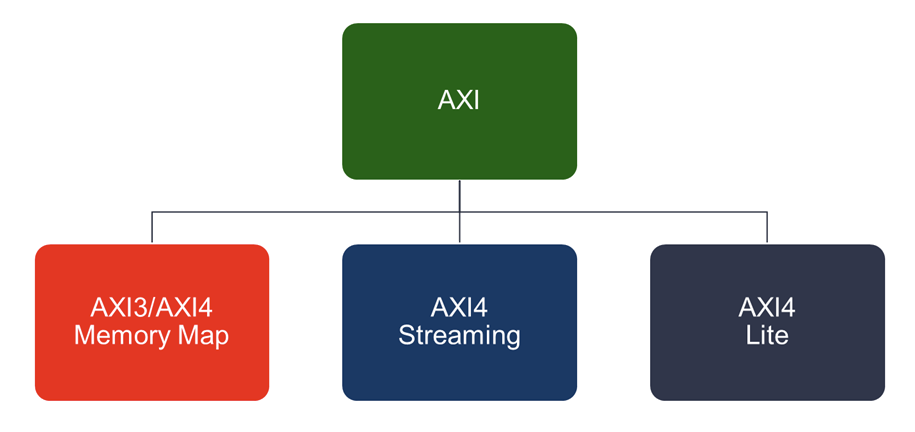
AXI参考连接：

[https://blog.csdn.net/qq\_41019681/article/details/118740047#t5](https://blog.csdn.net/qq_41019681/article/details/118740047" \l "t5)

<https://blog.csdn.net/moshanghongfeng/article/details/113857269?utm_medium=distribute.pc_relevant.none-task-blog-2~default~baidujs_baidulandingword~default-1-113857269-blog-118740047.235>^v43^pc\_blog\_bottom\_relevance\_base1&spm=1001.2101.3001.4242.1&utm\_relevant\_index=4

<https://support.xilinx.com/s/article/1053914?language=en_US> 重要

AXI, which means **A**dvanced e**X**tensible **I**nterface, is an interface protocol defined by ARM as par of the [AMBA](https://developer.arm.com/architectures/system-architectures/amba) (Advanced Microcontroller Bus Architecture) standard.



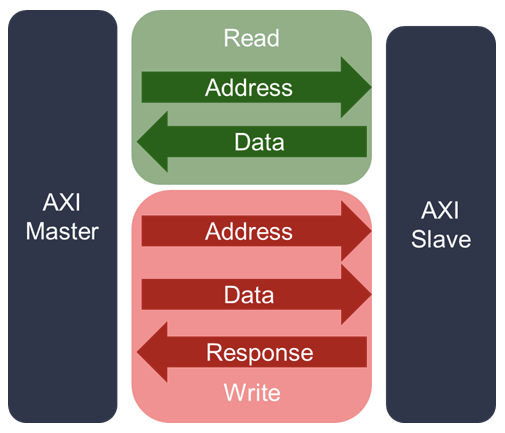
There are 3 types of AXI4-Interfaces (AMBA 4.0):

* AXI4 (Full AXI4): For high-performance **memory**-mapped requirements.
* AXI4-Lite: For simple, low-throughput memory-mapped communication (for example, to and from control and status **registers**).
* AXI4-Stream: For high-speed **streaming** data.

## AXI Read and Write Channels

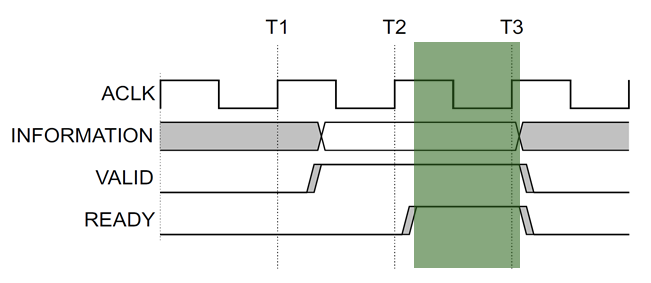
The AXI protocol defines 5 channels:

* 2 are used for Read transactions
  + read address
  + read data
* 3 are used for Write transactions
  + write address
  + write data
  + write response



A **channel** is an independent collection of AXI signals associated with the VALID and READY signals.

