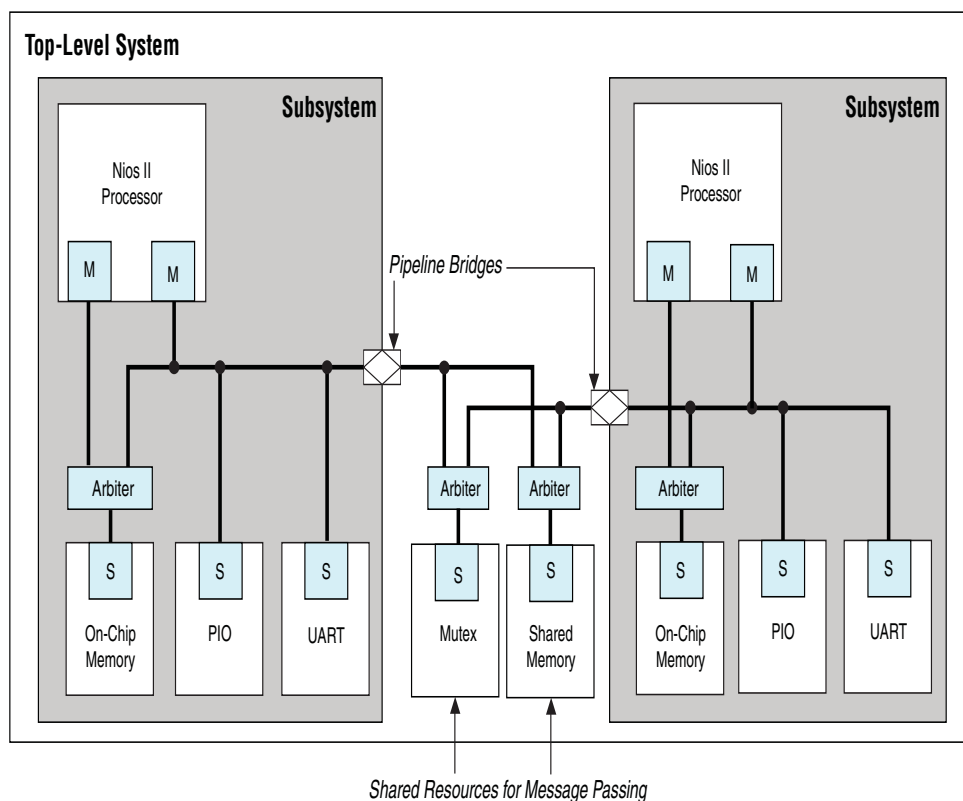
在系统中使用层次结构

使用层次结构可以将一个系统分成更小的子系统，这些子系统在顶层 Q_{sys} 系统中能够连接在一起。您可以使用层次结构来简化与存储器映射系统中每个主端口连接的从端口的验证控制。在您的设计中开始实现子系统之前，应该根据下面指南在顶层规划系统层次化模块：

- **规划共享资源** — 例如，确定共享资源在系统层次结构中的最佳位置。例如，如果两个子系统共享资源，那么您应该将使用这些资源的组件添加到更高层的系统中，这样更容易存取。
- **规划子系统之间的共享地址空间** — 规划地址空间确保对子系统之间的桥接设置正确的大小。
- **规划添加到您系统中的延迟量** — 在子系统之间添加流水线桥接时，可能要对整个系统添加更多的延迟。通过零周期延迟参数化流水线桥接可以降低添加的延迟。

图 10-2 显示了基于信息传递共享资源的两个 Nios II 处理器子系统的实例。每个子系统中的桥接导出 Nios II data master 到包含互斥组件 (mutual exclusion component) 和共享存储器组件 (可能是另一个片上 RAM, 或者用于片外 RAM 器件的控制器) 的顶层系统。

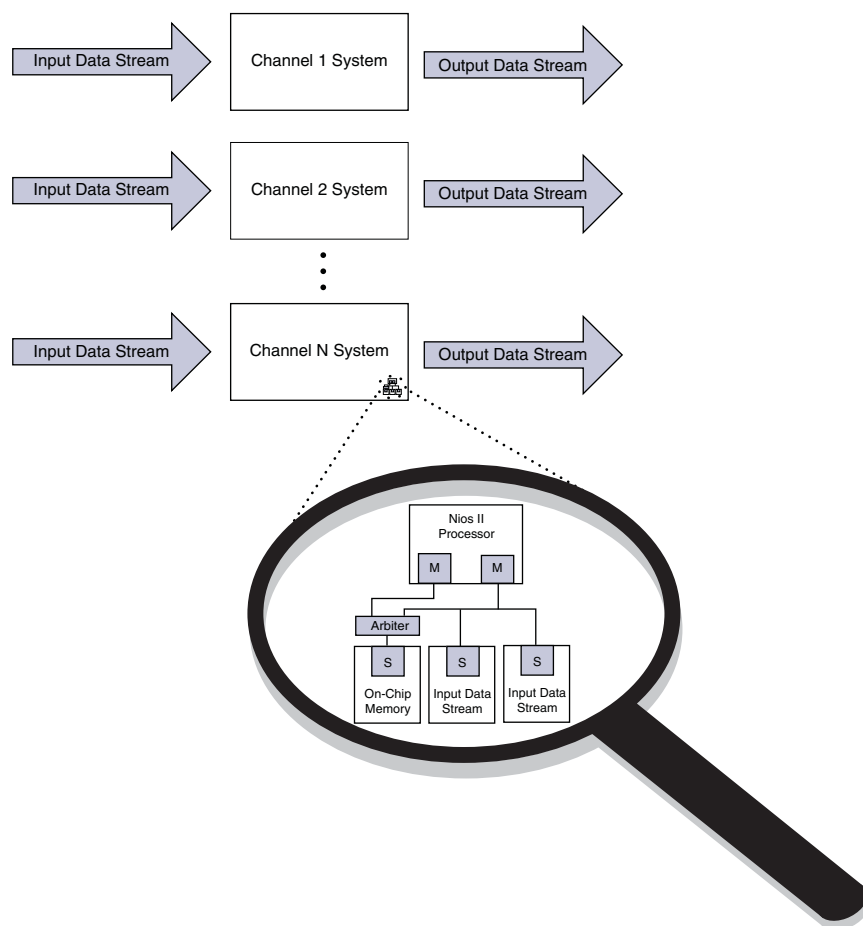
图 10-2. 子系统间的详细传递



如果一个设计包含一个或多个相同的功能单元，那么该功能单元能够定义为一个子系统，并且在顶层系统中能够被多次例化。通过对每个通道例化相同的子系统，您也可以设计用于处理多个数据通道的系统。与更大的非层次化的系统比较，这一方法更容易维护。此外，这样的系统更容易缩扩，因为通过子系统要求的倍数可以计算出所需资源。

图 10-3 显示了一个包含三个子系统的设计，每个子系统分别处理不同的通道。

图 10-3. 多通道系统



在存储器映射系统中使用并发机制

Qsys 互联使用 FPGA 中的并行硬件，使您能够将并行机制设计到您的系统中，并能够同时处理多个传输。接下来的部分描述了设计选择，这些设计选择能够增加您系统中的并行度。

创建多个主接口

实现并发机制要求系统中有多多个主接口。包含处理器的系统至少要包含两个主接口，因为处理器包含独立的指令和数据主接口。主组件分为以下几类：

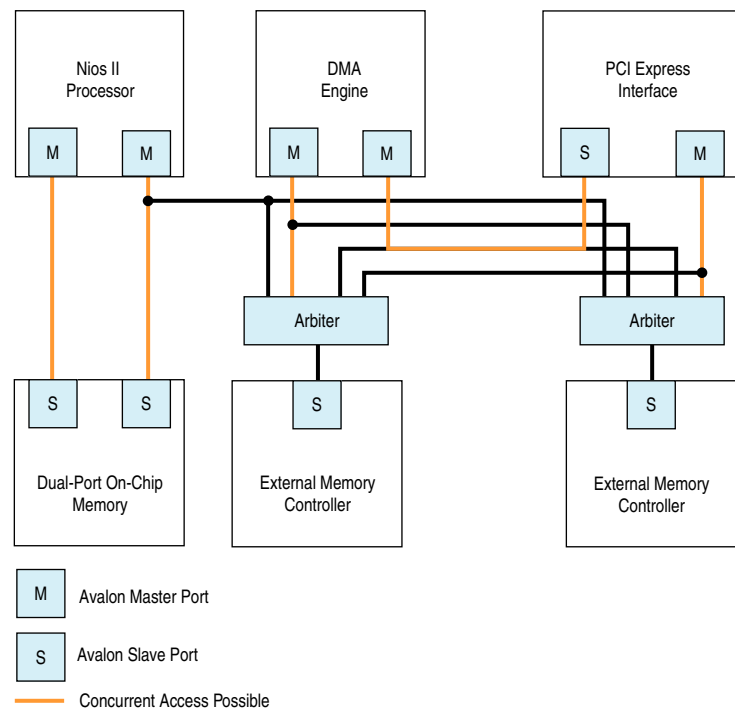
- 通用处理器，例如 Nios II 处理器
- DMA（直接存储器存取）引擎

■ 通信接口，例如 PCI Express

由于 Qsys 生成一个具有从端仲裁的互联，因此您系统中的每个主接口能够并行发出传输。系统中的主接口只要不将传输发送到同一从接口就能并行发出传输。并行机制受限于共享任意特定从接口的主接口的数量。如果您的设计要求更高的数据吞吐量，那么可以增加主从接口的数量来增加同时出现的传输数量。关于详细信息，请参考第 10-7 页“创建多个从接口”。

图 10-4 显示了具有三个主接口的系统，其中的连线是那些可同时有效 (active) 的连接实例。

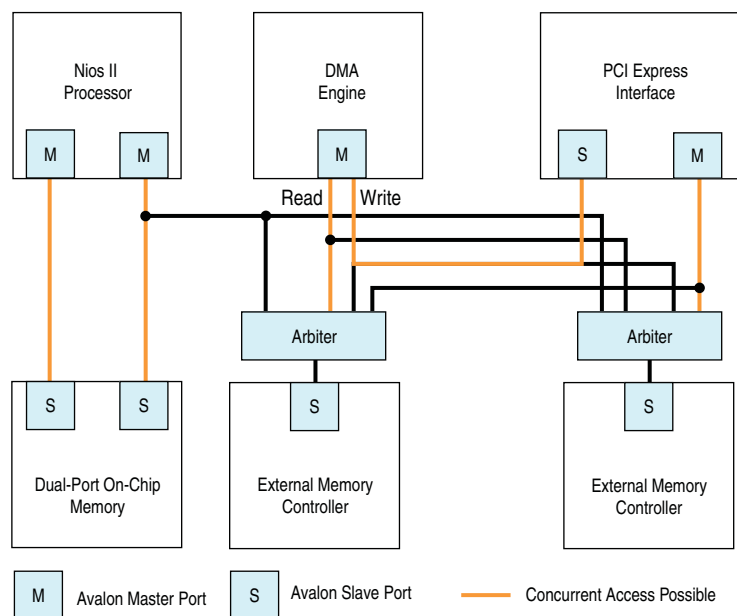
图 10-4. Avalon 多个主接口并行存取



在此 Avalon 实例中，DMA 引擎通过 Avalon-MM 读和写主接口运行。然而，AXI DMA 接口通常只有一个主接口，因为在 AXI 标准中主接口上的读写通道是独立的，并能够同时处理多个传输。

图 10-5 显示了一个 AXI 实例，其中 DMA 引擎通过单一主接口运行，因为在 AXI 标准中主接口上的读写通道是独立的，并能够同时处理多个传输。此实例显示了读通道与写通道之间的并发机制，黄线表示并发数据通路。

图 10-5. AXI 多个主接口并行存取

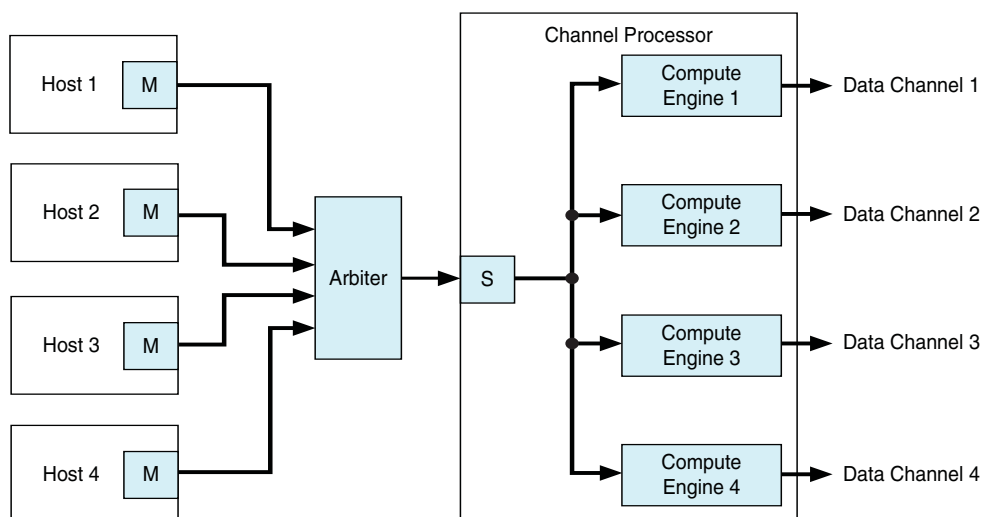


创建多个从接口

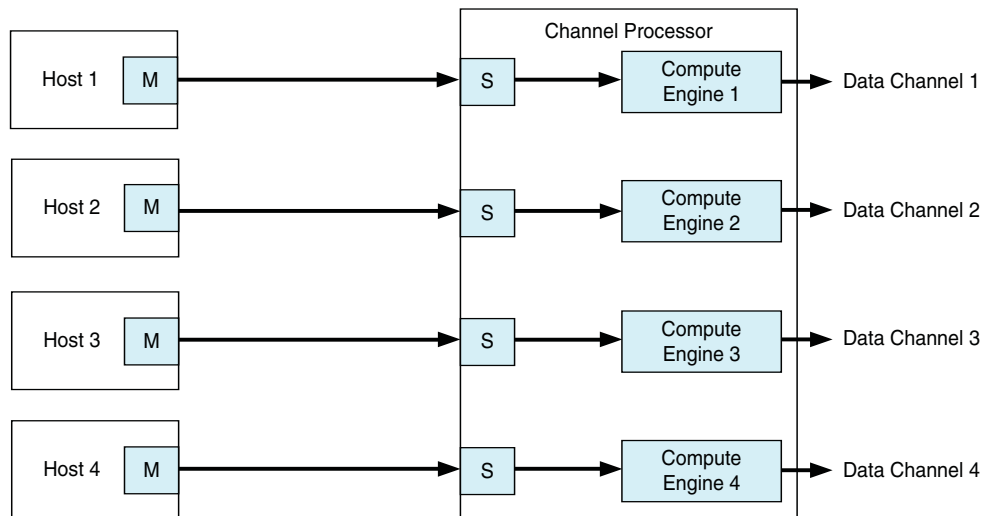
您可以对特定功能创建多个从接口以增加你设计中的并发度。图 10-6 显示两个通道处理系统。在第一个通道处理系统中，四个 host 必须对通道处理器的单一从接口仲裁。在第二个通道处理系统中，每个 host 驱动一个专用从接口，使所有主接口同时访问组件的从接口。当存在单一主接口和单一从接口时，仲裁不是必需的。

