**AXI协议调研报告**

目录

[一、AXI协议简介 2](#_Toc529974187)

[1.1 什么是AXI 2](#_Toc529974188)

[1.2 AXI的特点 2](#_Toc529974189)

[1.3 AXI的性能 3](#_Toc529974190)

[1.4 AXI的架构 3](#_Toc529974191)

[1.5 AXI4协议 5](#_Toc529974192)

[1.5.1 AXI4接口类型 5](#_Toc529974193)

[1.5.2 AXI4协议优势 6](#_Toc529974194)

[1.5.3 AXI4的工作模式 6](#_Toc529974195)

[1.5.3.1 握手机制 6](#_Toc529974196)

[1.5.3.2 AXI工作模式 7](#_Toc529974197)

[1.5.3.3 AXIS工作模式 8](#_Toc529974198)

[二、AXI协议信号描述 9](#_Toc529974199)

[2.1 信号描述 9](#_Toc529974200)

[2.2 信号接口要求 11](#_Toc529974201)

[2.2.1 时钟复位 11](#_Toc529974202)

[2.2.2通道信号 11](#_Toc529974203)

[2.2.2.1 通道间关系 12](#_Toc529974204)

[2.2.2.2 通道握手信号的依赖关系 12](#_Toc529974205)

[3.3传输结构 13](#_Toc529974206)

[3.3.1 地址结构 13](#_Toc529974207)

[3.3.2 数据读写结构 14](#_Toc529974208)

[3.3.2.1 窄传输 14](#_Toc529974209)

[3.3.2.2 非对齐传输 15](#_Toc529974210)

[3.3.3 读写响应结构 17](#_Toc529974211)

[四、AXI与FPGA 18](#_Toc529974212)

# 一、AXI协议简介

## 1.1 什么是AXI

AXI（Advanced eXtensible Interface）是一种总线协议，该协议的第一个版本AXI3是ARM公司提出的AMBA（Advanced Microcontroller Bus Architecture）3.0协议中最重要的部分，是一种面向高性能、高带宽、低延迟的片内总线，也是用来替代以前的AHB和APB总线的。它的地址/控制和数据相位是分离的，支持不对齐的数据传输，同时在突发传输中，只需要首地址，同时分离的读写数据通道、并支持Outstanding传输访问和乱序访问，并更加容易进行时序收敛。AXI 是AMBA 中一个新的高性能协议。AXI 技术丰富了现有的AMBA 标准内容，满足超高性能和复杂的片上系统（SoC）设计的需求。

AMBA AXI协议支持高性能、高频率系统设计。

* 适合高带宽低延时设计
* 无需复杂的桥就能实现高频操作
* 能满足大部分器件的接口要求
* 适合高初始延时的存储控制器
* 提供互联架构的灵活性与独立性
* 向下兼容已有的AHB和APB接口

## 1.2 AXI的特点

单向通道体系结构。信息流只以单方向传输，简化时钟域间的桥接，减少门数量。当信号经过复杂的片上系统时，减少延时。

支持多项数据交换。通过并行执行猝发操作，极大地提高了数据吞吐能力，可在更短的时间内完成任务，在满足高性能要求的同时，又减少了功耗。

独立的地址和数据通道。地址和数据通道分开，能对每一个通道进行单独优化，可以根据需要控制时序通道，将时钟频率提到最高，并将延时降到最低。

增强的灵活性。AXI技术拥有对称的主从接口，无论在点对点或在多层系统中，都能十分方便地使用AXI技术。

关键特点：

• 分离的地址/控制、数据相位

• 使用字节线来支持非对齐的数据传输

• 使用基于burst的传输，只需传输首地址

• 分离的读、写数据通道，能提供低功耗DMA

• 支持多种寻址方式

• 支持乱序传输

• 允许容易的添加寄存器级来进行时序收敛

## 1.3 AXI的性能

AXI 能够使SoC 以更小的面积、更低的功耗，获得更加优异的性能。AXI 获得如此优异性能的一个主要原因，就是它的单向通道[体系结构](https://baike.baidu.com/item/%E4%BD%93%E7%B3%BB%E7%BB%93%E6%9E%84)。单向通道体系结构使得片上的信息流只以单方向传输，减少了延时。

选择采用何种总线，我们要评估到底怎样的总线频率才能满足我们的需求，而同时不会消耗过多的功耗和片上面积。ARM一直致力于以最低的成本和功耗追求更高的性能。这一努力已经通过连续一代又一代处理器内核的发布得到了实现，每一代新的处理器内核都会引入新的流水线设计、新的[指令集](https://baike.baidu.com/item/%E6%8C%87%E4%BB%A4%E9%9B%86)以及新的高速缓存结构。这促成了众多创新移动产品的诞生，并且推动了ARM架构向性能、功耗以及成本之间的完美平衡发展。

AXI总线是一种多通道传输总线，将地址、读数据、写数据、握手信号在不同的通道中发送，不同的访问之间顺序可以打乱，用BUSID来表示各个访问的归属。主设备在没有得到返回数据的情况下可发出多个读写操作。读回的数据顺序可以被打乱，同时还支持非对齐数据访问。

AXI总线还定义了在进出低功耗节电模式前后的[握手协议](https://baike.baidu.com/item/%E6%8F%A1%E6%89%8B%E5%8D%8F%E8%AE%AE)。规定如何通知进入低功耗模式，何时关断时钟，何时开启时钟，如何退出低功耗模式。这使得所有IP在进行功耗控制的设计时，有据可依，容易集成在统一的系统中。AXI与上一代总线AHB的主要性能比较见表1。