A piece of data transmitted on a single channel is called a **transfer**. A transfer happens when both the VALID and READY signal are high while there is a rising edge of the clock. For example, in the figure below, the transfer is happening on T3:

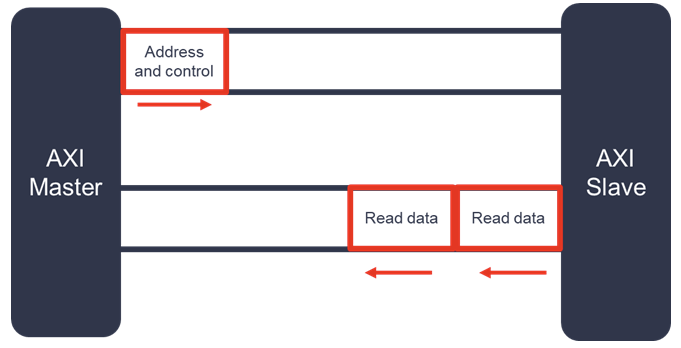


## AXI Read Transactions

An AXI Read transactions requires multiple transfers on the 2 Read channels.

* First, the **Address Read Channel** is sent from the Master to the Slave to set the address and some control signals.
* Then the data for this address is transmitted from the Slave to the Master on the **Read data channel**.

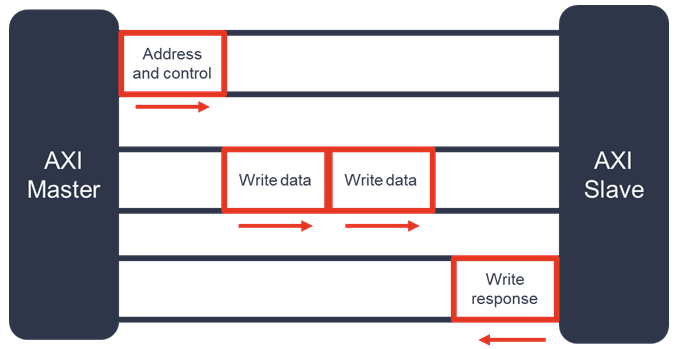
Note that, as for the figure below, there can be multiple data transfers per address. This type of transaction is called a **burst**.



## AXI Write Transactions

An AXI Write transactions requires multiple transfers on the 3 Read channels.

* First, the **Address Write Channel** is sent Master to the Slave to set the address and some control signals.
* Then the data for this address is transmitted Master to the Slave on the **Write data channel**.
* Finally the write response is sent from the Slave to the Master on the **Write Response Channel** to indicate if the transfer was successful.



The possible response values on the **Write Response Channel** are:

* **OKAY (0b00)**: Normal access success. Indicates that a normal access has been successful
* **EXOKAY (0b01)**: Exclusive access okay.
* **SLVERR (0b10)**: Slave error. The slave was reached successfully but the slave wishes to return an error condition to the originating master (for example, data read not valid).
* **DECERR (0b11)**: Decode error. Generated, typically by an interconnect component, to indicate that there is no slave at the transaction address

**Note**: Read transactions also have a response value but this response is transmitted as part of the **Read Response Channe**

1、AXI总线的Burst传输：

给出启始地址，一次性传送多字节就是Burst。

在AXI总线上一次Burst操作由若干transfers 亚操作完成。比如一次性Burst 64字节，数据通道位宽是8bits（1Byte），则需要64个transfers；数据通道位宽是32bits（4Bytes），则需要16个transfers。一次transfer的位宽叫做Burst Size（Byte）（应该叫做transfer size更为恰当），由AxSIZE[2:0]定义，最大2^AxSIZE[2:0] = 2^7 = 128。

一次Burst的transfers亚操作数量称为Burst Length。一次Burst传送的字节数 = Burst Size \* Burst Length。Burst Length取1、2、4、8、16、32、64、128、256中的一个（但有具体要求）。对于所有突发类型，AXI3支持1-16个传输的突发长度。AXI4将对INCR突发类型的突发长度支持扩展到1-256次传输。AXI4中对所有其他突发类型的支持保持在1-16次传输。对