图 10-6. 单一接口 Vs 多个接口

Single Channel Access



Multiple Channel Access



使用 DMA 引擎

在某些系统中，您可以使用 DMA 引擎来增加数据吞吐量。您可以使用 DMA 引擎传输接口之间的数据模块，这样 CPU 就不用执行这一例行任务。DMA 引擎在没有干预的情况下传输编程的起始和终止地址之间的数据，与 DMA 连接的组件决定数据的吞吐量。影响数据吞吐量的因素包括数据位宽和时钟频率。图 10-7 显示了一个通过包含更多 DMA 引擎来进行更多的并行读写操作的系统，在顶层系统对读写缓存的访问可以在两个 DMA 引擎之间分开，如图底部的 Dual DMA Channels 系统中所示。

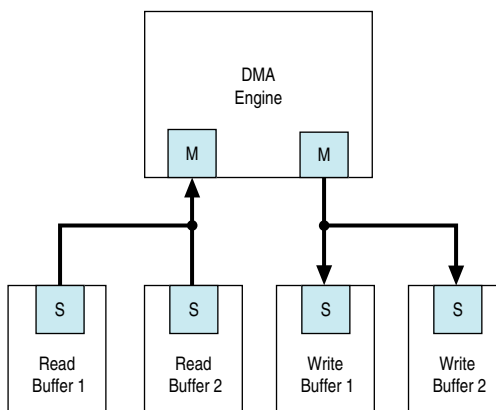


在此实例中，DMA 引擎同 Avalon-MM 读写主接口一起运行。AXI DMA 通常只有一个主接口，因为在 AXI 标准中主接口上的读写通道是独立的，并能够同时处理多个传输。

图 10-7. 单或双 DMA 通道

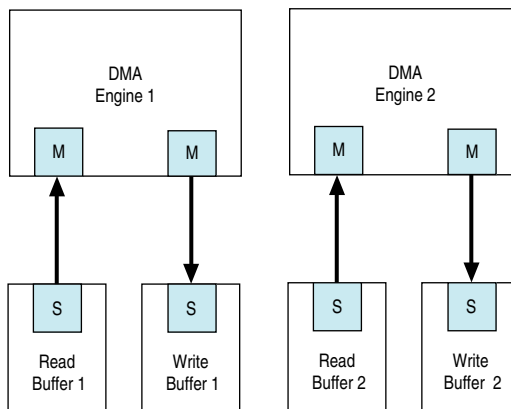
Single DMA Channel

Maximum of One Read & One Write Per Clock Cycle



Dual DMA Channels

Maximum of two Reads & Two Writes Per Clock Cycle



插入流水线级以增加系统频率

在 Qsys 中，Project Settings 标签上的 Limit interconnect pipeline stages to 选项用于生成您的系统时自动将流水线级添加到 Qsys 互联中。您可以指定 0 到 4 个流水线级，0 表示互联有一个组合数据通路。您可以分别对每个子系统指定不同的互联流水线级值。

添加流水线级可能会以额外的延迟和逻辑使用为代价，通过降低组合逻辑深度来增加您设计的 f_{MAX} 。

流水线级的插入需要某些互联组件。例如，在具有单一从接口的系统中没有多路复用器；因此不会出现多路复用流水线。当存在一个 Avalon 或 AXI single-master to single-slave 系统时，不管 **Limit interconnect pipeline stages to** 参数设置如何都不会出现流水线。



关于 **Limit interconnect pipeline stages to** 参数的详细信息，请参考 *Quartus II Handbook* 卷 1 中的 *Qsys Interconnect* 章节。

使用 Avalon 桥接

使用桥接可以提高系统频率，最小化生成的 Qsys 逻辑和适配器逻辑和当要控制 Qsys 添加流水线的位置时构建系统拓扑结构。当系统中存在并发机制时，您也可以使用桥接器和仲裁器。



AXI 桥接不支持 Quartus II 12.1；然而，您可以在 AXI 接口之间和 Avalon 域之间使用桥接。Qsys 自动创建 AXI 和 Avalon 接口之间的互联逻辑，因此不必明确例化这些域间的桥接。关于共享和分离域的利弊的详细信息，请参考 *Quartus II Handbook* 卷 1 中的 *Qsys Interconnect* 章节。

Avalon 桥接有一个 Avalon-MM 从接口和一个 Avalon-MM 主接口。可以有多个组件连接到桥接从接口，或者多个组件连接到桥接主接口，或者单一组件连接到单一桥接从接口或主接口。您可以配置桥接的数据位宽，这会影响 Qsys 如何在互联中生成总线规划逻辑。这两个接口都支持不同延迟的 Avalon-MM 流水线传输，同时也支持可配置突发长度。

桥接从接口上的传输被传播到连接到桥接的组件下游的主接口。当您需要对互联流水线有更多的控制时，您可以使用桥接，而不是使用 **Limit Interconnect Pipeline Stages to** 参数。

增加系统频率

在 Qsys 中，您可以通过互联流水线级或流水线桥接来提高您系统中的时钟频率。桥接控制系统互联拓扑结构，使您能够细分互联，具有对流水线和时钟交叉功能的更多控制。

插入流水线桥接

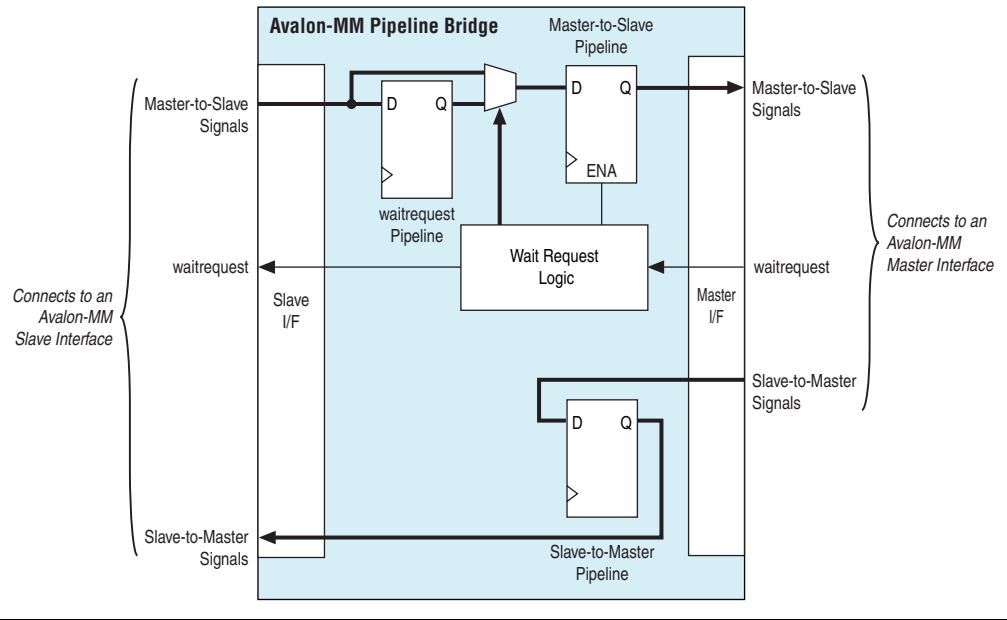
通过插入一个 Avalon-MM 流水线桥接可以在桥接与其主从接口间的路径中插入寄存器。如果在 Qsys 互联中出现关键寄存器到寄存器延时，那么流水线桥接能够帮助减少这一延时并增加系统 f_{MAX} 。

Avalon-MM 流水线桥接组件集成在所有的 Qsys 系统中。流水线桥接选项能够增加逻辑利用率和读延迟。如果多个主接口对桥接仲裁，那么拓扑结构的改变也可能会降低并发度。

您可以选择使用没有添加流水线级的 Avalon-MM 流水线桥接来控制拓扑结构。在某些延迟敏感的应用中，没有添加流水线级的流水线桥接是最佳的。例如，CPU 访问存储器时可能受益于最小延迟。

图 10-8 显示了 Avalon-MM 流水线桥接的体系结构。

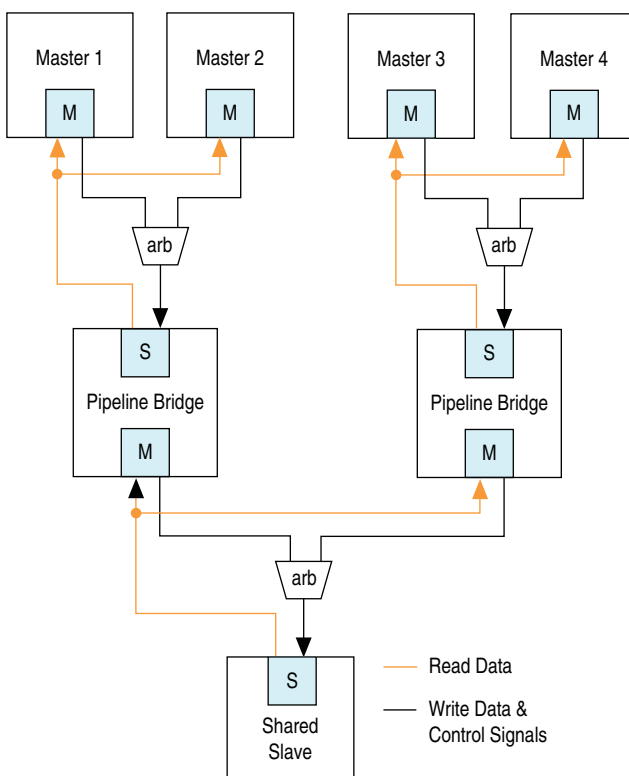
图 10-8. Avalon-MM 流水线桥接



实现命令流水线 (Implement Command Pipelining) (Master-to-Slave)

当多个主器件共享一个从器件时，要使用命令流水线来提高性能。从接口的仲裁逻辑必须多路复用 (multiplex) address, writedata 和 burstcount 信号。多路复用器宽度根据连接到单一从接口的主接口数量成比例增加。增加的多路复用器宽度可能会成为系统中的时序关键路径。如果一个流水线桥接没有提供足够的流水线，那么您可以例化树状结构中桥接的多个实例，以增加流水线并在从接口上进一步降低多路复用器宽度，如图 10-9 所示。

图 10-9. 桥接树 (Tree of Bridges)



响应流水线 (Response Pipelining) (Slave-to-Master)

一个系统中的主接口若连接到多个支持读传输的从接口，可受益于 slave-to-master pipelining。互联插入一个多路复用器用于每个读回主接口的数据通路。随着连接到主接口支持读传输的从接口数量的增加，读数据多路复用器的宽度也会增加。就 master-to-slave pipelining 来说，如果通过一个桥接性能提升的不够高，那么您可以在树形结构中使用多个桥接来提高 f_{MAX} 。

使用时钟交叉桥接 (Use Clock Crossing Bridges)

从接口的传输被传播到主接口。时钟交叉桥接包含一对时钟交叉 FIFO，将主从接口隔离在各自的异步时钟域中。

当对时钟域交叉使用 FIFO 时钟交叉桥接时，要添加数据缓冲。即使桥接的从接口下游不支持流水线传输，数据缓冲也能使流水线读主接口发出多个 read 到桥接。

独立的组件频率

您可以使用时钟交叉桥接在不同的时钟域中布置高频率组件和低频率组件。如果将快速时钟域限制在要求高性能的设计部分内，那么可能会达到此设计部分的更高 f_{MAX} 。

例如，嵌入式设计中包括的大多数处理器外设都不需要运行在高频上，因此您不需要对这些组件使用高频时钟。当使用 Quartus II 软件编译一个设计时，如果很难满足时钟频率要求就需要更长时间进行编译，因为 Fitter 需要更多的时间布置寄存器来达到所需的 f_{MAX} 。要减少 Fitter 在低优先权和低性能组件上的效力，您可以将这些组件布置在运行在较低频率上的时钟交叉桥接的后面，这样 Fitter 就会使用更多的效力在较高优先权和较高频率数据通路上。

最小化设计逻辑

通过降低 Qsys 生成的仲裁数量和多路复用器逻辑数量，桥接可以降低互联逻辑，这是因为桥接限制了能够出现的并发传输数量。接下来的部分介绍如何使用桥接来最小化由 Qsys 生成的逻辑。

避免增加逻辑的速度优化

通过主从接口之间的流水线桥接添加一个额外的流水线级会减少寄存器间的组合逻辑数量，这可以提供系统性能，如第 10-9 页“增加系统频率”中所述。

如果能增加设计逻辑的 f_{MAX} ，那么就能关闭 Quartus II 优化设置，例如 **Perform register duplication** 设置。寄存器复制 (register duplication) 创建复制的寄存器，被布置在 FPGA 中两个或更多的物理位置以减少寄存器到寄存器延时。对于优化方法，您也可能想选择 **Speed**，由于逻辑复用，这通常会产生更高的逻辑使用率。通过利用 Avalon-MM 桥接中寄存器或者 FIFO 可以增加设计速度和避免不需要的逻辑复制或者速度优化，从而减少设计的逻辑使用。

