# AXI 协议调研报告

## 目录


一、AXI 协议简介 ..... 2

1.1 什么是 AXI ..... 2
1.2 AXI 的特点 ..... 2
1.3 AXI 的性能 ..... 3
1.4 AXI 的架构 ..... 3
1.5 AXI4 协议 ..... 5
1.5.1 AXI4 接口类型 ..... 5
1.5.2 AXI4 协议优势 ..... 6
1.5.3 AXI4 的工作模式 ..... 7
1.5.3.1 握手机制 ..... 7
1.5.3.2 AXI 工作模式 ..... 8
1.5.3.3 AXIS 工作模式 ..... 9

二、AXI 协议信号描述 ..... 9

2.1 信号描述 ..... 9
2.2 信号接口要求 ..... 11
2.2.1 时钟复位 ..... 11
2.2.2 通道信号 ..... 12
2.2.2.1 通道间关系 ..... 12
2.2.2.2 通道握手信号的依赖关系 ..... 12
3.3 传输结构 ..... 13
3.3.1 地址结构 ..... 13
3.3.2 数据读写结构 ..... 15
3.3.2.1 窄传输 ..... 15
3.3.2.2 非对齐传输 ..... 15
3.3.3 读写响应结构 ..... 17

四、AXI 与 FPGA ..... 18

# 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。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。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。

表 1　AXI 与 AHB 的性能对比

| 总线名称 | AMBA 3 AXI | AMBA 2 AHB |
|---|---|---|
| 数据线宽度（位） | 8,16,32,64,128,256,512,1024 | 32, 64, 128, 256 |
| 地址线宽度（位） | 32 | 32 |
| 体系结构 | 多主/从设备<br>仲裁机制 | 多主/从设备<br>仲裁机制 |
| 数据线协议 | 支持流水/分裂传输<br>支持猝发传输<br>支持乱序访问<br>字节/半字/字 | 支持流水/分裂传输<br>支持猝发传输<br>字节/半字/字 |
| 数据对齐方式 | 大端/小端对齐<br>支持非对齐操作 | 大端/小端对齐<br>不支持非对齐操作 |
| 时序 | 同步 | 同步 |
| 互接 | 多路 | 多路 |
| 支持互接 | 不支持三态总线<br>分开的读/写数据线 | 不支持三态总线<br>分开的读/写数据线 |