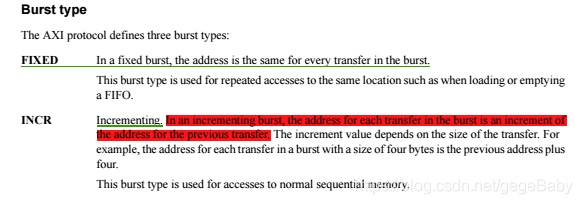
于包装突发，突发长度必须是2、4、8或16。为了适应AXI4中INCR突发类型的扩展突发长度，AXI4的突发长度定义为：*Burst\_Length = AxLEN[7:0] + 1 = 255 + 1 = 256*。

一次Burst传送的总字节数应该叫 Burst Size，显然 Burst Size = transfer size \* Burst Length更为恰当，但官方资料不是这样。一次Burst传送的总字节数官方没有专用名词，其最大值为4K Bytes。

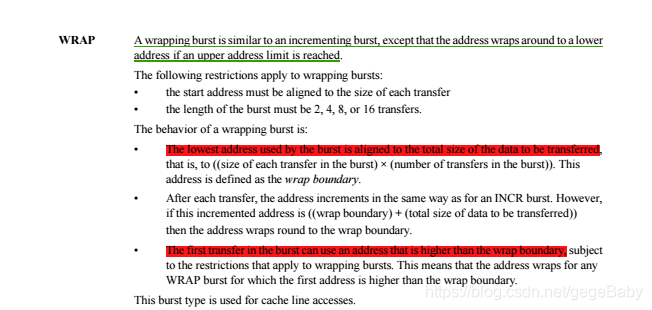
Burst的模式：

1、固定（Fixed）：向固定地址处连续发送数据。

2、增量（Increment）：向启始地址处，按照地址递增的方式，连续发送数据。



3、回环（换行）: 一个换行突发类似于一个递增突发，不同的是，如果达到地址上限，地址会绕到一个较低的地址。这种突发类型用于高速缓存行访问。



Wrap要求：1、启始地址必须按Transfer Size对齐（按数据通道位宽对齐）；

2、Burst Length只能为 2、4、8、16

使用场景：需要传送固定地址区域的数据，1）其数据总量等于一次Burst的数据量，那么Burst中每一次transfer的地址增加方式是循环的。2）其数据总量大于一次Burst的数据量，因而需要多次Burst。存在第2）种情况吗？见下文。

以下提供了确定突发内传输的地址和字节通道的方法。这些方程使用以下变量：

Start\_Address——由管理器发布的开始地址。

Number\_Bytes——每次数据传输的最大字节数。就是Burst Size。

Data\_Bus\_Bytes——数据总线中的字节通道数。可与Number\_Bytes不同吗？是可以的

Aligned\_Address——开始地址的对齐版本。

Burst\_Length——突发内数据传输的总数。一个Burst的transfer数量

Address\_N——突发中传输N的地址。对于突发中的第一次传输，N是1。

Wrap\_Boundary——换行突发中的最低地址。

Lower\_Byte\_Lane——传输的最低地址字节的字节通道。最低字节对应的最低有效通道。

Upper\_Byte\_Lane——传输的最高地址字节的字节通道。

INT(x)——x的向下舍入整数值。

Start\_Address = AxADDR，没有问题，按定义。

Number\_Bytes = 2 ^ AxSIZE，等价于Burst Size的定义。

Burst\_Length = AxLEN + 1，没有问题，按定义。

Aligned\_Address = (INT(Start\_Address / Number\_Bytes)) × Number\_Bytes，假如Start\_Address = 10， Number\_Bytes = 4，则Aligned\_Address = 8。

对于Incr模式和未换行的Wrap模式：

burst中第一次transfer的地址可能是没有对齐的，所以Address\_1 = Start\_Address。

burst中第二次transfer的地址要对齐，所以

Address\_N = Aligned\_Address + (N – 1) × Number\_Bytes，N >= 2，N <= Burst\_Length

对于换行Wrap模式：

Burst的换行地址Wrap\_Boundary =

(INT(Start\_Address / (Number\_Bytes × Burst\_Length)))×(Number\_Bytes × Burst\_Length)