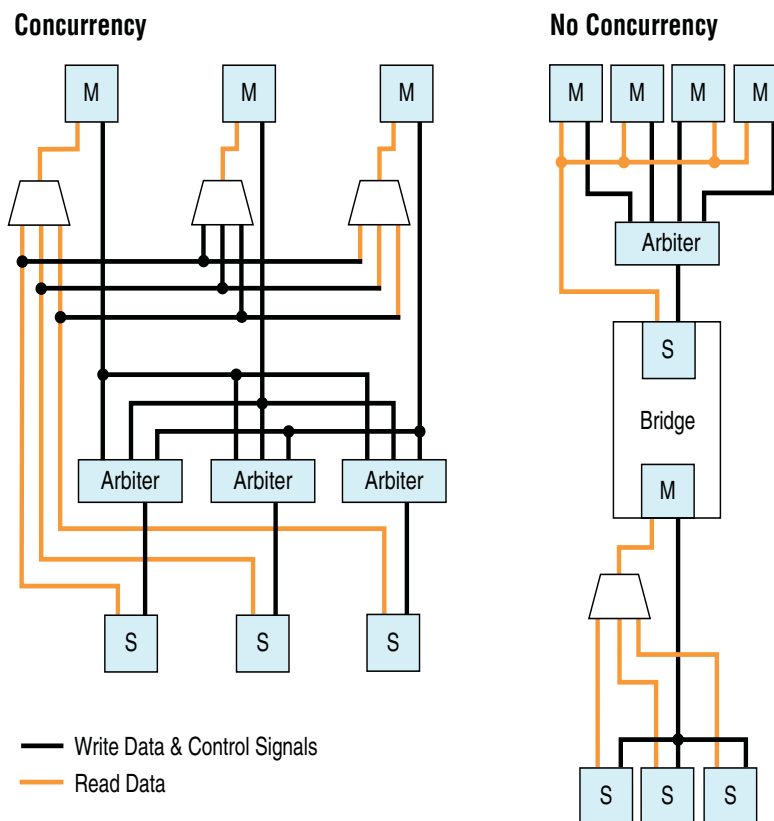
降低的并发性 (Reduced Concurrency)

由于 Qsys 对每个被多个主接口共享的从接口创建仲裁逻辑，因此随着系统不断变大，生成用于互联的逻辑数量也不断增加。Qsys 在连接到多个从接口的主接口之间插入多路复用器逻辑（如果两者都支持 read 数据通路）。大多数嵌入式处理器设计都包含能够支持高数据吞吐量的组件，或者包含不需要频繁存取的组件。这些组件能包含 Avalon-MM 主接口或者从接口。由于互联支持并发存取，因此您可以通过插入桥接到数据通路来限制生成的仲裁和多路复用器逻辑的数量来限制并发性。

例如，如果您的系统包含互联的三个主接口和三个从接口，那么 Qsys 会生成三个仲裁器和三个多路复用器用于读数据通路。如果这些主接口不需要太多数量的同步数据吞吐量，那么您可以通过连接这三个主接口到流水线桥接来降低设计消耗的资源。此桥接控制这三个从接口，并减少总线结构中的互联。Qsys 在桥接和三个主接口之间创建一个仲裁模块和在桥接和三个从接口之间创建一个读数据通路多路复用器，防止并发发生；与标准总线体系结构类似。不要对高吞吐量数据通路使用此方法来确保没有限制总体系统性能。

图 10-10 显示了包含流水线桥接和不包含流水线桥接的系统体系结构的差别。

图 10-10. Switch Interconnect to Bus



最小化适配器逻辑 (Minimizing Adapter Logic)

当主从接口对的时钟域，宽度或突发性能之间不匹配时，Qsys 生成适配器逻辑用于时钟交叉，宽度适用和突发支持。当主接口的最大突发长度大于从接口的主突发长度时，Qsys 创建突发适配器。适配器逻辑创建额外的逻辑资源，当您的系统包含连接到多个不共享相同特征的主接口时，这一点是至关重要的。通过在您的设计中布置桥接，可以降低由 Qsys 生成的适配器逻辑的数量。

桥接的高效布局 (Effective Placement of Bridges)

要确定高效的桥接布局，您应该首先分析您系统中的每一个主接口以确定连接的从器件是否支持不同的突发性能或者是否操作在不同的时钟域中。一个组件的最大 burstcount 作为组件的 HDL 文件中的 burstcount 信号可见。最大突发长度是 $2^{(\text{width}(\text{burstcount}) - 1)}$ ，如果 burstcount 宽度是四个比特，那么最大 burstcount 是八。如果没有 burstcount 信号出现，那么组件不支持突发，或者组件有一个为 1 的突发长度。

要确定系统是否在主从接口之间需要一个时钟交叉适配器，需要查看 Qsys 中主从接口旁边的 clock 列。对于主接口和从接口，如果时钟是不同的，那么 Qsys 会在它们之间插入一个时钟交叉适配器。为避免创建多个适配器，您需要将包含从接口的组件布置在桥接后面，从而只创建一个适配器。通过将具有相同突发或时钟特性的多个组件布置在桥接后面，可以限制并发性和适配器数量。

您可以使用桥接分离 AXI 和 Avalon 域，以最小化突发适应逻辑。例如，如果有多个 Avalon 从接口连接到一个 AXI 主接口，那么您可以考虑插入一个桥接，每次在桥接前访问适应逻辑，而不是在每个从接口前都有一个适应逻辑。每次在从接口存取适应逻辑。这将以延迟为代价，也可能会失去读写之间的并发性。

改变相应缓冲深度 (Changing the Response Buffer Depth)

当使用自动时钟交叉适配器时，Qsys 根据从接口属性决定所需的 FIFO 缓存深度。如果从接口有一个较高的 **Maximum Pending Reads** 参数，那么 Qsys 在主从接口之间插入的深响应缓存 FIFO 会消耗大量器件资源。要控制响应 FIFO 深度，需要使用时钟交叉桥接并手动调整它的 FIFO 深度，通过更少的存储器使用来权衡数据吞吐量。例如，如果您的主接口不能使从接口饱和，那么就不需要响应缓存，以便使用桥接降低 FIFO 存储器深度并降低从接口的 **Maximum Pending Reads**。

使用桥接带来的影响

在您的设计中使用流水线或时钟交叉桥接之前，必须认真考虑它们的影响。桥接对您的设计会产生正面或负面的影响。您可以在插入桥接之前和之后对您的系统进行基准测试以确定这些桥接带来的影响。接下来的部分讨论了添加桥接对系统可能造成的影响。

增加延迟

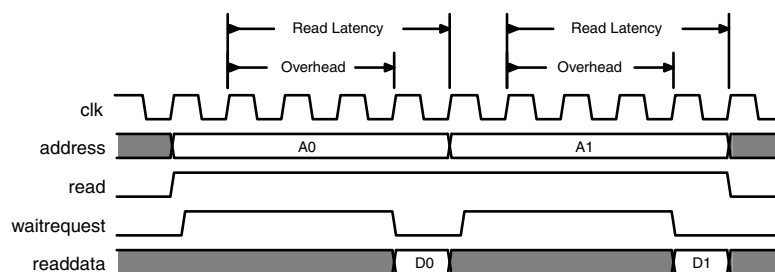
在您的设计中添加桥接会对主从接口之间的读延迟产生影响。该延迟在您的设计中可能是可接受的，也可能是不可接受的，这要取决于系统要求和主从接口的类型。

可接受的延迟增加

一个流水线桥接对每个使能的流水线选项添加一个周期的延迟。时钟交叉桥接中的缓冲也会添加延迟。如果您使用一个处理很多读传输的流水线或突发主接口，延迟的增加不会对性能产生显著的影响，因为与数据传输的长度相比，增加的延迟是非常小的。

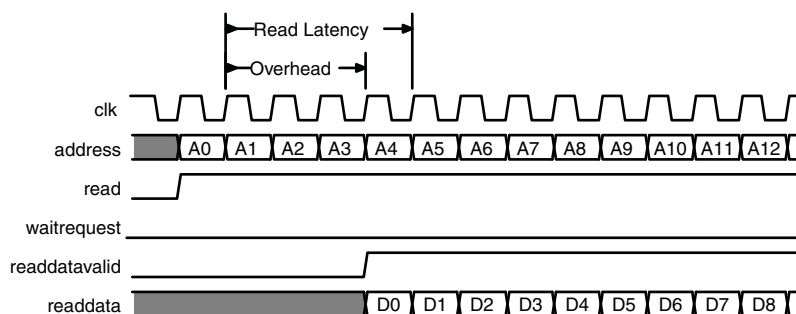
例如，如果使用一个流水线读主接口（例如 DMA 控制器）从四个时钟周期固定读延迟的组件读取数据，但仅执行一个字传输，那么开销 (overhead) 是四个时钟周期中的三个时钟周期，假设 Qsys 互联中没有其它的流水线延迟。读数据吞吐量仅为 25%。图 10-11 显示了这种低效读传输。

图 10-11. 低效读传输



然而，如果无中断传输 100 个字的数据，那么开销 (overhead) 是总共 103 个时钟周期中的 3 个时钟周期，相当于近似 97% 的读效率（如果互联中没有其它流水线延迟）。在此读路径中添加一个流水线桥接会增加额外两个时钟周期的延迟。此传输需要 105 个周期来完成，相当于近似 94% 的效率。尽管效率降低了 3%，但添加桥接可能会使 f_{MAX} 增加 5%，例如在此情况下，如果能够增加时钟频率，那么可能会增加总数据吞吐量。随着传输字数的不断增加，效率会逐渐提高至接近 100%，无论存在流水线桥接与否。图 10-12 显示了这种高效读传输。

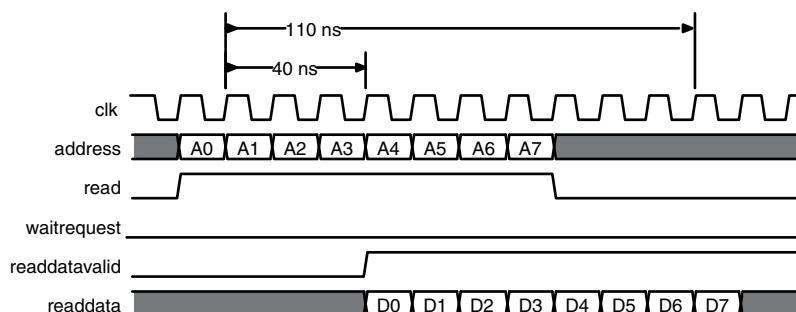
图 10-12. 高效读传输



不可接受的延迟增加

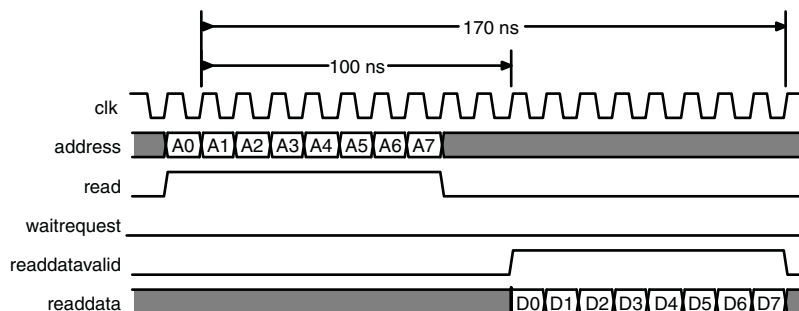
处理器对高延迟读时间敏感，通常将数据用于那些直到数据到达才能进行的运算。在添加一个桥接到处理指令或数据主接口的数据通路之前，要确定时钟频率的增加是否影响增加的延迟。图 10-13 显示了运行在 100 MHz 的 Nios II 处理器和存储器的性能。Nios II 处理器指令主接口有一个四个周期读延迟的高速缓存，每次读操作都会返回 8 个有序字。在 100 MHz 上，第一个读操作要用 40 ns 来完成。接下来的每个字用 10 ns 来完成，所以 8 个字总共用 110 ns 来完成。

图 10-13. 处理器系统：四个周期延迟的八个读操作



在此实例中，添加一个时钟交叉桥接使存储器运行在 125 MHz 上。然而，由于图 10-14 中所示的原因，设计上频率的提高导致延迟的增加而带来了负面影响。如果时钟交叉桥接添加 100 MHz 的六个时钟周期的延迟，那么存储器继续四个时钟周期的读延迟操作；因此，从存储器的第一个读操作使用 100 ns，由于读操作以处理器的工作频率 100 MHz 工作，每个连续字使用 10 ns。总共八个读操作使用 170 ns 完成。尽管存储器运行在一个更高的时钟频率，但主接口运行的频率却限制了数据吞吐量。

图 10-14. 处理器系统：10 个周期延迟的 8 个读操作



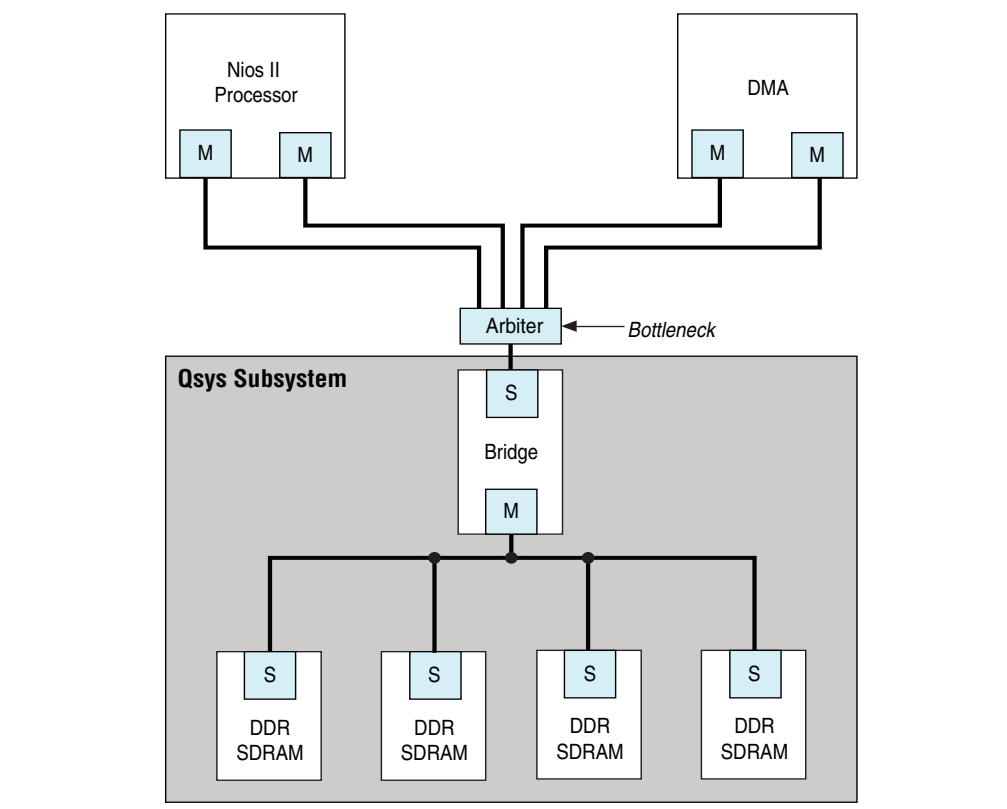
受限的并发性

在多个主从接口之间放置桥接会限制您的系统能够发启的并发传输的数量。这一限制与多个主接口连接到同一个从接口的情况相同。器件的从接口被所有主接口共享，因此 Qsys 创建仲裁逻辑。如果布置在桥接后面的组件不经常被访问，那么该并发性限制或许是可接受的。