## 1.4 AXI的架构

AXI协议是基于burst的传输，并且定义了以下5个独立的传输通道：读地址通道、读数据通道、写地址通道、写数据通道、写响应通道。

地址通道携带控制消息用于描述被传输的数据属性，数据传输使用写通道来实现“主”到“从”的传输，“从”使用写响应通道来完成一次写传输；读通道用来实现数据从“从”到“主”的传输。

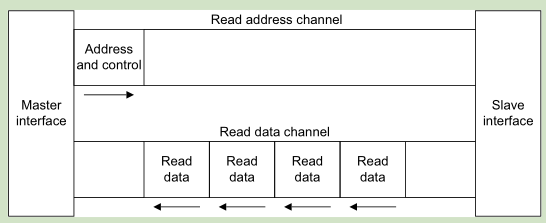


图1 AXI读架构

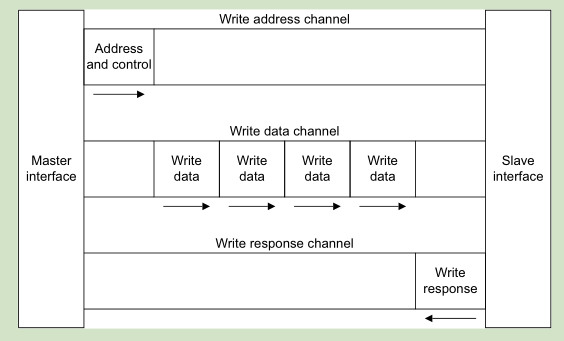


图2 AXI写架构

AXI是基于VALID/READY的握手机制数据传输协议，传输源端使用VALID表明地址/控制信号、数据是有效的，目的端使用READY表明自己能够接受信息。数据可以在主从设备间同步的双向传输，并且数据传输大小可以改变。AXI4将数据传输的突发长度限制为最大256，AXI4-Lite每次传输仅运输传输一个数据。

读/写地址通道：读、写传输每个都有自己的地址通道，对应的地址通道承载着对应传输的地址控制信息。

读数据通道：读数据通道承载着读数据和读响应信号,包括数据总线（8/16/32/64/128/256/512/1024bit）和指示读传输完成的读响应信号。

写数据通道：写数据通道的数据信息被认为是缓冲（buffered）了的，“主”无需等待“从”对上次写传输的确认即可发起一次新的写传输。写通道包括数据总线（8/16…1024bit）和字节线（用于指示8bit 数据信号的有效性）。

写响应通道：“从”使用写响应通道对写传输进行响应。所有的写传输需要写响应通道的完成信号。

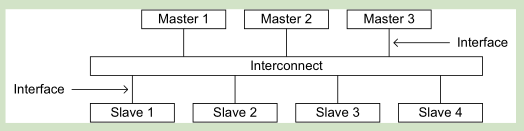


图3 AXI接口与互联

AXI协议提供单一的接口定义，能用在下述三种接口之间：master/interconnect、slave/interconnect、master/slave。AXI协议严格的讲是一个点对点的主从接口协议，当多个外设需要互相交互数据时，需要加入一个AXI Interconnect模块，也就是AXI互联矩阵，作用是提供将一个或多个AXI主设备连接到一个或多个AXI从设备的一种交换机制（有点类似于交换机里面的交换矩阵）。

可以使用以下几种典型的系统拓扑架构：

* 共享地址与数据总线
* 共享地址总线，多数据总线
* mul[ti](http://www.elecfans.com/tags/ti/)layer多层，多地址总线，多数据总线

在大多数系统中，地址通道的带宽要求没有数据通道高，因此可以使用共享地址总线，多数据总线结构来对系统性能和互联复杂度进行平衡。

寄存器片（Register Slices）：每个AXI通道使用单一方向传输信息，并且各个通道直接没有任何固定关系。因此可以在任何通道任何点插入寄存器片，当然这会导致额外的周期延迟。

使用寄存器片可以实现周期延迟（cycles of latency）和最大操作频率的折中；使用寄存器片可以分割低速外设的长路径。

## 1.5 AXI4协议

2010发布的AMBA4.0包含了AXI的第二个版本AXI4。

## 1.5.1 AXI4接口类型

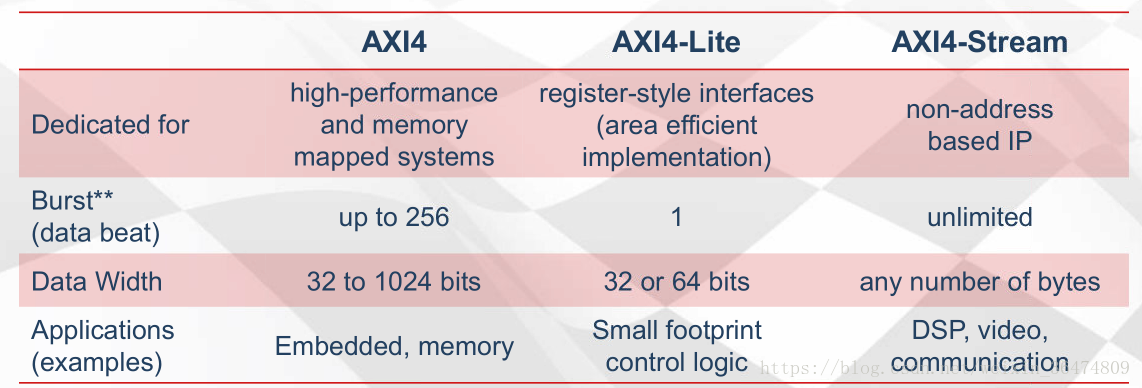
AXI4包含3种类型的接口：

1）AXI4：相当于原来的AHB协议，主要面向高性能地址映射通信的需求，在单地址传输的情况下最大允许256个时钟周期的数据突发长度，可以连续对一片地址进行一次性读写。主要用于处理器访问存储等需要高速数据的场合；