假设Start\_Address = 163， Number\_Bytes = 2， Burst\_Length = 8，则 Number\_Bytes × Burst\_Length = 16，Wrap\_Boundary = 160。由此可知，上文中Wrap模式的使用场景2）的描述不准确。

对于换行突发，如果 Address\_N = Wrap\_Boundary + (Number\_Bytes × Burst\_Length)，则：

Address\_N = Wrap\_Boundary；

如果Address\_N < Wrap\_Boundary + (Number\_Bytes × Burst\_Length)，则：

Address\_N = Aligned\_Address + (N – 1) × Number\_Bytes，N >= 2，N <= Burst\_Length

第一次Incr模式的transfer

Lower\_Byte\_Lane = Start\_Address – (INT(Start\_Address / Data\_Bus\_Bytes)) × Data\_Bus\_Bytes

= Address\_1 – Aligned\_Address

假如Start\_Address = 10， Data\_Bus\_Bytes = 4，则Aligned\_Address = 8，Lower\_Byte\_Lane = 2，也就是第3通道。所有通道是 Byte\_Lane0，Byte\_Lane1，Byte\_Lane2，Byte\_Lane3。

Upper\_Byte\_Lane = Number\_Bytes – 1

此后的Incr模式的transfer

Lower\_Byte\_Lane = 0

Upper\_Byte\_Lane = Number\_Bytes – 1

事务容器描述了如果地址对齐并且声明了（数据）选通脉冲，该事务中可以访问的所有字节：

Container\_Size = Number\_Bytes x Burst\_Length

对于INCR突发：

Container\_Lower = Aligned\_Address

Container\_Upper = Aligned\_Address + Container\_Size

对于WRAP突发：

Container\_Lower = Wrap\_Boundary

Container\_Upper = Wrap\_Boundary + Container\_Size

2、AXI定义了常规属性（Regular\_Transactions\_Only），以标识符合以下标准的事务：

* AxLEN是1、2、4、8或16。
* 如果AxLEN大于1，AxSIZE与数据总线宽度相同。
* AxBURST是INCR或WRAP，而不是FIXED。
* AxADDR与INCR事务的事务容器对齐。
* 对于WRAP事务，AxADDR与AxSIZE对齐。

Regular\_Transactions\_Only：

TRUE：Only Regular transactions are supported.  
FALSE：All legal combinations of AxBURST, AxSIZE, and AxLEN are supported.

3、AXI协议为读和写事务提供响应信令：

对于读事务，来自下属的响应信息在读数据通道上发出信号。

对于写事务，响应信息在写响应通道上发出信号。

响应通过以下方式发出信号：

RRESP[1:0]，用于读传输。

BRESP[1:0]，用于写传输。

响应如下：

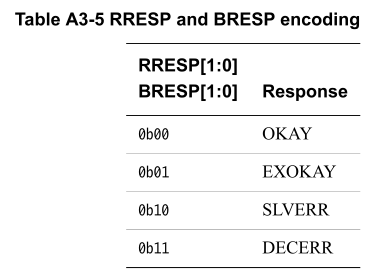
OKAY——正常访问成功。表示正常访问已成功。也可以表示独占访问失败。

EXOKAY——独家访问成功。表示独占访问的读或写部分已经成功。

SLVERR——从属错误。当访问成功到达下属，但下属希望向发起管理器返回错误情况时使用。

DECERR——解码错误。通常由互连组件生成，用于指示事务地址没有下属。

表A3-5显示了RRESP和BRESP信号的编码。



对于写事务，会针对整个突发发出单个响应信号，而不是针对突发中的每个数据传输。

在读事务中，从属端可以用信号通知突发中不同传输的不同响应。例如，在16次读取传输的突发中，从属端可能会为其中15次传输返回一个ok响应，为其中一次传输返回一个SLVERR响应。