如果使用不当，桥接会对系统性能产生负面影响。例如，如果多个存储器被若干个主接口使用，那么不应将存储器组件布置在桥接后面。桥接通过防止并发存储器访问来限制存储器性能。将多个存储器组件布置在桥接后面会导致这些独立的从接口对于访问桥接的主接口是以一个大存储器呈现；所有的主接口必须存取相同的从接口。

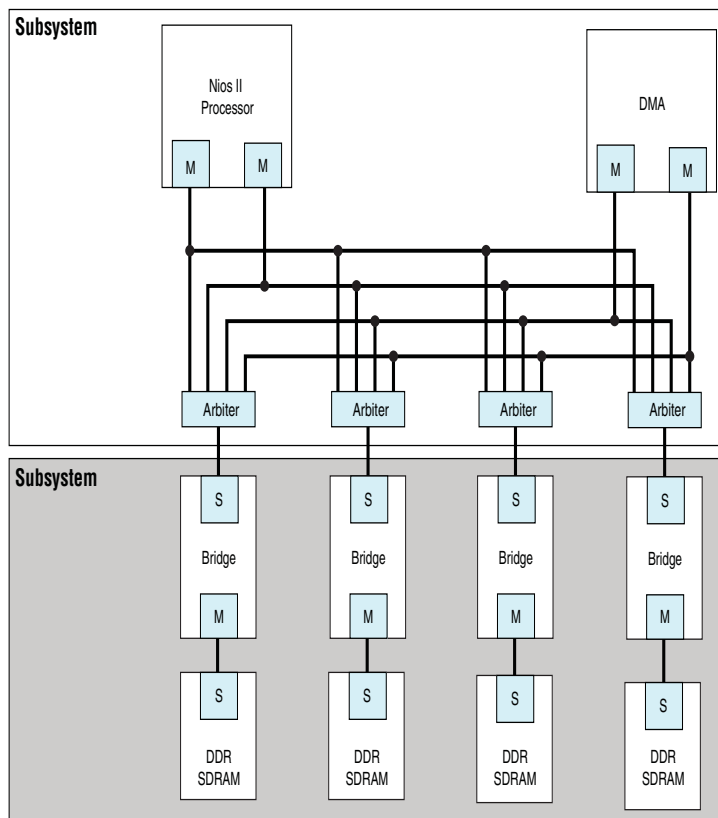
图 10-15 显示了一个存储器子系统，其中包括一个用于 Avalon-MM Nios II 和 DMA 主接口的桥接，作为单一从接口使用，从而形成一个瓶颈体系结构。桥接是两个主接口和存储器之间的瓶颈。AXI DMA 通常只有一个主接口，因为在 AXI 标准中主接口上的读写通道是独立的，能够同时处理传输。

图 10-15. 层次结构系统中桥接的不恰当使用



如果您存储器接口的 f_{MAX} 较低，并且想在子系统之间使用流水线桥接，那么您可以将每个存储器布置在其各自桥接的后面，这样便在不损并发性地情况下增加系统的 f_{MAX} ，如图 10-16 所示。

图 10-16. 层次结构系统中无瓶颈的高效存储器流水线



地址空间转换 (Address Space Translation)

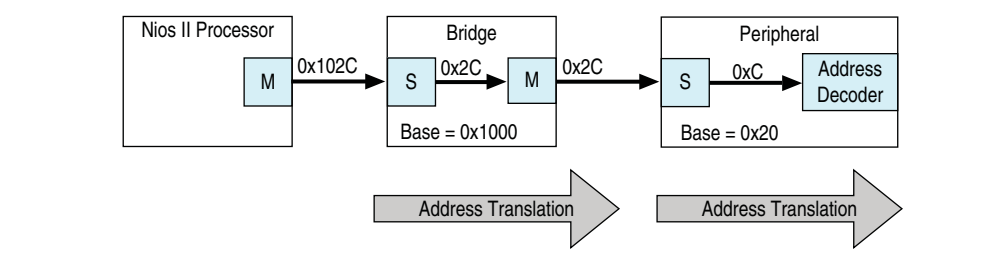
流水线的从接口或者时钟交叉桥接都有一个基地址和地址区段 (address span)。您可以设置基地址，或者让 Qsys 自动设置。从接口的地址是所有连接到桥接的组件的基偏移地址 (base offset address)。连接到桥接的组件的实际访问地址是该组件基偏移地址和组件的地址的和。

地址移位 (Address Shifting)

桥接的主接口在访问与其相连的从接口时，只驱动从接口的偏移地址所需要的地址线。每当主接口是通过桥接访问从接口时，必须把桥接组件的基偏移地址加上从接口的偏移地址作为实际访问地址，否则会传输失败。Qsys 中的 **Address Map** 标签显示连接到每个主接口的从接口的地址，并包括由系统桥接导致的地址转换。

图 10-17 显示如何进行地址转换。在此实例中，Nios II 处理器连接到位于基地址 0x1000 的桥接，从接口连接到 0x20 偏移上的桥接主接口，处理器对子接口中的第四个 32-bit 或 64-bit 字执行写传输。Nios II 驱动地址 0x102C 到互联，该地址在桥接地地址范围内。桥接主接口驱动 0x2C，该地址在从接口地址范围内，传输完成。

图 10-17. Avalon 桥接地址转换

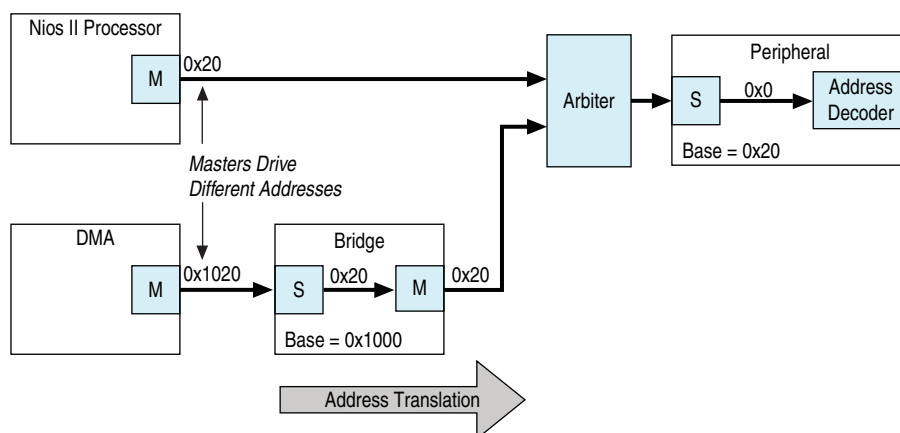


地址一致性 (Address Coherency)

要简化系统设计，所有主接口应该在同一地址访问从接口。在很多系统中，处理器传递缓存位置到其它控制组件，例如 DMA 控制器。如果处理器和 DMA 控制器不在同一位置访问从接口，那么 Qsys 必须对差异进行补偿。

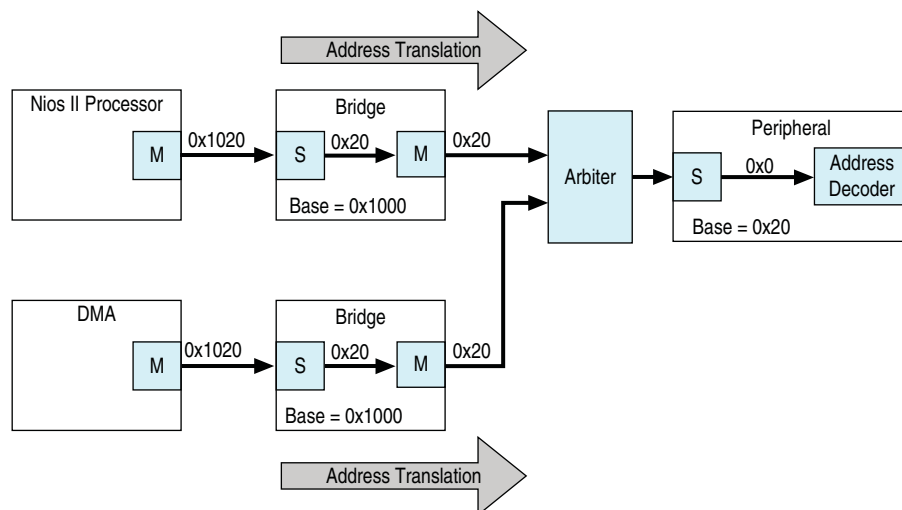
在图 10-18 中，Nios II 处理器和 DMA 控制器访问位于地址 0x20 的从接口。该处理器直接连接到从接口。DMA 控制器连接到位于地址 0x1000 的流水线桥接，然后连接到从接口。由于 DMA 控制器首先访问流水线桥接，因此它必须驱动 0x1020 来访问从接口的第一个位置。由于处理器从不同的位置访问从接口，因此您必须对从接口保持两个基地址。

图 10-18. 不同地址上的从接口，复杂化软件



要避免对两个地址的要求，您可以在系统中多添加一个桥接，将其基地址设为 0x1000，然后禁用第二个桥接中的所有流水线选项，这样桥接就会对系统时序和资源利用产生最小影响。由于第二个桥接的基地址与原始桥接的基地址相同，因此 DMA 控制器连接到处理器以及 DMA 控制器，并访问具有相同地址范围的从接口，如图 10-19 所示。

图 10-19. 通过桥接的地址转换纠正



增加传输数据吞吐量

提高系统中主从接口的传输效率将会提高您系统的数据吞吐量。对成本或功耗有着严格要求的设计受益于传输效率的提高，因为接下来可以使用成本和频率更低的器件。要求高性能的设计也受益于提高的传输效率，因为提高的效率会改善频率受限硬件的性能。

数据吞吐量是能够在特定时钟周期内传输的数据符号数（例如字节数）。读延迟是一个传输的地址和数据阶段之间的时钟周期数。例如，值为 2 的读延迟表示地址传输两个周期之后数据是有效的。如果主接口必须等待一个请求完成后才能发起下一个请求，例如处理器，那么读延迟对总的吞吐量是非常重要的。

通过观察仿真的波形可以测量数据吞吐量和延迟，或者使用验证 IP 监控器。关于详细信息，请参考 Altera 网站上的 *Avalon Verification IP Suite User Guide* 或者 *Mentor Graphics AXI Verification IP Suite - Altera Edition*。

使用流水线的传输 (Using Pipelined Transfers)

通过使主接口在早前读数据返回前发出多个读操作，流水线传输能提高读效率。

支持流水线传输的主接口连续发出传输，依赖于 `readdatavalid` 信号指示有效数据。Avalon-MM 从接口通过包含 `readdatavalid` 信号或者通过固定读延迟运行来支持流水线传输。

AXI 主接口通过 `writeIssuingCapability` 和 `readIssuingCapability` 参数来表明它能够发出的读写操作的数量。同样，一个从接口通过 `readAcceptanceCapability` 参数能够表明它能接受的读数量。

具有大于 1 的读发出能力的 AXI 主接口以与 Avalon 主接口和 `readdatavalid` 信号相同的方式被流水线化。

使用最大待定读参数 (Using the Maximum Pending Reads Parameter)

如果通过支持可变延迟读 (variable-latency reads) 的从接口创建一个定制组件，那么必须在 Component Editor 中指定 **Maximum Pending Reads** 参数。Qsys 使用 **Maximum Pending Reads** 参数生成相应的互联，并代表您的流水线从组件能够处理的最大数量的读传输。如果对从接口发起的读操作数量超过 **Maximum Pending Reads** 参数，那么从接口必须置位 `waitrequest`。

优化 **Maximum Pending Reads** 参数值需要您对您的定制组件的延迟有一个很好的理解。此参数应该基于组件中各种逻辑路径的组件最高读延迟。例如，如果您的流水线组件有两种模式，一种要求两个时钟周期而另一种要求五个时钟周期，那么要将 **Maximum Pending Reads** 参数设为 5，这样您的组件就能够流水线化 (pipeline) 五个传输，消除了因首次访问过程中的 5 个周期延迟所造成的浪费。

您也可以通过监控系统仿真期间或者运行硬件时的待定读数量来确定 **Maximum Pending Reads** 参数的正确值。要使用此方法，需要将 **Maximum Pending Reads** 设为一个较高的值，并使用一个主接口在每个时钟上发出读请求。您可以使用 DMA 来完成此任务，只要数据能够在 `waitrequest` 不被频繁置为有效的情况下被写到一个位置。如果用硬件实现这一方法，那么可以通过逻辑分析器或者内置监控硬件观测您的组件。

对您的定制流水线读组件的 **Maximum Pending Reads** 参数选择正确的值是非常重要的。如果低估了 **Maximum Pending Reads** 值，那么可能导致主接口被 `waitrequest` 信号阻塞，直到从接口响应之前的读请求并释放一个 FIFO 位置。

对于每个连接到从接口的主接口，**Maximum Pending Reads** 参数控制插入到互联的响应 FIFO。该 FIFO 使用少量硬件资源。高估您定制组件的 **Maximum Pending Reads** 参数会导致硬件使用的略微增加。基于这些原因，如果您不能确认最佳值，那么就应该高估此值。

如果您的系统包括桥接，那么必须也对此桥接设置 **Maximum Pending Reads** 参数。要允许最大数据吞吐量，此值应该大于或等于具有最高值的连接从接口的 **Maximum Pending Reads** 值。如第 10-14 页 “[改变相应缓冲深度 \(Changing the Response Buffer Depth\)](#)” 中所述，您可以限制从接口的最大待定读数，并通过降低桥接的参数值来降低缓存深度（如果不要求高数据吞吐量）。如果您不了解从组件的 **Maximum Pending Reads** 值，那么可以监控系统仿真期间或者运行硬件时的待定读数量。要使用此方法，需要将 **Maximum Pending Reads** 参数设成一个较高的值，并使用主接口在每个时钟上发出读请求，例如 DMA。然后，降低桥接的最大待定读数直到桥接降低访问此桥接的任意主接口的性能。