2）AXI4-Lite：相当于原来的APB协议，是一个轻量级的地址映射单次传输接口，是AXI的简化版本，占用较少的资源，不支持批量传输，读写时一次只能读写一个字（32bit），适用于吞吐量较小的地址映射通信总线，用于访问一些低速外设；

3）AXI4-Stream：是一种连续流接口，面向高速流数据传输，去掉了地址传输的功能，允许无限制的数据突发传输，无需考虑地址映射。由ARM公司和Xilinx公司共同提出，主要用在FPGA进行以数据为主导的大量数据的传输应用，如视频、高速AD、PCIe、DMA接口等，跟Xilinx原来的Local Link协议类似。AXI4-Stream协议是一种用来连接需要交换数据的两个部件的标准接口，它可以用于连接一个产生数据的主机和一个接受数据的从机。当然它也可以用于连接多个主机和从机。该协议支持多种数据流使用相同共享总线集合，允许构建类似于路由、宽窄总线、窄宽总线等更为普遍的互联。

表2 AXI4接口类型



AXI4总线分为主、从两端，两者之间可以连续地进行通信。

ISE从12.3版本，Virtex6，Spartan6芯片开始对AXI4总线提供支持，并且随着Xilinx与ARM的合作面逐渐展开而得到大力推广。新一代FPGA中采用的基本都是AXI4总线协议，例如与slaver侧的DMA或DDR等通信。

## 1.5.2 AXI4协议优势

1）高效：通过标准化的AXI接口，开发者只需要学习一种IP核的通讯协议即可；

2）易用：针对具体应用提供合适的接口协议；AXI4是面向地址映射的接口，允许最大256轮的数据突发传输；AXI4-Lite是一个轻量级的地址映射单次传输接口，占用很少的逻辑单元；AXI4-Stream去掉了地址项，允许无限制的数据突发传输规模；

3）易得：标准化的AXI接口协议资源，不仅可以在xilinx官网上获得，也可以在全球范围内ARM的所有合作伙伴处获得。大量的IP core支持AXI4协议；大量的第三方AXI工具可提供多样的系统开发、验证和功能定制。

NetFPGA10G采用了AXI4系列总线架构，而Xilinx从Virtex6、Spartan6才开始支持AXI4。因此斯坦福的开发人员对在Virtex6上编译出的带有AXI4总线的ngc文件进行了修改加载到NetFPGA10G的ipcore中，使之骗过ISE，继续网表映射流程。

NetFPGA10G主要使用了其中的轻量级AXI4-lite（后文简写为axi）以及AXI4-stream（后文简写为axis）两者。前者用于CPU与ip core之间的通信；后者用于各ip core之间进行高速数据传输。如果将整个构架分为控制层面与数据层面，则axi主要负责控制层面，axis主要负责数据层面。

## 1.5.3 AXI4的工作模式

### 1.5.3.1 握手机制

AXI4所采用的是一种READY，VALID握手通信机制，即主从模块进行数据通信前，新根据操作对各所用到的数据、地址通道进行握手。主要操作包括传输发送者A等到传输接受者B的READY信号后，A将数据与VALID信号同时发送给B。以下图片是几种握手机制：

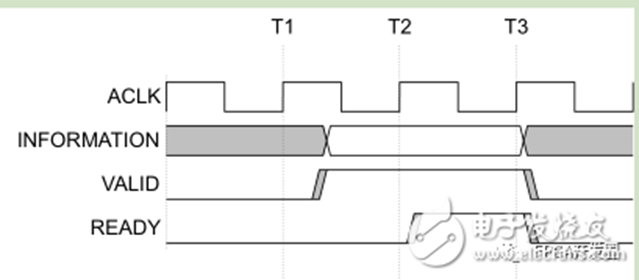


图4 VALID before READY 握手

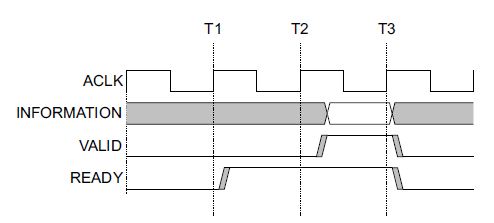


图5 READY before VALID 握手

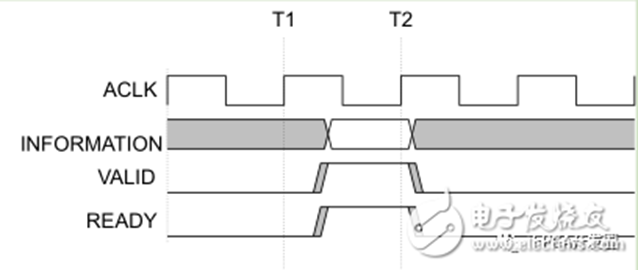


图6 VALID with READY 握手

### 1.5.3.2 AXI工作模式

AXI总线分为五个通道，其中每个通道都是一个独立的AXI握手协议：

读地址通道，包含ARVALID, ARADDR, ARREADY信号；

写地址通道，包含AWVALID，AWADDR, AWREADY信号；

读数据通道，包含RVALID, RDATA, RREADY, RRESP信号；

写数据通道，包含WVALID, WDATA，WSTRB, WREADY信号；

写应答通道，包含BVALID, BRESP, BREADY信号；

系统通道，包含：ACLK，ARESETN信号；

其中ACLK为AXI总线时钟，ARESETN是AXI总线复位信号，低电平有效；读写数据与读写地址类信号宽度都为32bit；READY与VALID是对应的通道握手信号；WSTRB信号为1的bit对应WDATA有效数据字节，WSTRB宽度是32bit/8=4bit；BRESP与RRESP分别为写回应信号，读回应信号，宽度都为2bit，‘h0代表成功，其他为错误。