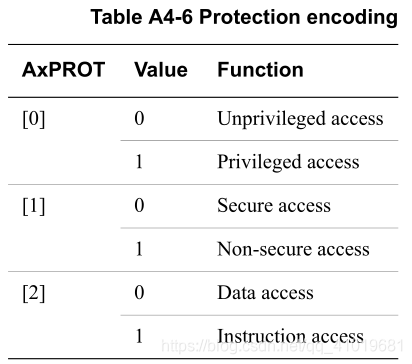
该协议规定，即使报告了错误，也必须执行所需数量的数据传输。例如，如果从下属请求读取八次传输，但下属有错误情况，则下属必须执行八次数据传输，每次都有错误响应。如果从属设备给出单个错误响应，突发的剩余部分不会被取消。

4、AXI提供访问许可信号，可用于防范非法事务：

ARPROT[2:0]定义了读取访问的访问权限。

AWPROT[2:0]定义了写访问的访问权限。

表A4-6显示了AxPROT[2:0]编码。



保护属性包括：

无特权的或有特权的：AXI管理器可能支持多种级别的操作权限，并将这种权限概念扩展到内存访问。AxPROT[0]将访问标识为非特权或特权。

安全还是不安全：AXI管理器可能支持安全和非安全操作状态，并将这种安全概念扩展到内存访问。AxPROT[1]将访问标识为安全或非安全。AxPROT[1]可以被认为定义了两个地址空间，一个安全地址空间和一个非安全地址空间。该信号可以被视为一个附加的地址位。必须正确处理安全和非安全地址空间之间的任何混淆。

指令或数据：该位表示事务是指令访问还是数据访问。AXI协议将这一迹象定义为一种暗示。它不是在所有情况下都准确，例如，当事务包含指令和数据项的混合时。本规范建议管理器将AxPROT[2]设置为低电平，以指示数据访问，除非已知该访问是指令访问。

5、AXI协议包括AXI ID事务标识符。管理器可以使用这些信息来确定必须按顺序返回的单独事务。

具有给定AXI标识值的所有事务都必须保持有序，但是对具有不同标识值的事务的排序没有限制。单个物理端口可以通过充当多个逻辑端口来支持无序事务，每个逻辑端口都按顺序处理事务。

通过使用AXI标识，管理器可以发布事务，而无需等待较早的事务完成。这可以提高系统性能，因为它支持事务的并行处理。

##### ID 信号，每个事务通道都有自己的事务标识。表A5-1显示了这些指定信号。

###### 读数据排序

RID值与它所响应的地址的ARID值相匹配。

互连必须确保从一系列事务中读取的数据具有相同的ARID值，目标是不同的下属，管理器按照其发布地址的顺序接收这些数据。

###### 写数据排序

组合来自不同管理器的写事务的互连必须确保按地址顺序转发写数据。AXI3允许不同标识的写数据交错，但AXI4及更高版本不推荐使用。

##### AXI排序模式概述

AXI排序模型基于事务标识符的使用，事务标识符在ARID或AWID上发出信号，从属端通过RID或WID相应。具有相同标识和目的地的同一通道上的事务请求保证保持有序。

排序模型不提供以下两者之间的任何排序保证：

* 来自不同管理器的事务
* 读写事务
* 具有不同标识的事务
* 到不同外设区域的事务
* 到不同内存位置的事务

如果管理器要求在没有排序保证的事务之间排序，则管理器必须在发出第二个事务之前等待收到对第一个事务的响应。

存储位置和外围区域

AMBA的地址映射由内存位置和外设区域组成。

内存位置具有以下所有属性：

从内存位置读取字节会返回写入该字节位置的最后一个值。

对内存位置字节的写入会将该位置的值更新为新值，该新值是通过后续读取该位置获得的。

读取或写入内存位置不会对任何其他内存位置产生副作用。

每个位置都有内存观察保证。？

内存位置的大小等于该组件的单副本原子性大小。

外围区域具有以下所有属性：

从外围区域中的地址读取不一定返回写入该地址的最后一个值。

对外围区域字节地址的写入不一定会将该地址的值更新为后续读取获得的新值。

访问外围区域内的地址可能会对该区域内的其他地址产生副作用。

每个地区都有外设观察保证。

外围区域的大小由实现定义，但必须包含在单个从属组件中。

事务是对一个或多个地址位置的读取或写入。这些位置由AxADDR和任何相关限定符（如AxPROT中的非安全位）决定。

事务可以是“设备”或“正常”类型：

Device：一种读或写操作，其中请求的AxCACHE[1]被取消声明。设备事务可用于访问外围区域或内存位置。

Normal：一种读或写操作，其中请求断言了AxCACHE[1]。正常事务用于访问内存位置，预计不会用于访问外围区域。对外围区域的正常访问必须以符合协议的方式完成，但结果是实现已定义。