仲裁共享和突发 (Arbitration Shares and Bursts)

仲裁共享提供对仲裁进程的控制。默认情况下，仲裁算法分配均匀，所有主接口都接收一个机会。

通过对需要更大数据吞吐量的主接口分配一个较大的共享值，您可以对您的系统要求调整仲裁进程。仲裁共享值越大，分配给主接口访问从接口的传输就越多。只要主接口在传输数据（读或写），主接口就会对从接口进行共享数量的无中断访问。

如果一个主接口不能发出传输，并且其它主接口正等待获得对特定从接口的访问权限，那么仲裁器会授予另一个主接口访问权限。这种机制可以防止主接口在不能发出背靠背传输时浪费仲裁周期。突发传输包括多个 beat（或字）的数据，开始于一个单一地址。突发允许一个主接口保持对从接口的多个单字传输的访问。如果一个突发主接口发出一个突发长度为 8 的写传输，那么对于 8 个写周期它是被保证的仲裁。

您可以对 Avalon-MM 突发主接口和 AXI 主接口（一直被当作是突发主接口）分配仲裁共享。每个共享包括一个突发传输（例如多周期写），并允许主接口在仲裁切换到下一个主接口之前完成一组突发。

 关于仲裁共享和突发的详细信息，请参考 *Avalon Interface Specifications*，或者 ARM 网站上的 *AMBA Protocol Specification*。

仲裁共享与突发之间的差别

从以下三个主要方面区分仲裁共享和突发：

- 仲裁锁定 (arbitration lock)
- 按序寻址 (sequential addressing)
- 突发适配器 (burst adapters)

仲裁锁定 (Arbitration Lock)

当主接口发出一个突发传输时，仲裁对该主接口锁定；因此，此突发主接口应该能够在锁定期间维持传输。如果在第四个写之后主接口置低写信号 (Avalon-MM write 或 AXI wvalid) 50 个周期，那么其它主接口在此挂起期间继续等待访问。

要避免浪费带宽，在请求对从器件的访问前，您的主器件设计应该等待，直到一个全突发传输准备好。或者，通过对 burstcounts 分配一个与可用数据量相等的值来避免浪费带宽。例如，如果使用最大值 8 的 burstcount 创建一个定制突发写主接口，但数据中只有三个字可用，那么可以把 burstcount 置为 3。如果从接口能够处理更大的突发，那么该策略不会产生系统带宽的最佳利用。然而，该策略可以防止挂起，并支持对系统中其它主接口的访问。

Avalon-MM 按序寻址 (Avalon-MM Sequential Addressing)

一个 Avalon-MM 突发传输包括一个基地址和一个 burstcount。burstcount 代表要传输的一组字的数量，从基地址开始并按序递增。对于处理器，DMA 和缓存处理加速器，突发传输是常用的；然而，有时主接口必须访问非有序地址。因此，突发主接口必须将 burstcount 设为有序地址的数量，然后复位 burstcount 用于下一个位置。

仲裁共享算法没有地址限制；因此，您的定制主接口能够在每个读或写传输更新出现在互联的地址。

 AXI 的突发类型不同于 Avalon 接口。关于 AXI 突发类型的详细信息，请参考 *Quartus II Handbook* 卷 1 中的 *Qsys Interconnect* 章节和 ARM 网站上的 *AMBA AXI Protocol Specification*。

突发适配器 (burst Adapters)

Qsys 使您能够创建突发和非突发主从接口混合的系统。该设计策略旨在连接支持最大突发长度的突发主接口和突发从接口，Qsys 在适当的时候生成突发适配器。

每当主接口的突发长度超过从接口的突发长度时，或者主接口发出一个从接口不支持的突发类型时，Qsys 就会插入一个突发适配器。例如，如果 AXI 主接口连接到 Avalon 从接口，则插入一个突发适配器。

Qsys 对非突发主从接口分配一个值为 1 的突发长度。突发适配器将一个较长的突发分成若干个较短的突发。因此，突发适配器将逻辑添加到地址和主从接口之间的 burstcount 路径上。

选择 Avalon-MM 接口类型

要避免低效的 Avalon-MM 传输，地址主接口或从接口必须使用正确的简单接口，流水线接口或者突发接口。这三种可能的传输类型描述如下：

简单的 Avalon-MM 接口 (Simple Avalon-MM Interfaces)

简单的接口传输不支持读写的流水线或者突发；因此，它们的性能是受限的。简单接口适用于主接口和不常使用的从接口之间的传输。在 Qsys 中，PIO、ART 和 Timer 包括使用简单传输的从接口。

流水线 Avalon-MM 接口 (Pipelined Avalon-MM Interfaces)

流水线读传输允许流水线主接口连续执行多个读传输，而不需要等待前一个传输完成。流水线传输使主从接口对能够达到更高的数据吞吐量，即便从端口可能需要一个或多个周期的延迟来对每次传输返回数据。

在很多系统中，如果使用简单读传输并且流水线传输能够提高数据吞吐量，那么读数据吞吐量会变得不充分。如果用一个固定的读延迟定义组件，那么 Qsys 会自动提供必要的流水线逻辑来支持流水线读传输。Altera 建议使用固定的延迟流水线作为从接口的默认设计起始点。如果您的从接口有一个可变的延迟响应时间，那么要使用 readdatavalid 信号来指示有效数据何时可用。互联实现读响应 FIFO 缓冲以处理最大数量的待定读请求。

要使用支持流水线读传输的组件，并有效使用流水线系统互联，您的系统就必须包括流水线主接口。关于流水线读主接口的实例，请参考第 10-35 页 “Avalon 流水线读主接口实例”。Altera 建议使用流水线主接口作为新的主组件的默认起始点。对这些主接口使用 readdatavalid 信号。

由于主接口和从接口常有失配的流水线延迟，因此互联通常包含用于调解此差异的逻辑。表 10 - 1 显示了可能的流水线延迟情况。

表 10 - 1. 主从接口对 (master-slave pair) 中的各种流水线延迟情况 (1/2)

主接口	从接口	流水线管理逻辑结构
无流水线 (no pipelene)	无流水线 (no pipelene)	Qsys 互联不例化逻辑来处理流水线延迟。
无流水线 (no pipelene)	使用固定或可变延迟流水线化 (pipelined with fixed or variable latency)	Qsys 互联强制主接口在任意的 slave-side 延迟周期中等待。该主从接口对不受益于流水线，因为主接口要在新传输开始之前等待每个传输完成。然而，在主接口等待的同时，从接口能够接受来自不同主接口的传输。

表 10 - 1. 主从接口对 (master-slave pair) 中的各种流水线延迟情况 (2/2)

主接口	从接口	流水线管理逻辑结构
流水线的 (pipelined)	无流水线 (no pipeline)	如果 Qsys 互联就会以主从接口都没有被流水线化的方式进行传输，从而导致主接口在从接口返回数据前一直等待。一个非流水线从接口的实例是异步片外接口。
流水线的 (pipelined)	使用固定延迟流水线化 (pipelined with fixed latency)	Qsys 互联允许主接口在从接口的数据有效时在指定时钟周期上采集数据以使能最大数据吞吐量。一个固定延迟的实例是片上存储器。
流水线的 (pipelined)	使用可变延迟流水线化 (pipelined with variable latency)	从接口的 readdata 有效时置位一个信号，主接口采集数据。如果从接口有可变的延迟，那么 master-slave pair 能达到最大数据吞吐量。可变延迟从接口的实例包括 SDRAM 和 FIFO 存储器。

突发 Avalon-MM 接口 (Burst Avalon-MM Interfaces)

突发传输常用于例如 SDRAM 的存储器和例如 PCI Express 的片外通信接口。要高效使用能突发 (burst-capable) 的从接口，就必须连接到一个突发主接口。那些需要突进行高效操作的组件通常有一个与短突发或非突发传输相关联的开销惩罚 (overhead penalty)。

如果您知道您的组件需要按序传输进行高效操作，那么 Altera 建议您设计一个能突发的从接口。因为 SDRAM 存储器切换 bank 或 row 时会遭遇惩罚，所以当使用突发按序访问 SDRAM 存储器时会提高性能。

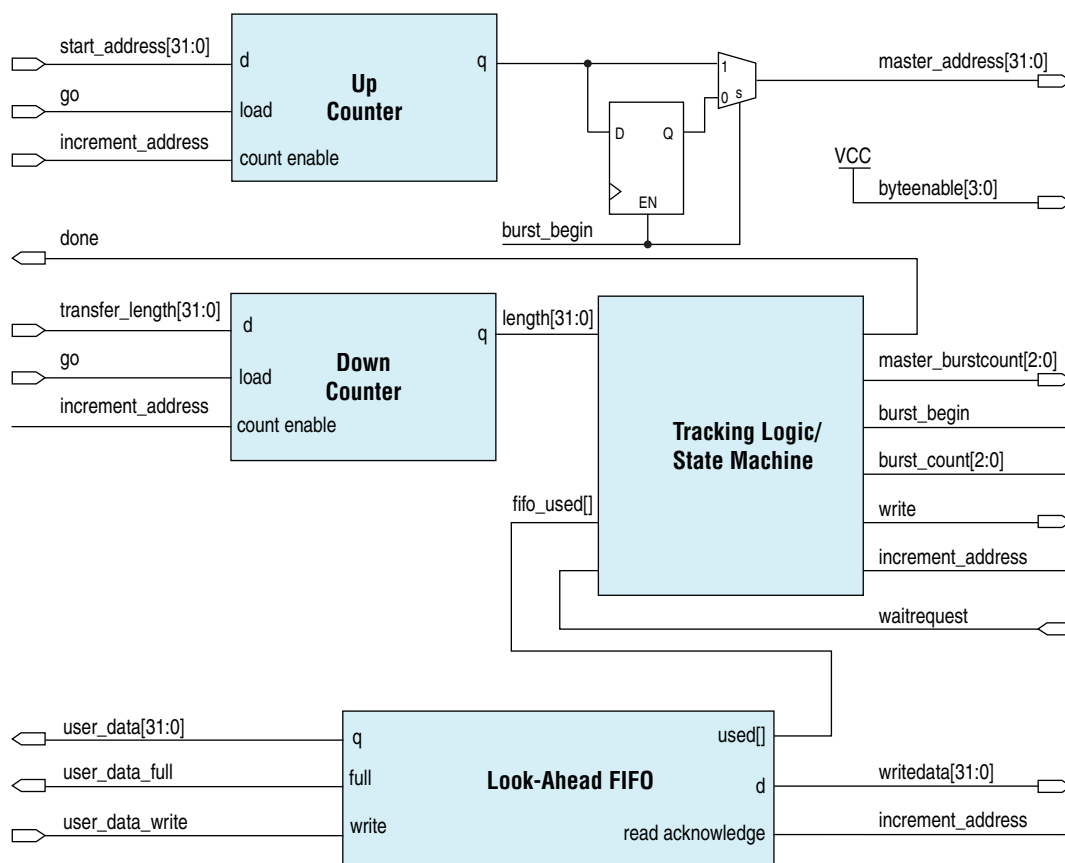
使用相同信号传输地址和数据的体系结构同样受益于突发。每当通过共享地址和数据信号传输地址时会降低数据传输的吞吐量。由于地址阶段会增加开销，因此使用较大的突发会提高连接的吞吐量。

Avalon-MM 突发主接口实例 (Avalon-MM Burst Master Example)

图 10-20 显示了突发写主接口的体系结构，该突发写主接口接收来自 FIFO 的数据并将数据写入存储器。您可以使用此主接口作为您自己突发组件的起始点，例如定制 DMA，硬件加速器或者片外通信接口。在图 10-20 中，主接口执行字存取并写入到按序存储器位置。

关于图 10-20 中实例的详细信息，请参考 Altera 网站上的 *Avalon Memory-Mapped Master Templates* 中的写主接口设计。

图 10-20. Avalon 突发写主接口



当 `go` 被置位时，`start_address` 和 `transfer_length` 被寄存。在下一个时钟周期上，控制逻辑置位 `burst_begin`。除了出现在互联的 `master_address` 和 `master_burstcount`，`burst_begin` 信号也与内部控制信号同步。这两个信号的时序是非常重要的，因为在突发写传输期间 `address`，`byteenable` 和 `burstcount` 必须在整个突发过程中保持不变。

要避免低效写传输，主接口要在足够的数据被缓存进 FIFO 中时仅发出一个突发。要最大化突发效率，主接口要只在从接口置位 `waitrequest` 时挂起。在此实例中，FIFO 的 `used` 信号跟踪存储在 FIFO 中的数据字数，并确定何时缓存进足够的数据。

`address` 寄存器在每个字传输后递增，`length` 寄存器在每个字传输后递减。整个突发过程中地址保持不变。由于一个突发不能保证每个突发边界的完成，需要额外的逻辑来识别短突发的完成和完成传输。

降低逻辑使用 (Reducing Logic Utilization)

这一部分介绍如何最小化 Qsys 系统的逻辑大小。通常情况下，在逻辑使用和性能之间有一个权衡。本小节中的信息适用于 Avalon 以及 AXI 接口。

最小化互联逻辑 (Minimize Interconnect Logic)

在 Qsys 中，主从接口之间的连接变化会降低系统中所需互联逻辑的数量。

创建专用主从接口连接

您或许能够创建一个系统，使主接口连接到单一从接口。这种配置消除了地址解码，仲裁和返回数据多路复用，从而简化了互联。专用主接口到从接口 (master-to-slave) 的连接实现了与 Avalon-ST 连接相同的时钟频率。

通常情况下，这些一对一连接包含一个 Avalon 存储器映射桥接或者硬件加速器。例如，如果在一个从接口和所有其它主接口之间插入一个流水线桥接，那么桥接主从接口之间的逻辑被简化成线。第 10-18 页图 10-16 显示了这一方法。如果一个硬件加速器仅连接到专用存储器，那么在主从接口对之间不会生成系统互联逻辑。

移除不必要的连接

主从接口间的连接数量会影响您系统的 f_{MAX} 。每个连接到从接口的主接口会增加多路复用器位宽。随着多路复用器位宽的增加，实现 FPGA 中多路复用器的逻辑深度和宽度也会增加。要提高您的系统性能，仅在必要时连接主从接口。