**读操作**

顺序为主与从进行读地址通道握手并传输地址内容，然后在读数据通道握手并传输所读内容以及读取操作的回应，时钟上升沿有效。如图所示：

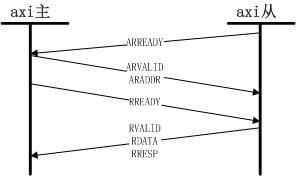


图7 AXI读操作

**写操作**

顺序为主与从进行写地址通道握手并传输地址内容，然后在写数据通道握手并传输所读内容，最后再写回应通道握手，并传输写回应数据，时钟上升沿有效。如图所示：

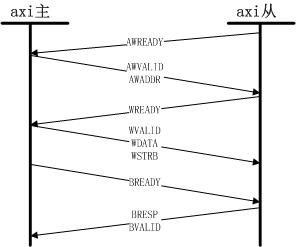


图8 AXI写操作

### 1.5.3.3 AXIS工作模式

AXIS分为：

tready信号：从告诉主做好传输准备；

tvalid信号：主告诉从数据传输有效；

tlast信号：主告诉从该次传输为突发传输结尾；

tdata信号：数据，可选宽度32,64,128,256bit；

tstrb信号：为1的bit为对应tdata有效字节，宽度为tdata/8；

tuser信号：用户定义信号，宽度为128bit；

aclk信号：总线时钟，上升沿有效；

aresetn信号：总线复位，低电平有效；

通信时序如图所示：

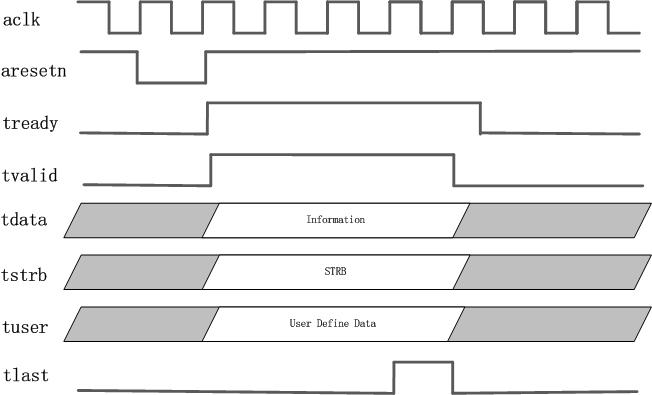


图9 AXIS通信时序

AXI与AXIS是AXI4总线中通信复杂度较低的两条总线，最大开发难度存在于AXI的控制平面向AXIS的数据平面下发参数时，由于AXI与AXIS时钟频率不同而产生的跨时钟域数据传输问题。

# 二、AXI协议信号描述

## 2.1 信号描述

表2-1 全局信号

|  |  |  |
| --- | --- | --- |
| 信号名 | 源 | 描述 |
| ACLK | [时钟](http://www.elecfans.com/tags/%E6%97%B6%E9%92%9F/)源 | 全局时钟信号 |
| ARESETn | 复位源 | 全局复位信号，低有效 |

表2-2 写地址通道信号

|  |  |  |
| --- | --- | --- |
| 信号名 | 源 | 描述 |
| AWID | 主机 | 写地址ID，用来标志一组写信号 |
| AWADDR | 主机 | 写地址，给出一次写突发传输的写地址 |
| AWLEN | 主机 | 突发长度，给出突发传输的次数 |
| AWSIZE | 主机 | 突发大小，给出每次突发传输的字节数 |
| AWBURST | 主机 | 突发类型 |
| AWLOCK | 主机 | 总线锁信号，可提供操作的原子性 |
| AWCACHE | 主机 | 内存类型，表明一次传输是怎样通过系统的 |
| AWPROT | 主机 | 保护类型，表明一次传输的特权级及安全等级 |
| AWQOS | 主机 | 质量服务QoS |
| AWREGION | 主机 | 区域标志，能实现单一物理接口对应的多个逻辑接口 |
| AWUSER | 主机 | 用户自定义信号 |
| AWVALID | 主机 | 有效信号，表明此通道的地址控制信号有效 |
| AWREADY | 从机 | 表明“从”可以接收地址和对应的控制信号 |

表2-3 写数据通道信号

|  |  |  |
| --- | --- | --- |
| 信号名 | 源 | 描述 |
| WID | 主机 | 一次写传输的ID tag |
| WDATA | 主机 | 写数据 |
| WSTRB | 主机 | 写数据有效的字节线，用来表明哪8bits数据是有效的 |
| WLAST | 主机 | 表明此次传输是最后一个突发传输 |
| WUSER | 主机 | 用户自定义信号 |
| WVALID | 主机 | 写有效，表明此次写有效 |
| WREADY | 从机 | 表明从机可以接收写数据 |

表2-4 写响应通道信号

|  |  |  |
| --- | --- | --- |
| 信号名 | 源 | 描述 |
| BID | 从机 | 写响应ID tag |
| BRESP | 从机 | 写响应，表明写传输的状态 |
| BUSER | 从机 | 用户自定义 |
| BVALID | 从机 | 写响应有效 |
| BREADY | 主机 | 表明主机能够接收写响应 |