写事务可以是不可缓冲的，也可以是可缓冲的。有可能对可缓冲的写入发送早期响应。

* 不可缓冲的写操作取消声明了AWCACHE[0]。
* 缓冲写入断言了AWCACHE[0]。

观察和完成定义

对于对外围区域的访问，当设备读或写访问DRW1先于DRW2到达从属组件时，设备读或写访问DRW1被设备读或写访问DRW2观察到。

对于对内存位置的访问，以下所有条件都适用：

如果W2在W1之后生效，写W1由写W2观察。

当W2在W3之后时，如果R1从写W3返回数据，则写W2观察到读R1。

当W3在W1之后时，如果R2从W1或写W3返回数据，读R2会观察到写W1。

读R1或写W1可以是设备或正常类型。写和读完成的定义是：

写完成响应：给出相关的BRESP握手的周期，此时BVALID和BREADY被断言。

读完成响应：给出最后一次相关RDATA握手的周期，此时断言RVALID、RLAST和RREADY。

有序写观察

为了提高与支持不同排序模型（是一些琐碎的抽象要求，要看如何实现）的接口协议的兼容性，从属接口可以为写事务提供更强的排序保证。更强的排序保证称为有序写观察。有序写观察属性用于定义接口是否显示有序写观察，对于单个接口，它可以是真或假。

True：接口被定义为具有“有序写入观察”属性。

False：没有“有序写入观察”属性的接口。如果未声明Ordered\_Write\_Observation，则认为它为False。

呈现有序写入观察的接口为不依赖于目的地或地址的写入事务提供保证：

写W1保证被写W2观察到，其中W2在W1之后从相同的管理器发出，具有相同的标识。

使用Producer-Consumer排序模型的管理器连接到显示有序写观察的从属接口，在发出相关写入之前，不需要等待来自早期写入的完成响应。

6、原子访问

原子事务确保的是，没有一个管理者能够观察到原子数据的部分更新形式。例如，在许多系统中，数据结构(如链表)由32位原子元素组成。这些元素之一的原子更新需要同时更新整个32位值。任何管理器都不能一次只观察到16位的更新，然后再观察到其他16位的更新。）事务的原子性保证永远不会大于其起始地址的对齐。例如，64位单拷贝原子组中可保证没有任何未与8字节边界对齐的突发64位拷贝。单拷贝指仅有一份拷贝。

7、独占访问

独占访问机制可以提供信号量类型的操作，而不需要总线在操作期间保持专用于特定的管理器。这意味着信号量类型的操作既不影响总线访问延迟，也不影响最大可达到的带宽。

AxLOCK信号选择独占访问，RRESP和BRESP信号分别指示独占访问读或写的成功或失败。

从属需要额外的逻辑来支持独占访问。AXI协议提供了一种机制来指示管理者何时尝试对不支持它的下属进行独占访问。

##### 8、 锁定访问

AXI4不支持锁定的事务。但是，AXI3实现必须支持锁定事务。

当管理器使用某个事务的AxLOCK信号来表明它是锁定的事务时，则互连必须确保只有该管理器才能访问目标从属区域，直到来自同一管理器的后续未锁定事务完成（最后执行解锁事务）。互连中的仲裁器必须实施这一限制。

9、默认信号和互操作性

AXI协议不要求组件使用AXI接口上可用的全套信号。为了帮助连接不使用每个信号的组件，本章定义了接口的主要类别以及适用于每个类别的限制。

A9.1 互操作性原则

以下互操作性原则适用于AXI3和AXI4组件。

作为一般原则，组件必须支持所有输入组合，但不必生成所有输出组合。例如，下属必须支持所有可能的不同突发长度，而管理器只需生成其使用的突发类型。此策略确保所有组件与所有其他组件一起工作。