当一个主接口连接到多个从接口时，读数据信号的多路复用器会增加。Avalon 通常使用一个 `readdata` 信号，AXI 读数据信号使用命令 `rdata`，`rresp` 和 `rlastread` 来添加一个响应状态和最后指示器到读响应通道。使用桥接帮助控制多路复用器的深度，如图 10-9 所示。

简化地址解码逻辑

如果地址代码逻辑在关键路径中，那么您能够修改地址映射来简化解码逻辑。使用不同的地址映射进行试验（包括独热编码）来观察结果是否有所改进。

通过将多个接口合并成一个接口来最小化仲裁逻辑

随着您设计中的组件数不断增加，实现互联所需要的逻辑数量也不断增加。对于多个主接口共享的每个从接口，仲裁模块数也不断增加。随着支持读传输的从接口数量在每个主接口基础上不断增加，读数据多路复用器的宽度也不断增加。基于这些原因，考虑作为单一接口实现多个模块的逻辑以减少互联逻辑使用。

逻辑整合权衡

修改您的系统或接口之前需要考虑下面权衡。



关于并发性权衡的更多讨论，请参考第 10-4 页“在存储器映射系统中使用并发机制”。

首先，考虑整合组件时带来的并发性影响。当您的系统有四个主组件和四个从接口时，它能发起四个并发性存取。如果将四个从接口合并成一个接口，那么四个主接口一定会争用存取。因此，您应该只合并那些低优先权的接口，例如低速并行 I/O 器件（如果此组合对性能没有影响）。

其次，对于互联之前包括的从接口，要确定合并是否会引进新的解码和多路复用逻辑。如果一个接口包含多个读写地址位置，那么此接口就已经包含必要的解码和多路复用逻辑。合并接口时，通常重新使用已经包含在原来其中一个接口中的解码器和多路复用器模块；然而，合并接口可能只会移动解码器和多路复用器逻辑，而不是消除重复。

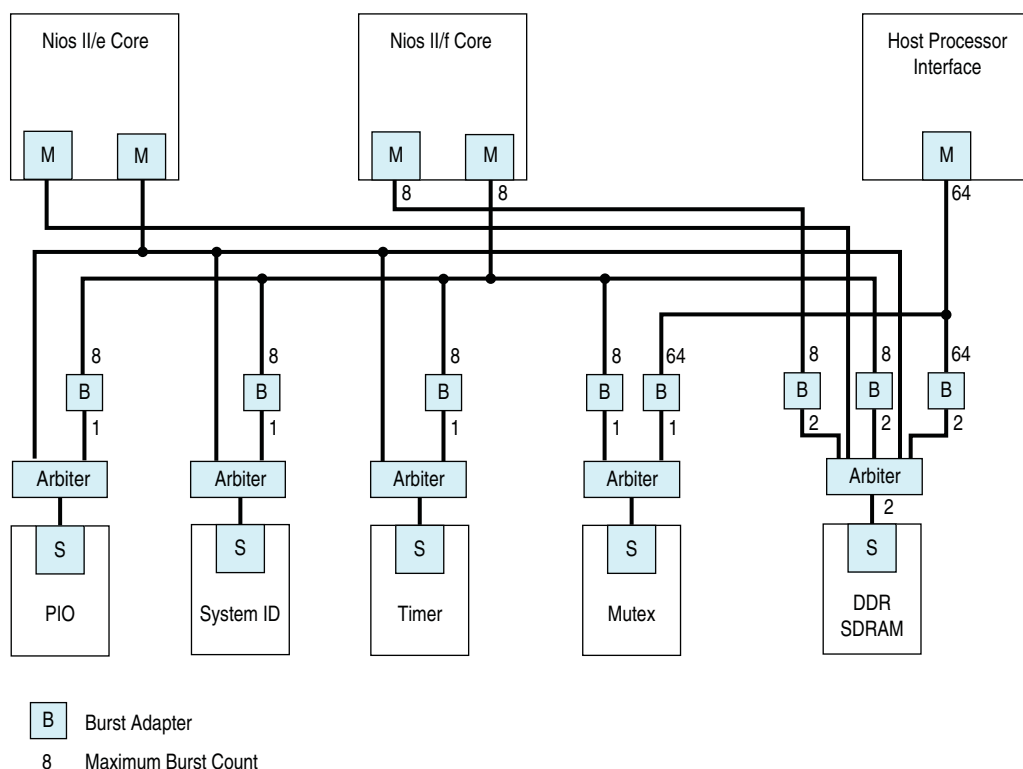
最后，要考虑合并接口是否会使设计变得复杂。如果是，那么 Altera 建议您不要合并接口。

合并接口的系统实例

在此实例中，Nios II/e 内核维持 Nios II /f 内核与外部处理器之间的通信。Nios II/f 内核支持值为 8 的最大突发大小。外部处理器接口支持值为 64 的最大突发长度。Nios II/e 内核不支持突发。系统中的唯一存储器是最大突发长度为 2 的 SDRAM。

图 10-21 显示了一个具有不同突发性能的组件混合系统。该系统包含一个 Nios II/e 内核，一个 Nios II/f 内核和一个外部处理器（将某些处理任务卸载给 Nios II/f 内核）。

图 10-21. 混合的突发系统

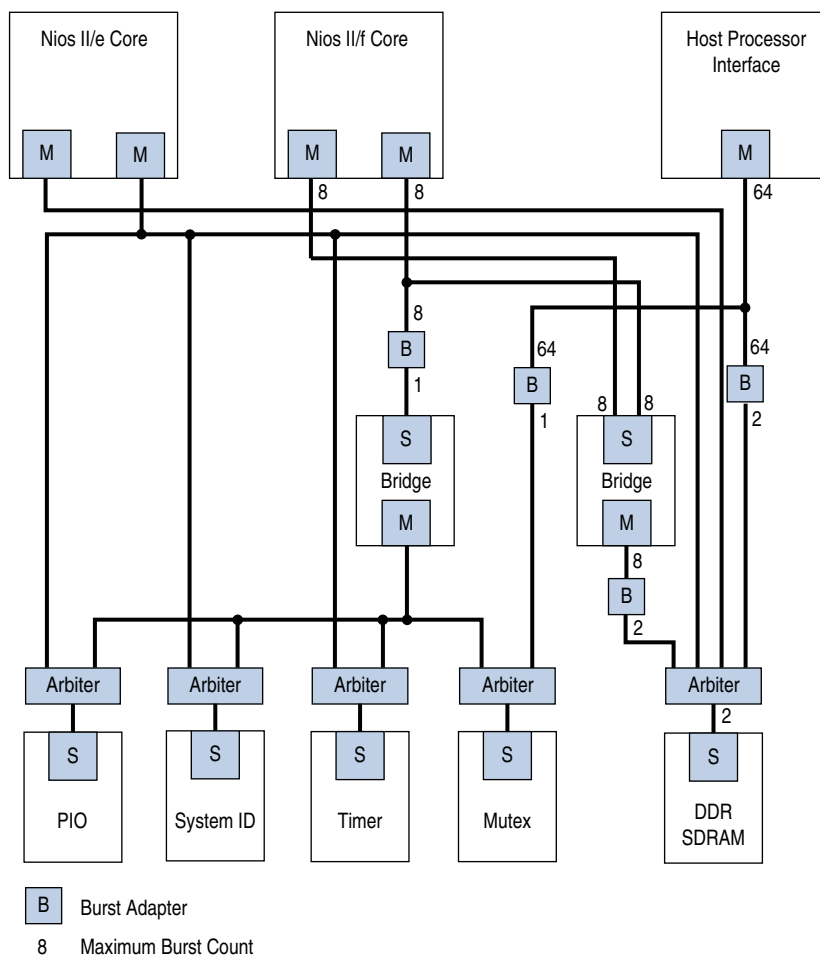


Qsys 自动插入突发适配器来对突发长度失配提供补偿。适配器将突发降低成一个传输或者两个传输的长度。对于连接到 DDR SDRAM 的外部处理器接口，64 个字的突发被分成 32 个突发传输，每个突发传输的长度为 2 个字。

当您生成一个系统时，Qsys 会根据 burstcount 最大值插入突发适配器；因此，如果主接口能够突发，那么互联逻辑就包括不要求突发的主从接口对之间的突发适配器。在图 10-21 中，Qsys 在 Nios II 处理器与计时器，系统 ID 和 PIO 外设之间插入突发适配器。这些组件不支持突发，Nios II 处理器仅对这些组件执行单一字读写访问。

要减少适配器的数量，如图 10-22 所示添加流水线桥接 (pipeline bridge)。Nios II/f 内核与不支持突发的外设之间的流水线桥接从图 10-21 中消除了三个突发适配器。Nios II/f 内核与 DDR SDRAM 之间的第二个流水线桥接（最大突发长度设为 8）消除了另一个突发适配器。

图 10-22. 带桥接的混合突发系统



实现多个时钟域

在 Qsys 中的 **System Contents** 标签中指定时钟域。时钟源能够被 Qsys 的外部输入信号驱动，或者被 Qsys 中的 PLL 驱动。时钟域根据时钟名进行区分。您可以创建具有相同频率的多个异步时钟。

时钟域交叉逻辑 (Clock Domain Crossing Logic)

Qsys 生成 Clock Domain Crossing Logic (CDC)，CDC 隐藏了运行在不同时钟域的接口组件的细节。系统互联独立支持每个端口的存储器映射协议，因此主接口不需要为连接不同域中的从接口而集成时钟适配器。Qsys 互联逻辑在时钟域边界之间自动传播传输。

时钟域适配器具有如下优点：

- 允许组件接口运行在不同的时钟频率上。
- 无需设计 CDC 硬件。
- 允许每个存储器映射端口只运行在一个时钟域中，从而降低组件设计的复杂性。
- 使主接口能够访问任意的从接口，而不需要与从时钟域进行通信。
- 使您能够对要求快速时钟的组件进行集中性能优化。

一个时钟域适配器包含两个有限状态机 (FSM) (每个时钟域一个), FSM 使用简单的“握手”协议在时钟边界之间传播传输控制信号 (read_request, write_request 和 master waitrequest 信号)。

图 10-23 显示了一个主接口与一个从接口之间的时钟域适配器。

图 10-23. 时钟交叉适配器结构图

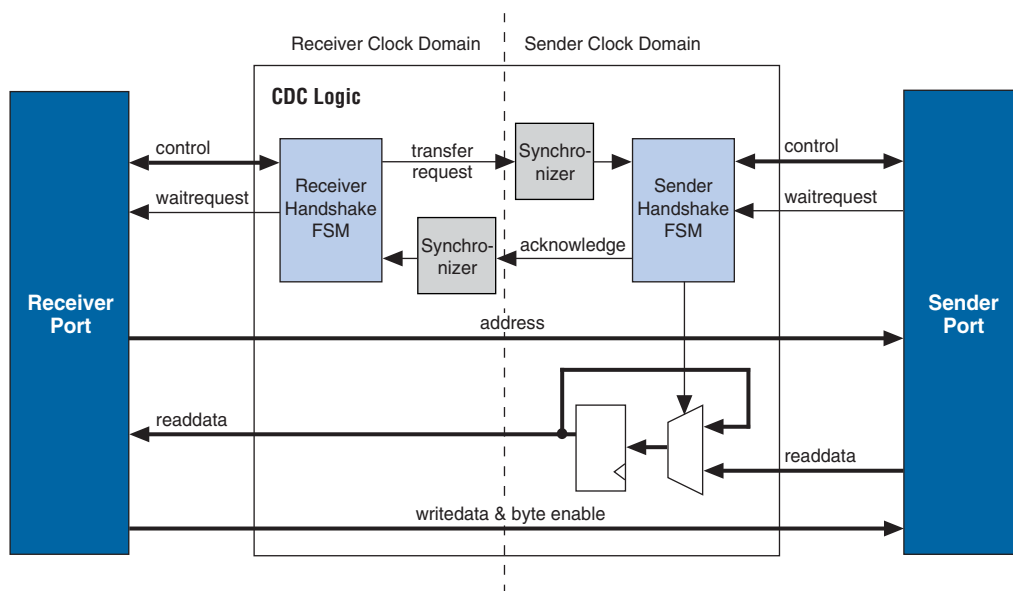



图 10-23 中同步器模块 (synchronizer block) 使用多级触发器 (flipflop) 消除进入握手 FSM 的控制信号上的亚稳事件。CDC 逻辑运行在任意的时钟比率。

CDC 逻辑之间传输的典型事件顺序如下：

1. 主接口置位地址，数据和控制信号。
2. 主接口握手 FSM 采集控制信号，并立即强制主接口等待。

 FSM 仅使用控制信号，不使用地址和数据信号。例如，主接口保持地址信号不变直到从接口安全采集到此信号。

3. 主接口握手 FSM 初始化到从握手 FSM 的传输请求。
4. 传输请求被同步到从时钟域。
5. 从接口握手 FSM 处理请求，通过从接口执行请求的传输。
6. 从接口传输完成时，从接口握手 FSM 发回一个收悉 (acknowledge) 到主接口握手 FSM。

7. 收悉 (acknowledge) 被同步回主时钟域。
8. 主握手 FSB 通过将主接口从等待状态中释放来完成传输。

传输在主从接口两侧正常进行，无需特殊协议来处理交叉时钟域。从从接口的角度看，在不同的时钟域中由主接口初始化的传输没有差别。从主接口的角度看，时钟域之间的传输仅需要额外的时钟周期。与其它传输延迟情况类似（例如，从接口一侧上的仲裁延迟或等待状态），Qsys 强制主接口等待直到传输终止。因此，当在不同的时钟域进行传输时，流水线主端口不受益于流水线 (pipelining)。

Qsys 根据系统内容和组件之间的连接自动决定插入 CDC 逻辑的位置，并布局 CDC 逻辑以保持所有组件的最高传输率。Qsys 对每个主从接口对 (master & slave pair) 独立地评估对 CDC 逻辑的需要，并在必要时生成 CDC 逻辑。

交叉时钟域传输持续时间

CDC 逻辑扩展在时钟域边界上主接口传输的持续时间。在最坏情况下（读），每个传输被扩展 5 个主时钟周期和 5 个从时钟周期。假设默认的 Master domain synchronizer 长度和 Slave domain synchronizer 长度是 2，此延迟的组件如下：

- 四个额外主时钟周期，由于主侧时钟同步器 (master-side clock synchronizer)
- 四个额外从时钟周期，由于从侧时钟同步器 (slave-side clock synchronizer)
- 每个方向一个额外时钟，由于控制信号跨越时钟域导致的潜在亚稳事件