表2-5 读地址通道信号

|  |  |  |
| --- | --- | --- |
| 信号名 | 源 | 描述 |
| ARID | 主机 | 读地址ID，用来标志一组写信号 |
| ARADDR | 主机 | 读地址，给出一次写突发传输的读地址 |
| ARLEN | 主机 | 突发长度，给出突发传输的次数 |
| ARSIZE | 主机 | 突发大小，给出每次突发传输的字节数 |
| ARBURST | 主机 | 突发类型 |
| ARLOCK | 主机 | 总线锁信号，可提供操作的原子性 |
| ARCACHE | 主机 | 内存类型，表明一次传输是怎样通过系统的 |
| ARPROT | 主机 | 保护类型，表明一次传输的特权级及安全等级 |
| ARQOS | 主机 | 质量服务QoS |
| ARREGION | 主机 | 区域标志，能实现单一物理接口对应的多个逻辑接口 |
| ARUSER | 主机 | 用户自定义信号 |
| ARVALID | 主机 | 有效信号，表明此通道的地址控制信号有效 |
| ARREADY | 从机 | 表明“从”可以接收地址和对应的控制信号 |

表2-6 读数据通道信号

|  |  |  |
| --- | --- | --- |
| 信号名 | 源 | 描述 |
| RID | 从机 | 读ID tag |
| RDATA | 从机 | 读数据 |
| RRESP | 从机 | 读响应，表明读传输的状态 |
| RLAST | 从机 | 表明读突发的最后一次传输 |
| RUSER | 从机 | 用户自定义 |
| RVALID | 从机 | 表明此通道信号有效 |
| RREADY | 主机 | 表明主机能够接收读数据和响应信息 |

表2-7 低功耗接口信号

|  |  |  |
| --- | --- | --- |
| 信号名 | 源 | 描述 |
| CSYSREQ | 时钟控制器 | 系统退出低功耗请求，信号从“时钟控制器”到“外设” |
| CSYSACK | 外设 | 退出低功耗状态确认 |
| CACTIVE | 外设 | 外设请求时钟有效 |

## 2.2 信号接口要求

## 2.2.1 时钟复位

时钟：每个AXI组件使用一个时钟信号ACLK，所有输入信号在ACLK上升沿采样，所有输出信号必须在ACLK上升沿后发生。

复位：AXI使用一个低电平有效的复位信号ARESETn，复位信号可以异步断言，但必须和时钟上升沿同步去断言。

复位期间对接口有如下要求：①主机接口必须驱动ARVALID，AWVALID，WVALID为低电平；②从机接口必须驱动RVALID，BVALID为低电平；③所有其他信号可以被驱动到任意值。

在复位后，主机可以在时钟上升沿驱动ARVALID，AWVALID，WVALID为高电平。

## 2.2.2通道信号

通道握手信号：每个通道有自己的xVALID/xREADY握手信号对。

写地址通道：当主机驱动有效的地址和控制信号时，主机可以断言AWVALID，一旦断言，需要保持AWVALID的断言状态，直到时钟上升沿采样到从机的AWREADY。AWREADY默认值可高可低，推荐为高（如果为低，一次传输至少需要两个周期，一个用来断言AWVALID，一个用来断言AWREADY）；当AWREADY为高时，从机必须能够接受提供给它的有效地址。

写数据通道：在写突发传输过程中，主机只能在它提供有效的写数据时断言WVALID，一旦断言，需要保持断言状态，知道时钟上升沿采样到从机的WREADY。WREADY默认值可以为高，这要求从机总能够在单个周期内接受写数据。主机在驱动最后一次写突发传输是需要断言WLAST信号。

写响应通道：从机只能它在驱动有效的写响应时断言BVALID，一旦断言需要保持，直到时钟上升沿采样到主机的BREADY信号。当主机总能在一个周期内接受写响应信号时，可以将BREADY的默认值设为高。

读地址通道：当主机驱动有效的地址和控制信号时，主机可以断言ARVALID，一旦断言，需要保持ARVALID的断言状态，直到时钟上升沿采样到从机的ARREADY。ARREADY默认值可高可低，推荐为高（如果为低，一次传输至少需要两个周期，一个用来断言ARVALID，一个用来断言ARREADY）；当ARREADY为高时，从机必须能够接受提供给它的有效地址。

读数据通道：只有当从机驱动有效的读数据时从机才可以断言RVALID，一旦断言需要保持直到时钟上升沿采样到主机的BREADY。BREADY默认值可以为高，此时需要主机任何时候一旦开始读传输就能立马接受读数据。当最后一次突发读传输时，从机需要断言RLAST。

### 2.2.2.1 通道间关系

AXI协议要求通道间满足如下关系：

* 写响应必须跟随最后一次burst的的写传输；
* 读数据必须跟随数据对应的地址；

### 2.2.2.2 通道握手信号的依赖关系

为防止死锁，通道握手信号需要遵循一定的依赖关系。

①VALID信号不能依赖READY信号。

②AXI接口可以等到检测到VALID才断言对应的READY，也可以检测到VALID之前就断言READY。下面有几个图表明依赖关系，单箭头指向的信号能在箭头起点信号之前或之后断言；双箭头指向的信号必须在箭头起点信号断言之后断言。

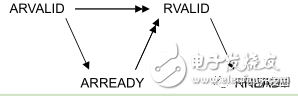
[](http://www.elecfans.com/uploads/allimg/170919/105913IC-6.png)

图10 读传输握手依赖关系

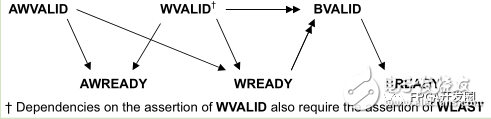
[](http://www.elecfans.com/uploads/allimg/170919/1059135V5-7.png)

图11 写传输握手依赖关系

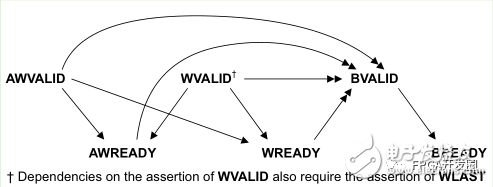
[](http://www.elecfans.com/uploads/allimg/170919/1059135W9-8.png)