AXI接口可以省略信号的条件是：

可选输出

如果组件可能需要与默认值不匹配的值，则该组件必须有输出信号。

可选输入

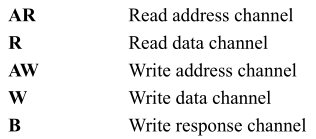
如果管理器或下属不需要观察输入信号来进行正确的功能操作，则可以省略输入信号。

（适当时，互连组件也可以省略信号。例如，当一个信号只被驱动到其默认值时，不需要通过互连传输该信号。信号可以在目的地产生。同样，如果信号没有在任何目的地使用，则不需要通过互连传输。）

A9.2主要接口类别

A9.2.1 读/写接口

读写接口包括以下AXI通道：



A9.2.2 只读接口

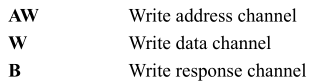
只读接口仅支持读事务，包括以下AXI通道：



（只读接口不支持独占访问。）

A9.2.3 只写接口

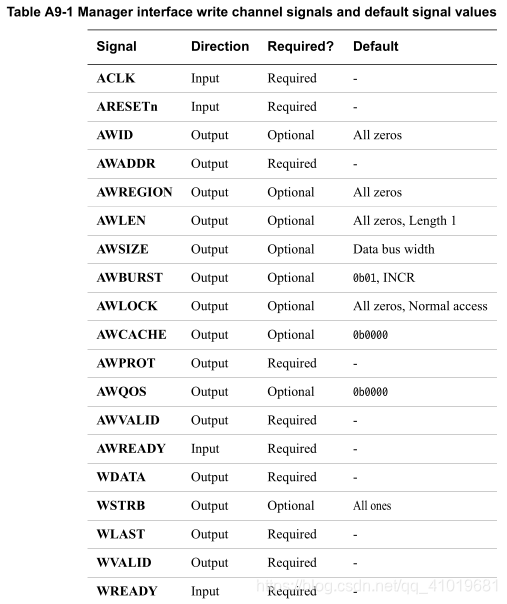
只写接口仅支持写事务，包括以下AXI通道：

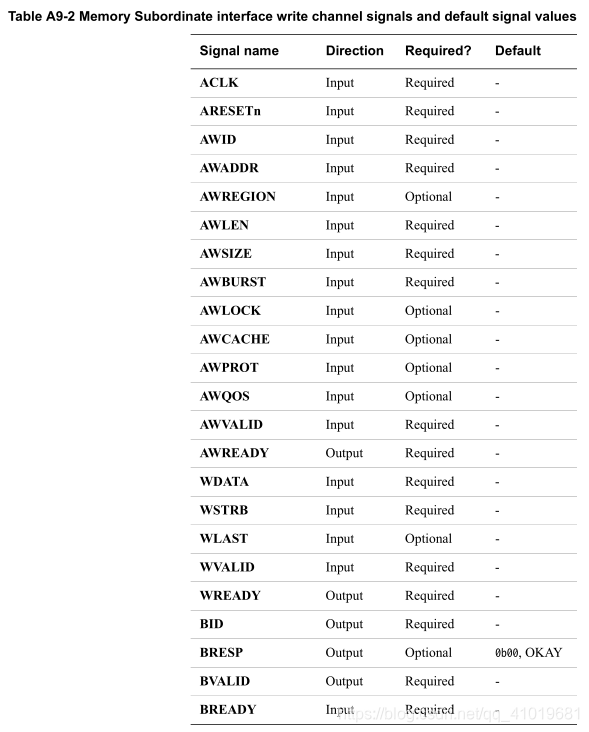


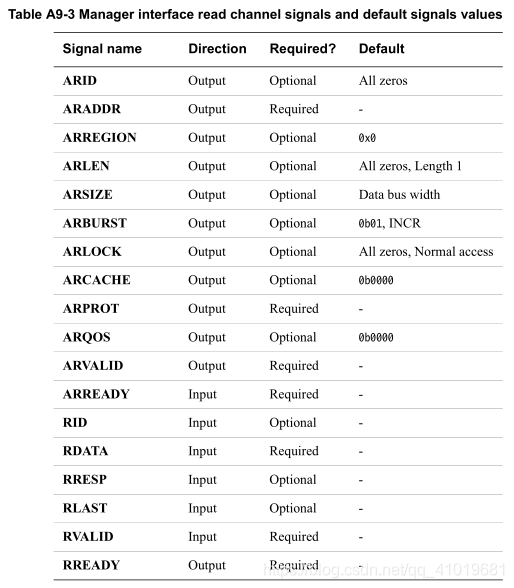
（只写接口不支持独占访问。）

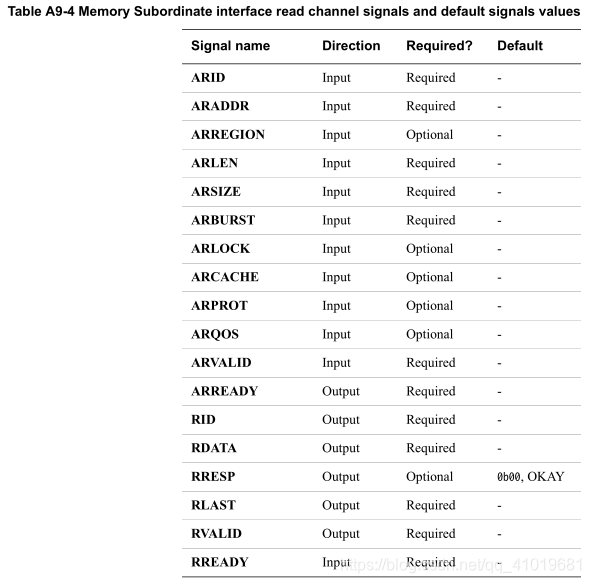
##### A9.3 默认信号值

该规范建议，一般来说，为了最大限度地重用IP，AXI组件接口包括所有信号。所有信号的存在降低了设计流程的系统集成阶段出错的风险，并且还可以帮助支持一些不有效支持缺省信号默认值的设计流程。









A9.3.1 管理器地址

AxADDR

对管理器提供的地址位数没有最低要求。如果管理器连接的系统的地址总线宽度与管理器提供的不同：

如果系统地址比管理器提供的更宽，则必须为附加的高位地址位使用全零的默认值。

如果系统地址比管理器提供的地址窄，则管理器的高位地址位必须保持不连接。