要求更高性能时钟的系统应该使用 Avalon-MM 时钟交叉桥接，而不是自动插入的 CDC 逻辑。时钟交叉桥接包括一个缓存机制，以便能够流水线化多个读写操作。对第一次读或写过程中存在的延迟损耗，对于待定的读写没有其它的延迟损耗，从而增加高达 4 倍的数据吞吐量，以添加逻辑资源为代价。



关于更多信息，请参考 *Embedded Design Handbook* 中的 *Avalon Memory-Mapped Design Optimizations*。

降低功耗

这一部分介绍了各种低功耗设计变更，通过应用这些变更能够降低互联和您的定制组件的功耗。



QII 当前版本的 Qsys 不支持 AXI 标准低功耗扩展。

使用多个时钟域

当使用多个时钟域时，您应该将非关键逻辑放置在较慢的时钟域中。Qsys 通过插入时钟交叉逻辑 (handshake 或 FIFO) 自动调解异步时钟域的数据交叉。

您可以使用 Qsys 中的时钟交叉来降低不要求高频时钟的逻辑的时钟频率，从而降低功耗。您可以使用握手时钟交叉桥接或者握手时钟交叉适配器来分离时钟域。

时钟交叉桥接

您可以使用时钟交叉桥接将运行在较高频率上的主接口连接到运行在较低频率上的从接口。仅将低数据吞吐量或低优先权的组件连接到运行在降低的时钟频率的时钟交叉桥接。下面是低数据吞吐量或低优先权的组件的实例：

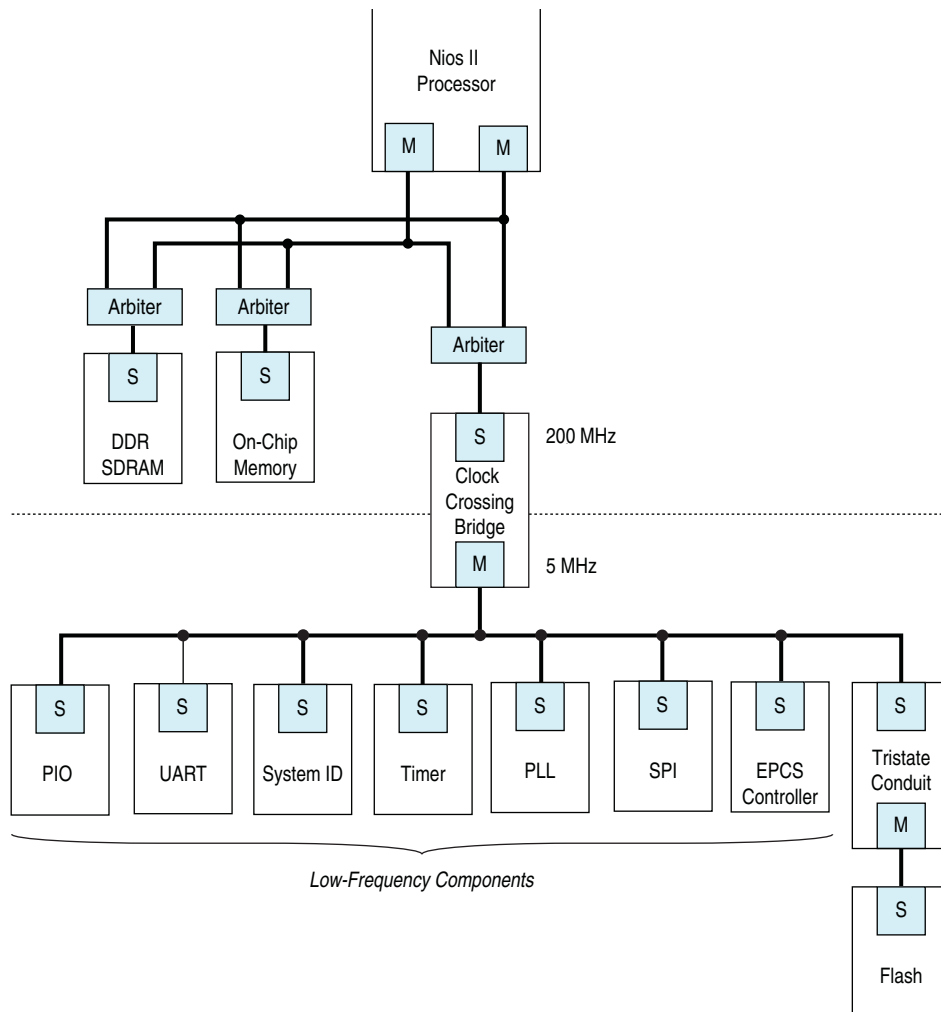
- PIOs

- UARTs (JTAG or RS-232)
- 系统识别 (SysID)
- 计时器
- PLL (在 Qsys 中例化)
- 串行外设接口 (SPI)
- EPCS 控制器
- 三态桥接和连接到桥接的组件

通过降低连接到桥接的组件的时钟频率，可以降低您设计的动态功耗。动态功耗影响翻转率，降低时钟频率会降低翻转率。

图 10-24 显示一个通过桥接降低功耗的系统。

图 10-24. 通过使用桥接分离时钟域来降低电源使用



Qsys 在运行在不同时钟频率的主从接口之间自动插入时钟交叉适配器。在 Qsys **Project Settings** 标签上选择时钟交叉适配器的类型。在 Qsys 中有三种可用的时钟交叉适配器类型，描述如下。由于没有插入适配器，因此它们没有出现在 Qsys **Connection** 列中。

时钟交叉适配器类型

对自动插入的时钟交叉适配器指定默认实现。可使用的适配器类型如下：

- **Handshake**— 使用简单的握手协议在时钟边界之间传播传输控制信号和响应。由于每个传输在下一个传输开始之前被安全地传播到目标域，因此该适配器使用更少的硬件资源。Handshake 适配器适用于要求低数据吞吐量的系统。
- **FIFO**— 将双时钟 FIFO 用于同步。FIFO 适配器的延迟比握手时钟交叉组件的延迟多了近似 2 个时钟周期，但基于 FIFO 的适配器能够维持更高的数据吞吐量，因为它能同时支持多个传输。FIFO 适配器需要更多的资源。FIFO 适配器适用于要求高数据吞吐量的，在时钟域之间的存储器映射传输。
- **Auto**— Qsys 对突发链路指定相应的 FIFO 适配器，对其它链路指定 Handshake 适配器。

吞吐量

由于时钟交叉桥接使用 FIFO 实现时钟交叉逻辑，因此它缓冲传输和数据。时钟交叉适配器未被流水线化，在当前传输完成之前，每个传输都被阻止。阻止传输本质上可能会降低数据吞吐量；因此，如果想要降低功耗，但又对吞吐量没有太大的限制，那么应该使用时钟交叉桥接或者 FIFO 时钟交叉适配器。但是，如果设计要求单一读传输，那么使用时钟交叉适配器会更好一些，因为它的延迟更低。

资源利用

除了片上存储器，时钟交叉桥接需要很少的逻辑资源。所使用的片上存储器模块数与桥接的地址跨度，数据位宽，缓冲深度以及突发性能成正比。时钟交叉适配器不使用片上存储器，但需要适量的逻辑资源。时钟交叉适配器的地址跨度，数据位宽，以及突发性能决定了器件的资源使用。

吞吐量 vs 存储器权衡

当您决定使用时钟交叉桥接或者时钟交叉适配器时，您必须要考虑到您设计中数据吞吐量和存储器使用所带来的影响。如果片上存储器资源是有限的，那么您可能被强制选择时钟交叉适配器。使用时钟交叉桥接来降低一个组件的功耗可能不会证明使用更多的资源是合理的。然而，如果将所有低优先权的组件布置在单一时钟交叉桥接的后面，那么可以降低您设计中的功耗。

最小化翻转率

每当逻辑在 on 与 off 状态之间跳变时，您的设计就会消耗功耗。当时钟边沿之间的状态保持不变时，不会出现充放电。这一小节讨论了三种用于降低您设计的翻转率的技术方法：

- 寄存组件边界
- 使用时钟使能信号
- 插入桥接

寄存组件边界 (registering Component Boundaries)

当没有适配器，桥接和互联流水线时，Qsys 互联是独特组合的。当一个从接口没有被主接口选择时，多个信号可能翻转并传播到组件中。通过在主或从接口上寄存组件的边界，可以最小化互联和组件的翻转。此外，寄存边界能够提高操作频率。当您在接口级上寄存信号时，一定要确保组件继续在接口标准规格范围内运行。

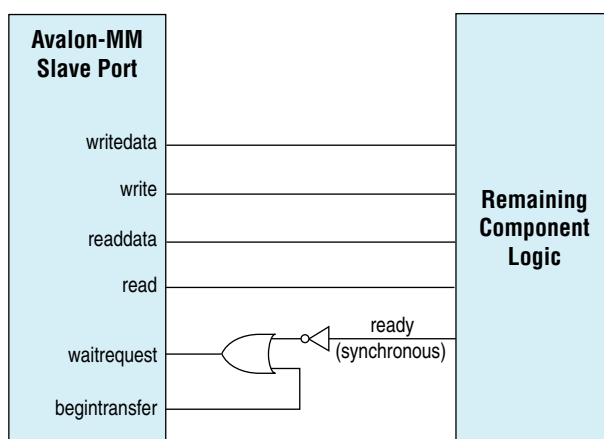
当添加寄存器到您的组件中时，Avalon-MM waitrequest 是一个很难同步的信号。为延长传输，在主接口置位 read 或 write 的同一时钟周期中必须被置位 waitrequest。一个主接口可能过早地读取 waitrequest 信号，并过早地发出更多的读写操作。



对于 waitrequest 和 burstcount，不存在直接的 AXI 等效，尽管 *AMBA Protocol Specification* 指示 ready (Avalon-MM waitrequest 的等效) 不能综合取决于 AXI valid。因此，Qsys 通常缓冲 AXI 组件边界 (至少用于 ready 信号)。

对于从接口，互联管理 begintransfer 信号，此信号在 read 或 write 传输的第一个时钟周期中被置位。如果您的 waitrequest 迟了一个时钟周期，那么可以逻辑 OR 您的 waitrequest 和 the begintransfer 信号来形成一个新的正确同步的 waitrequest 信号，如图 10-25 所示。

图 10-25. 可变延迟



或者，您的组件可以在选择 waitrequest 之前将其置位，从而保证 waitrequest 在传输的第一个时钟周期中已经被置位。

使用时钟使能 (Using Clock Enables)

通过使用时钟使能可以保持您的逻辑处于一个稳定状态。您可以使用读写信号作为从组件的时钟使能。即便添加寄存器到您的组件边界，您的接口在不使用时钟使能的情况下也可能翻转。

您也可以使用时钟使能来禁用您的组件的组合部分。例如，您可以使用有效高电平时钟使能 (active high clock enable) 屏蔽组合逻辑的输入，防止该输入在组件无效 (inactive) 时翻转。在防止无效逻辑翻转之前，你必须确定此屏蔽是否导致您的电路非常规运转。如果屏蔽导致了功能性失败，那么可能要使用寄存器级 (register stage) 在时钟周期之间保持组合逻辑不变。

插入桥接

如果不想通过使用边界寄存器或者时钟使能来修改组件，那么您可以使用桥接来降低翻转率 (toggle rate)。一个桥接作为中继器运行，将主接口上的传输复制在从接口上。如果桥接没被访问，那么连接到它的主接口的组件也没被访问。桥接的主接口在一个主接口访问桥接的从接口之前保持空闲。

桥接也能够降低信号（其它主接口的输入）的翻转率。这些信号通常是 `readdata`，`readdatavalid` 和 `waitrequest`。支持读访问的从接口驱动 `readdata`，`readdatavalid` 和 `waitrequest` 信号。一个桥接在主从接口之间插入一个寄存器或者时钟交叉 FIFO，以降低主接口输入信号的翻转率。

禁用逻辑

通常有两种低功耗模式：易失性 (volatile) 和非易失性 (non-volatile)。volatile 低功耗模式保持组件处于复位状态。当重新启用逻辑时，之前的操作状态丢失。non-volatile 低功耗模式恢复之前的操作状态。本小节介绍使用软件控制或硬件控制的睡眠模式来禁用组件，从而降低功耗。

软件控制的睡眠模式

要设计一个支持软件控制睡眠模式的组件，需要创建一个单一存储器映射位置，通过写入 0 或 1 来使能或禁用逻辑。根据组件是否有非易失性要求，使用寄存器的输出作为时钟使能或复位。从接口在睡眠期间必须保持有效 (active)，以便需要启用组件时能设置 `enable` 位。

如果多个主接口能够访问一个支持睡眠模式的组件，那么您可以使用 Qsys 中的互斥内核 (mutex core) 提供对组件的互斥访问。您也可以内置逻辑，当系统中的任意主接口每次首访时重新使能组件。如果组件需要多个时钟周期进行重新启用 (re-activate)，那么在它退出睡眠模式时必须置位等待请求以延长传输。



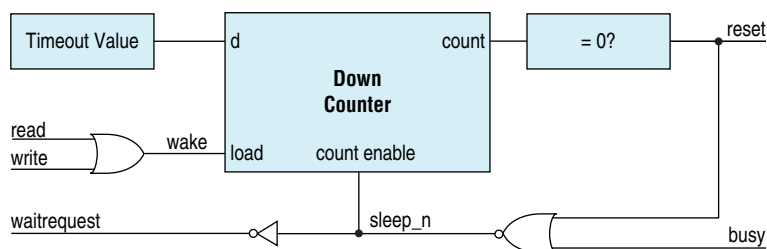
关于互斥内核的更多信息，请参考 *Embedded Peripherals IP User Guide* 的 [Mutex Core](#) 章节。

硬件控制的睡眠模式

您可以在您的组件中实现一个计时器 (timer)，该计时器根据读或写访问之间的时钟周期中指定的超时值 (timeout value) 自动使组件进入睡眠模式。每次访问都将计时器复位到超时值。没有访问的每个周期以 1 递减超时值。如果计数器达到零，那么硬件进入睡眠模式，直到下一次访问。图 10-26 显示了此逻辑的原理图。如果将组件恢复到活动状态需要很长时间，那么就要使用一个较长的超时值，这样组件就不会连续进入和退出睡眠模式。