图12 从机写响应握手依赖关系

## 3.3传输结构

## 3.3.1 地址结构

AXI协议是基于burst的，主机只给出突发传输的第一个字节的地址，从机必须计算突发传输后续的地址。突发传输不能跨4KB边界（防止突发跨越两个从机的边界，也限制了从机所需支持的地址自增数）。

1）突发长度

ARLEN[7:0]决定读传输的突发长度，AWLEN[7:0]决定写传输的突发长度。AXI3只支持1~16次的突发传输（Burst\_length=AxLEN[3:0]+1），AXI4扩展突发长度支持INCR突发类型为1~256次传输，对于其他的传输类型依然保持1~16次突发传输（Burst\_Length=AxLEN[7:0]+1）。

burst传输具有如下规则：

* wra[pi](http://www.elecfans.com/tags/pi/)ng burst ,burst长度必须是2,4,8,16；
* burst不能跨4KB边界；
* 不支持提前终止burst传输；

所有的组件都不能提前终止一次突发传输。然而，主机可以通过解断言所有的写的strobes来使非所有的写字节来减少写传输的数量。读burst中，主机可以忽略后续的读数据来减少读个数。也就是说，不管怎样，都必须完成所有的burst传输。

注：对于FIFO，忽略后续读数据可能导致数据丢失，必须保证突发传输长度和要求的数据传输大小匹配。

2）突发大小

ARSIZE[2:0]，读突发传输；AWSIZE[2:0]，写突发传输。

AxSIZE[2:0] bytes in transfer：

‘b000 1

‘b001 2

‘b010 4

‘b011 8

‘b100 16

‘b101 32

‘b110 64

‘b111 128

3）突发类型

FIXED：突发传输过程中地址固定，用于FIFO访问；

INCR：增量突发，传输过程中，地址递增。增加量取决AxSIZE的值；

WRAP：回环突发，和增量突发类似，但会在特定高地址的边界处回到低地址处。回环突发的长度只能是2,4,8,16次传输，传输首地址和每次传输的大小对齐。最低的地址整个传输的数据大小对齐。回环边界等于（AxSIZE\*AxLEN）。

AxBURST[1:0] burst type

‘b00 FIXED

‘b01 INCR

‘b10 WRAP

‘b11 Reserved

Start\_Address=AxADDR

Number\_Bytes=2^AxSIZE

Burst\_Length=AxLEN+1

Aligned\_Addr=（INT（Start\_Address/Number\_Bytes))xNumber\_Bytes。//INT表示向下取整。

对于INCR突发和WRAP突发但没有到达回环边界，地址由下述方程决定：

Address\_N=Aligned\_Address+(N-1)xNumber\_Bytes

WRAP突发，突发边界：

Wrap\_Boundary=(INT(Start\_Address/(Number\_Bytes x Burst\_Length)))x(Number\_Bytes x Burst\_Length)

## 3.3.2 数据读写结构

WSTRB[n:0]对应于对应的写字节，WSTRB[n]对应WDATA[8n+7:8n]。WVALID为低时，WSTRB可以为任意值，WVALID为高时，WSTRB为高的字节线必须指示有效的数据。

### 3.3.2.1 窄传输

当主机产生比它数据总线要窄的传输时，由地址和控制信号决定哪个字节被传输：

INCR和WRAP，不同的字节线决定每次burst传输的数据，FIXED，每次传输使用相同的字节线。

下图给出了5次突发传输，起始地址为0，每次传输为8bit，数据总线为32bit，突发类型为INCR。

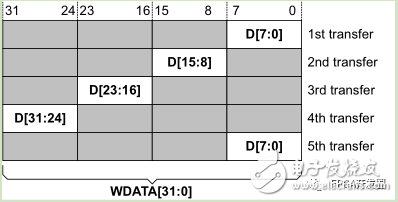
[](http://www.elecfans.com/uploads/allimg/170919/1059136096-9.png)