（通常，管理器提供32位寻址，或者管理器最多支持64位寻址。）

A9.3.2 从属地址

AxADDR

对下属使用的地址位数没有最低要求。

下属不需要具有低阶地址位来支持系统数据总线宽度内的解码，并且可以假设这种低阶地址位具有全零的默认值。如果从属设备的地址位多于互连提供的地址位，则高阶地址位使用全零的默认值。

通常，内存从属设备至少有足够的地址位来完全解码4KB地址范围。

A9.3.3 内存从属

AxLOCK

内存从属不需要使用AxLOCK输入。然而，支持独占访问的内存从属需要这些信号。

AxCACHE

不需要从属内存来使用AxCACHE输入。在下列情况下，存储器从属设备不需要这些信号：

它没有缓存行为。

它以同样的方式缓存所有事务。

A9.3.4 写事务

WSTRB

如果管理器总是执行全数据总线宽度的写事务，则不需要使用写选通信号。写选通脉冲的默认值是所有信号有效。

WLAST

不要求下属使用WLAST信号。由于定义了写突发的长度，从属端可以根据突发长度AWLEN[7:0]信号计算最后一次写数据传输。

A9.3.5 读事务

RLAST

管理器不需要使用RLAST信号。由于读取脉冲串的长度已定义，管理器可以根据脉冲串长度ARLEN[7:0]信号计算最后一次读取数据传输。

A9.3.6 响应信号

RRESP, BRESP

管理器不需要RRESP和BRESP输入，如果它们都：

不执行独占访问

不要求通知事务错误

下属不需要RRESP和BRESP输出，如果它都：

不支持独占访问

不生成错误响应

A9.3.7 非安全和安全访问

AxPROT

AxPROT不需要区分非安全访问和安全访问，也不需要任何额外保护支持的从属设备，不需要AxPROT输入信号。

（注意AxPROT信号。AxPROT[1]信号指示事务的安全或非安全性质，这些位的不正确分配会导致不正确的系统行为。）