当组件的其余部分处于睡眠模式时，从接口必须保持可用。当组件退出睡眠模式时，组件必须置位 `waitrequest` 信号，直到可以读写。

图 10-26. 硬件控制的睡眠组件



关于降低电源使用的详细信息，请参考 *Quartus II Handbook* 中的 *Power Optimization*。

设计实例

下面实例显示了 Qsys 系统设计挑战的应对方法。

Avalon 流水线读主接口实例

对于使用 Avalon-MM 标准的高吞吐量系统，您可以设计一个流水线读主接口 (pipelined read master)，使您的系统能够在数据返回前发出多个读请求。流水线读主接口通过频繁（每个时钟周期）发出读操作来隐藏读操作的延迟。当地址逻辑依赖数据返回时，您可以使用这个主接口。

设计要求

对于流水线读主接口的控制和数据通路，您必须仔细认真地设计逻辑。每当 `waitrequest` 信号被置位时，控制逻辑必须扩展一个读周期。此逻辑也必须控制主接口 `address`、`byteenable` 和 `read` 信号。要实现最大吞吐量，只要 `waitrequest` 被置低，流水线读主接口就应该连续发出读操作 (post reads)。当 `read` 被置位时，互联中的地址被保存。

数据通路逻辑包括 `readdata` 和 `readdatavalid` 信号。如果您的主接口能够在每个时钟周期上接收数据，那么您可以使用 `readdatavalid` 作为使能位来寄存数据。如果您的主接口不能处理连续的读数据，那么它必须在 FIFO 中缓存数据。当 FIFO 达到预定的填充级时，控制逻辑就必须停止发送读操作，从而防止 FIFO 上溢。

关于实现 Avalon 流水线读主接口的信号的详细信息，请参考 *Avalon Interface Specifications*。

提高到所需数据吞吐量

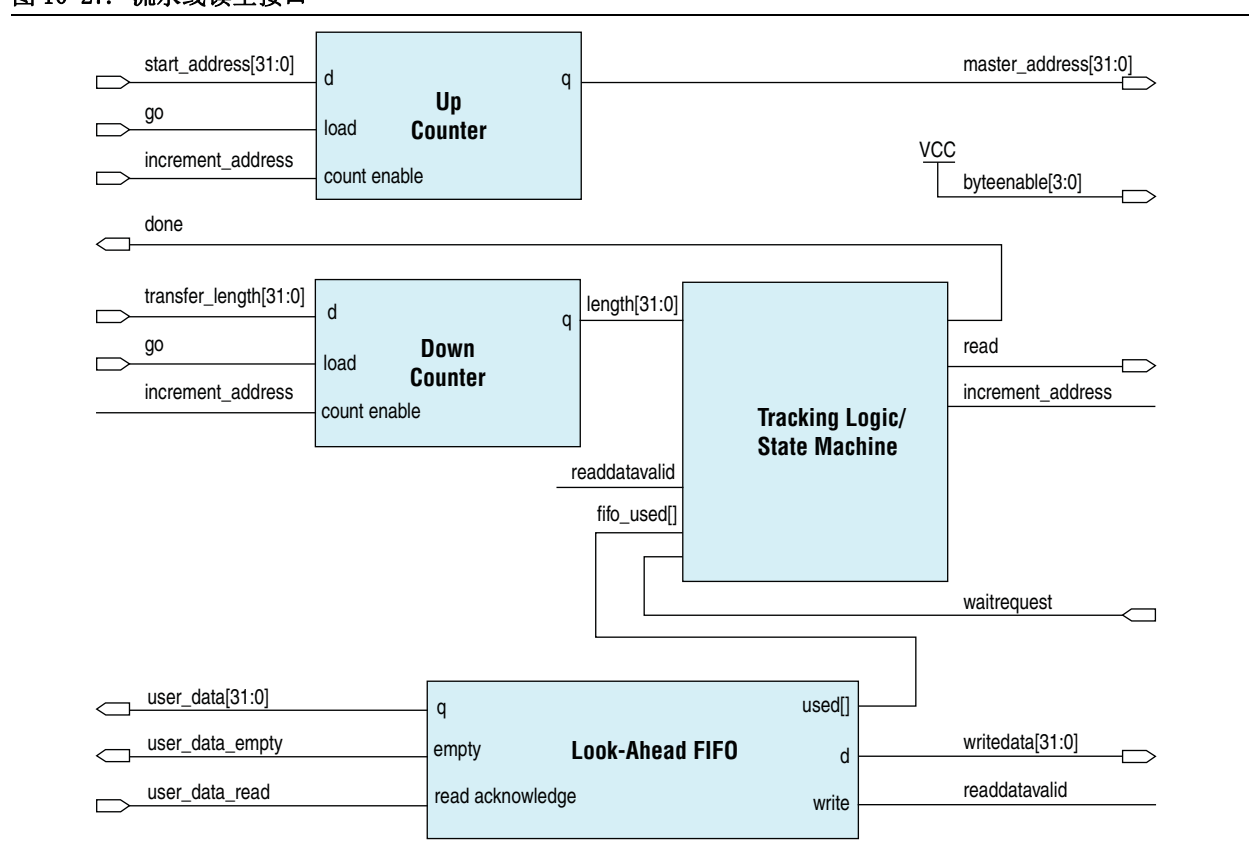
可以通过流水线读主接口实现的数据吞吐量通常与互联和从接口的流水线深度成正比。例如，如果总延迟是两个周期，那么通过插入流水线读主接口可以使吞吐量翻一番，假设从接口也支持流水线传输。如果主接口或从接口不支持流水线读传输，那么互联置位 `waitrequest`，直达传输完成。读响应之前，如果存在一些周期的开销 (overhead)，那么也可以增加吞吐量。

第 10-14 页“增加延迟”介绍了一个主从接口都支持流水线读传输的实例。在此实例中，数据在初始延迟之后能够在连续的数据流上传输。读操作没有被流水线化，吞吐量就会降低。当主从接口都支持流水线读传输时，数据在初始延迟之后在连续的数据流中传输。图 10-27 显示了没被流水线化的读操作。此系统对每个读操作使用三个周期的延迟，从而达到 25% 的总数据吞吐量。图 10-20 显示了被流水线化的读操作。三个周期的延迟后，数据连续传输。

通过使用流水线读主接口（将数据存储在 FIFO 中）可以实现一个定制 DMA，硬件加速器，或者片外通信接口。图 10-27 显示了一个将数据存储在 FIFO 中的流水线读主接口。主接口执行字对齐的字存取，并从有序存储器地址读取数据。传输长度是字大小的倍数。在图 10-27 中，主接口执行字对齐的字存取，并从有序存储器地址读取数据。传输长度是字大小的倍数。

图 10-27 显示了一个将数据存储在 FIFO 中的流水线读主接口。

图 10-27. 流水线读主接口



当 go 比特被置位时，主接口寄存 start_address 和 transfer_length 信号。主接口在下一个时钟开始连续发出读操作，直到 length 寄存器达到 0。在此实例中，字大小为 4 个字节，这样地址就会始终以 4 递增，长度以 4 递减。read 信号保持置位，除非 FIFO 填充到预定水平。如果 length 还没达到 0，并且 read 被发出，那么 address 寄存器递增，length 寄存器递减。

每当 read 信号被置位，waitrequest 信号被置低时，主接口就会发出一个读传输。主接口发出读操作，直到整个缓存被读取或者 waitrequest 被置位。一个可选的跟踪模块监控 done 比特。当 length 寄存器达到 0 时，一些读操作未被处理。跟踪逻辑防止最后的 read 完成前 done 被置位。跟踪逻辑监控发到互联的读数量，从而不会超过 readdata FIFO 中的剩余空间。此逻辑包括一个计数器，验证以下条件是否被满足：

- 如果发出一个读操作并且 readdatavalid 被置低，那么计数器递增。
- 如果没有发出一个读操作并且 readdatavalid 被置位，那么计数器递减。

当 length 寄存器和跟踪逻辑计数器达到 0 时，所有读操作都已完成，done 比特被置位。如果第二个主接口覆盖流水线读接口访问的存储器位置，那么 done 比特是很重要的。此比特保证了覆盖原始数据前读操作已经完成。

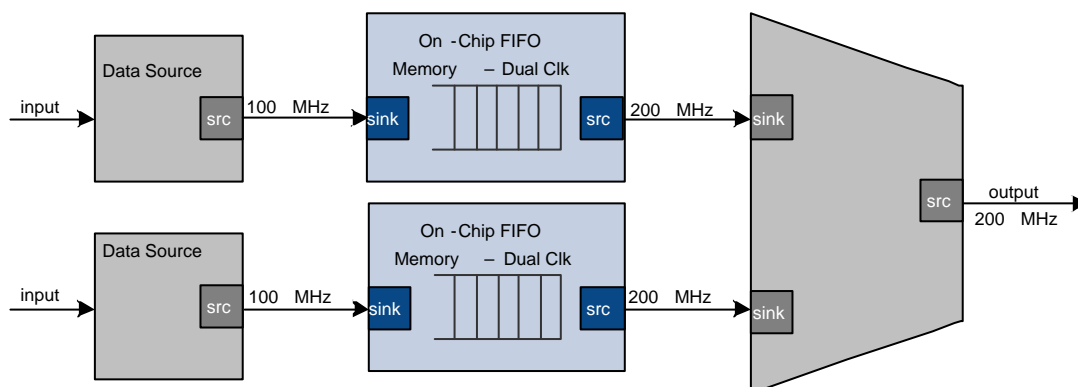
多路复用器实例

您可以将适配器与流组件 (streaming component) 组合起来创建数据通路，该数据通路的输入和输出流有不同的属性。以下给出了数据通路的实例，在这些数据通路中输出流的性能要高于输入流。图 10-28 显示双倍时钟频率的每个接口的双倍吞吐量的输出。图 10-29 显示双倍数据位宽。图 10-30 显示了通过复用两个数据源的输入数据提升 10% 的流频率。

双倍时钟频率的实例

图 10-28 显示了一个数据通路，此数据通路使用双时钟版本的片上 FIFO 存储器和 Avalon-ST 通道多路复用器将两个流数据源的 100 MHz 输入合并成一个 200 MHz 流输出。此实例通过提高频率和组合输入来提高数据吞吐量。

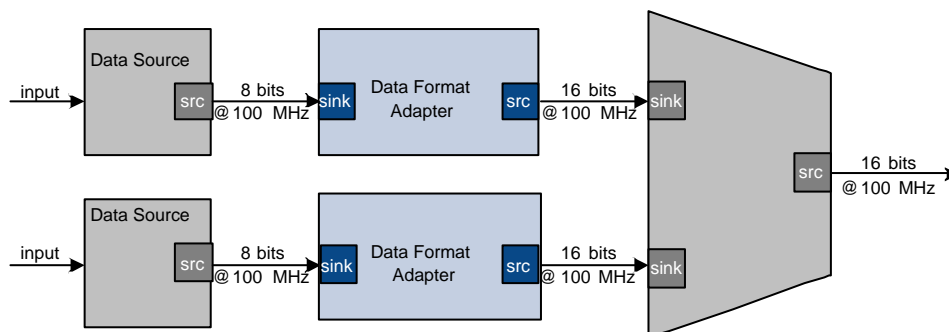
图 10-28. 双倍时钟频率的数据通路



双倍数据位宽和保持频率的实例

图 10-29 显示了一个数据通路，该数据通路使用数据格式适配器和 Avalon-ST 通道多路复用器将两个运行在 100 MHz 的 8-bit 输入转换成一个 100 MHz 的 16-bit 输出。

图 10-29. 双倍数据位宽和保持原始频率的数据通路

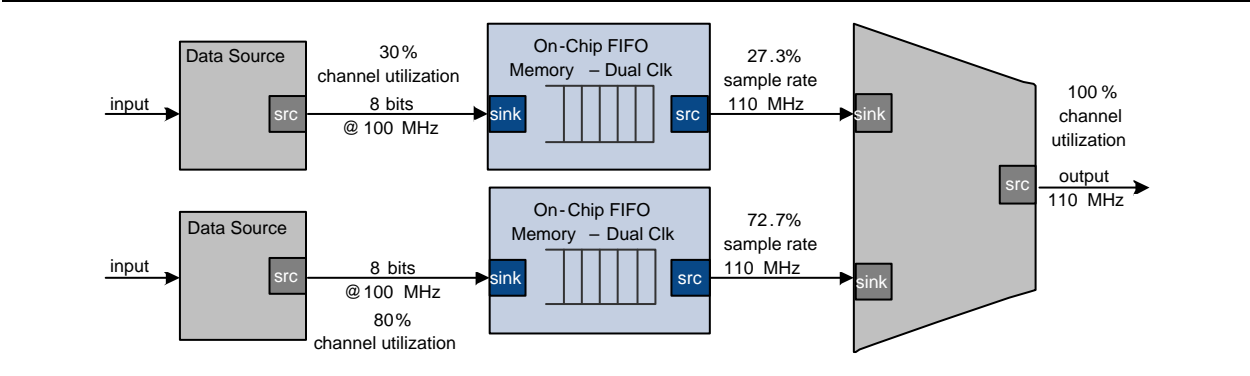


提升频率的实例

图 10-30 显示了一个数据通路，该数据通路使用双时钟版本的片上 FIFO 存储器通过对采样两个不同速率的输入数据流将输入数据的频率从 100 MHz 提升到 110 MHz。在此实例中，片上 FIFO 存储器有一个 100 MHz 的输入时钟频率和一个 110 MHz 的输出时钟频率。通道多路复用器运行在 110 MHz，采样一个输入数据流（27.3% 的时间）和第二个输入数据流（72.7% 的时间）。

开始这种设计之前，您不需要了解典型的和最大的输入通道使用率是多少。例如，如果第一个通道达到 50% 使用率，那么输出数据流将超过 100% 使用率。

图 10-30. 提升时钟频率的数据通路



结论


本章中给出建议可提高您系统的最大时钟频率，并发性和数据吞吐量，逻辑使用率，甚至电源利用率。当设计一个 Qsys 系统时，除了 Qsys 提供的自动优化，您还要根据您的设计意图和目标来进一步优化系统性能。

文档修订历史

表 10-2 显示了本文档的修订历史。

表 10-2. 文档修订历史

日期	版本	修订内容
2013 年 5 月	13.0.0	■ 添加了 AMBA APB 支持。
2012 年 11 月	12.1.0	■ 添加了 AMBA AXI4 支持。
2012 年 6 月	12.0.0	■ 添加了 AMBA AXI3 支持。
2011 年 11 月	11.1.0	■ 新文档发布。

 关于 *Quartus II Handbook* 的更早版本，请参考 [Quartus II Handbook Archive](#)。