图13 窄传输示例1

下图给出3次突发，起始地址为4，每次传输32bit，数据总线为64bit。

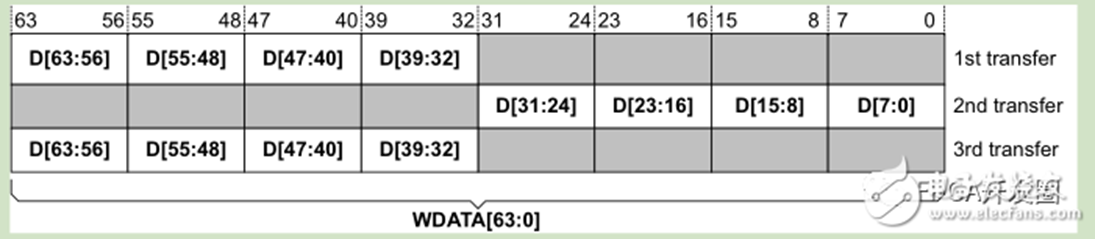


图14 窄传输示例2

### 3.3.2.2 非对齐传输

AXI支持非对齐传输。在大于一个字节的传输中，第一个自己的传输可能是非对齐的。如32-bit数据包起始地址在0x1002，非32bit对齐。

主机可以①使用低位地址线来表示非对齐的起始地址；②提供对齐的起始地址，使用字节线来表示非对齐的起始地址。

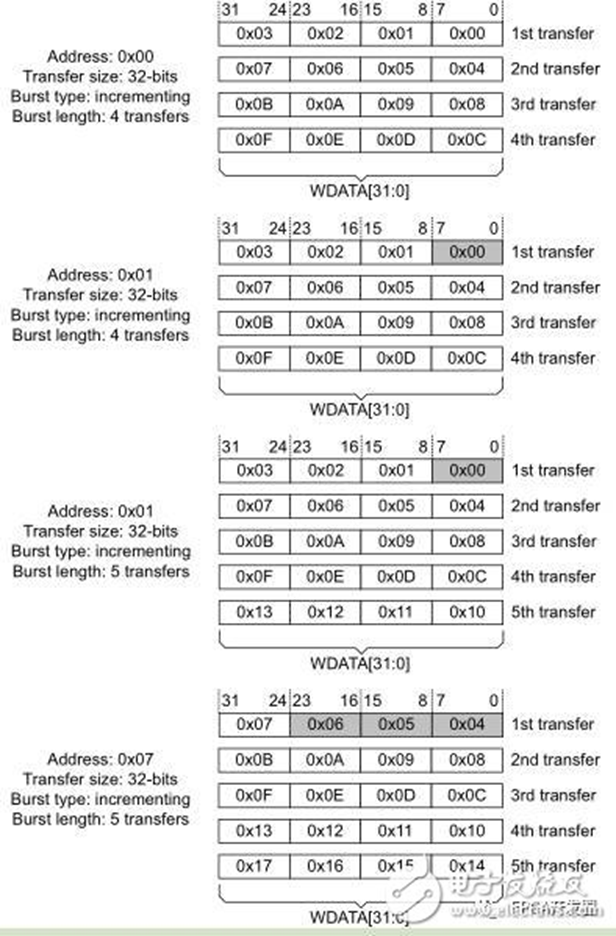


图15 对齐非对齐传输示例1-32bit总线

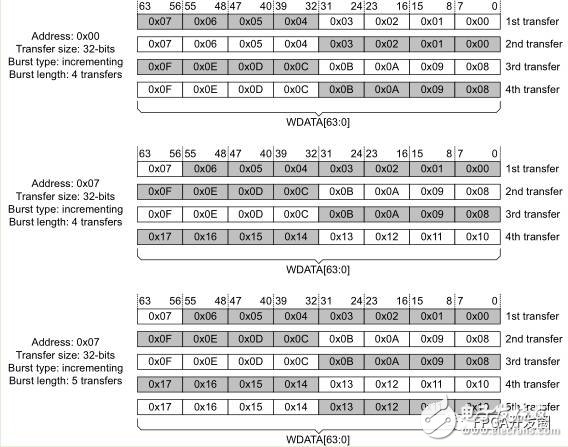


图16 对齐非对齐传输示例2-64bit总线

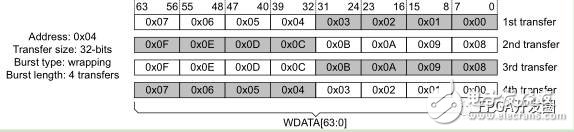
[](http://www.elecfans.com/uploads/allimg/170919/105913HC-13.png)

图17 对齐的回环传输示例

## 3.3.3 读写响应结构

读传输的响应信息是附加在读数据通道上的，写传输的响应在写响应通道。

RRESP[1:0]，读传输；

BRESP[1:0]，写传输；

OKAY(‘b00)：正常访问成功；

EXOKAY(‘b01)：Exclusive 访问成功；

SLVERR(‘b10)：从机错误。表明访问已经成功到了从机，但从机希望返回一个错误的情况给主机；

DECERR(‘b11)：译码错误。一般由互联组件给出，表明没有对应的从机地址。

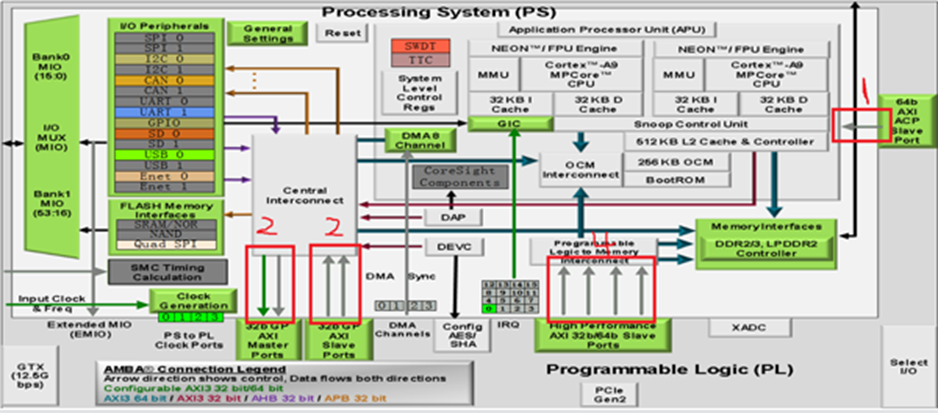
 AXI-stream总线简介-LDD

# 四、AXI与FPGA

Xilinx从6系列的FPGA开始引入的AXI接口协议，主要描述了主设备和从设备之间的数据传输方式。在ZYNQ中继续使用，版本是AXI4，ZYNQ内部设备都有AXI接口。AXI不属于Zynq，也不属于Xilinx，而是属于ARM。它是ARM最新的总线接口，之前叫做AMBA，从3.0以后被称为AXI。

Zynq是以ARM作为核心的，运行时也是第一个“醒”过来，然后找可执行代码，找到后进入FSBL（第一引导阶段），接着找配置逻辑部分的bit文件，找到后就叫醒PL按照bit中的方式运行，再接着找可执行代码，进入SSBL（第二引导阶段），这时就可以初始化操作系统的运行环境，引导像Linux这样的大型程序，随后将控制权交给Linux。Linux运行时可以跟PL进行数据交互。注意，这时数据交互的通路，就是AXI总线。AXI就是负责ARM与FPGA之间通信的专用数据通道。

ARM内部用硬件实现了AXI总线协议，包括9个物理接口，主要用于PS与PL的互联，分别为AXI-GP0~AXI-GP3、AXI-HP0~AXI-HP3和AXI-ACP接口。



可以看到，只有两个AXI-GP是Master Port，即主机接口，其余7个口都是Slave Port（从机接口）。主机接口具有发起读写的权限，ARM可以利用两个AXI-GP主机接口主动访问PL逻辑，其实就是把PL映射到某个地址，读写PL寄存器如同在读写自己的存储器。其余从机接口就属于被动接口，接受来自PL的读写。AXI\_ACP接口，是ARM多核架构下定义的加速器一致性端口，用来管理DMA之类不带缓存的AXI外设，PS端是Slave接口。AXI\_HP接口，是高性能、高带宽的AXI3.0标准的接口，总共有四个，PL模块作为主设备连接，主要用于PL访问PS上的存储器（DDR和On-Chip RAM）。

这9个AXI接口性能也是不同的。GP接口是32位的低性能接口，理论带宽600MB/s，而HP和ACP接口为64位高性能接口，理论带宽1200MB/s。

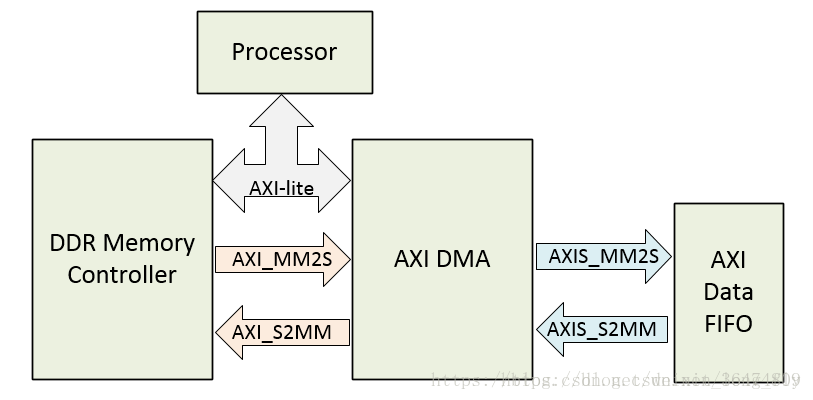
有人会问，为什么高性能接口不做成主机接口呢？这样可以由ARM发起高速数据传输。答案是高性能接口根本不需要ARM CPU来负责数据搬移，真正的搬运工是位于PL中的DMA控制器。

位于PS端的ARM直接有硬件支持AXI接口，而PL则需要使用逻辑实现相应的AXI协议。Xilinx提供现成IP如AXI-DMA、AXI-GPIO、AXI-Datamover都实现了相应的接口，使用时直接从XPS的IP列表中添加即可实现相应的功能。

有时，用户需要开发自己定义的IP同PS进行通信，这时可以利用XPS向导生成对应的IP。XPS中用户自定义IP核可以拥有AXI-Lite、AXI4、AXI-Stream、PLB和FSL这些接口。后两种由于ARM这一端不支持，所以不用。

Xilinx提供的流式IP核有很多用途，可以实现音频流、视频流、数据流到内存或者相反方向的传输。有人问了，内存是PS控制的，怎么才能把PS里DDR2的内容以Stream形式发出去呢（例如以固定速度送往D3A，完成信号发生器的设计）？答案就是利用AXI总线做转换。ZYNQ的PS部分是ARM CortexA9系列，支持AXI4、AXI-Lite总线。PL部分也有相应AXI总线接口，这样就能完成PS到PL的互联。仅仅这样还不够，需要PL部分实现流式转换，即AXI-Stream接口实现。Xilinx提供的从AXI到AXI-Stream转换的IP核有：AXI-DMA，AXI-Datamover，AXI-FIFO-MM2S以及AXI-VDMA等。这些IP核可以在XPS的IP Catalog窗口中看到。

AXI-DMA：实现从PS内存到PL高速传输高速通道AXI-HP到AXI-Stream的转换；



AXI-FIFO-MM2S：实现从PS内存到PL通用传输通道AXI-GP到AXI-Stream的转换；

AXI-Datamover：实现从PS内存到PL高速传输高速通道AXI-HP到AXI-Stream的转换，只不过这次是完全由PL控制的，PS是完全被动的；

AXI-VDMA：实现从PS内存到PL高速传输高速通道AXI-HP到AXI-Stream的转换，只不过是专门针对视频、图像等二维数据的。

除了上面的还有一个AXI-CDMAIP核，这个是由PL完成的将数据从内存的一个位置搬移到另一个位置，无需CPU来插手。

这里要和大家说明白一点，就是AXI总线和接口的区别。总线是一种标准化接口，由数据线、地址线、控制线等构成，具有一定的强制性。接口是其物理实现，即在硬件上的分配。在ZYNQ中，支持AXI-Lite、AXI4和AXI-Stream三种总线，但PS与PL之间的接口却只支持前两种，AXI-Stream只能在PL中实现，不能直接和PS相连，必须通过AXI-Lite或AXI4转接。PS与PL之间的物理接口有9个，包括4个AXI-GP接口和4个AXI-HP接口、1个AXI-ACP接口，均为内存映射型AXI接口。

上面的IP是完成总线协议转换，如果需要做某些处理（如变换、迭代、训练……），则需要生成一个自定义Stream类型IP，与上面的Stream接口连接起来，实现数据输入输出。用户的功能在自定义Stream类型IP中实现。