## 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。

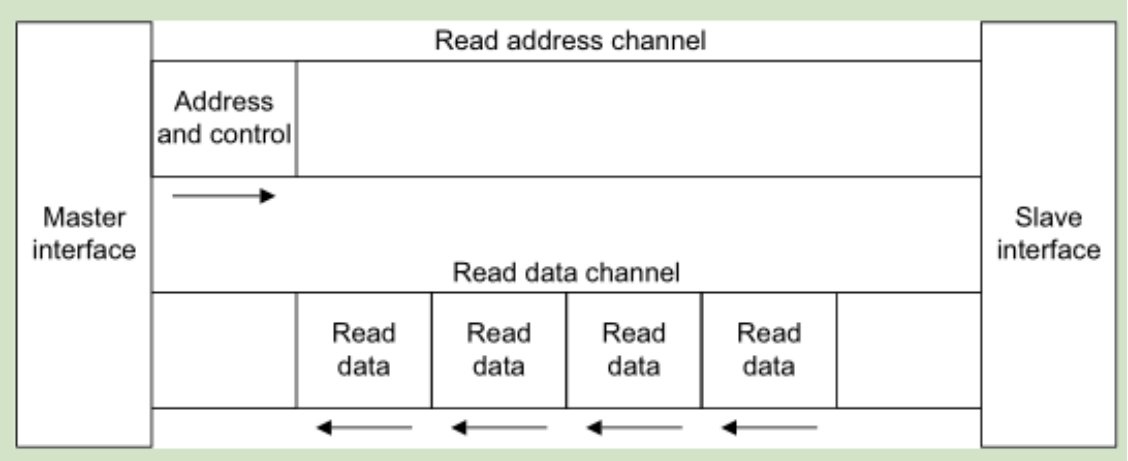



图 1 AXI 读架构

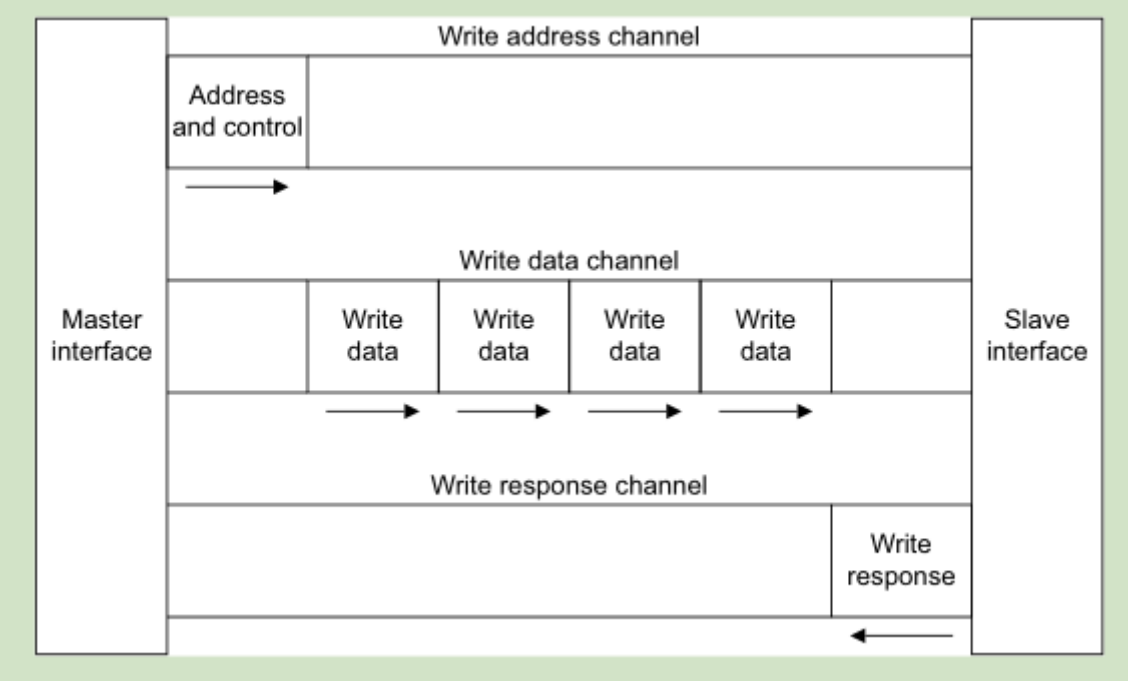



图 2 AXI 写架构

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。

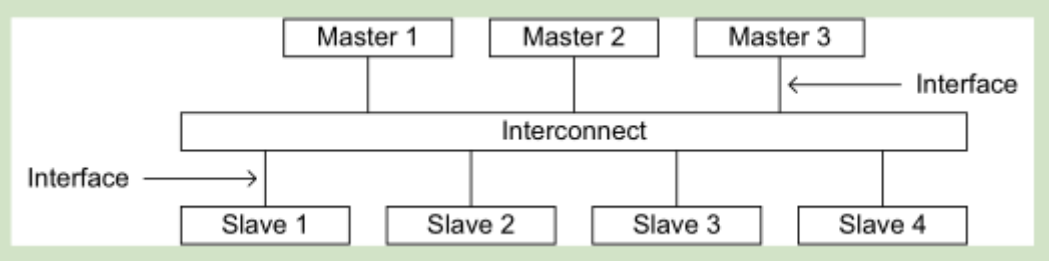



图 3 AXI 接口与互联

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
- multilayer 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 | AXI4-Lite | AXI4-Stream |
|---|---|---|---|
| Dedicated for | high-performance and memory mapped systems | register-style interfaces (area efficient implementation) | non-address based IP |
| Burst** (data beat) | up to 256 | 1 | unlimited |
| Data Width | 32 to 1024 bits | 32 or 64 bits | any number of bytes |
| Applications (examples) | Embedded, memory | Small footprint control logic | DSP, video, communication |



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：

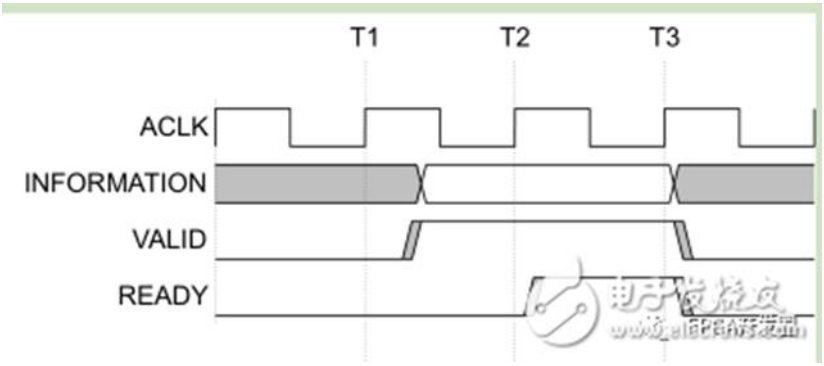



图 4 VALID before READY 握手

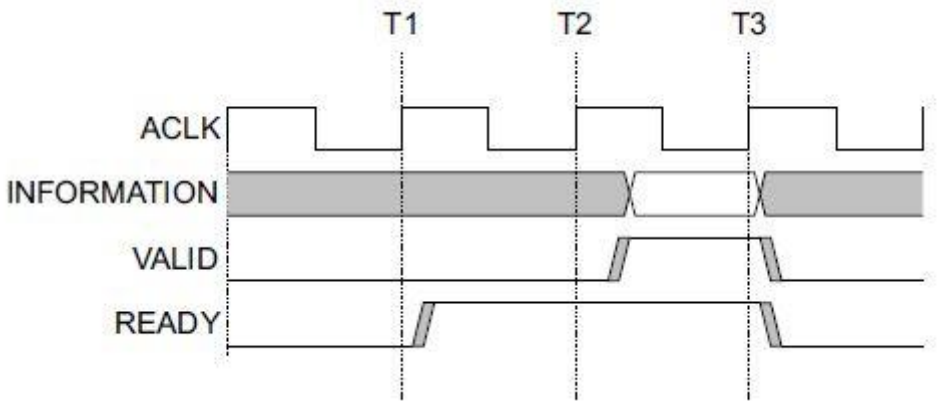



图 5 READY before VALID 握手

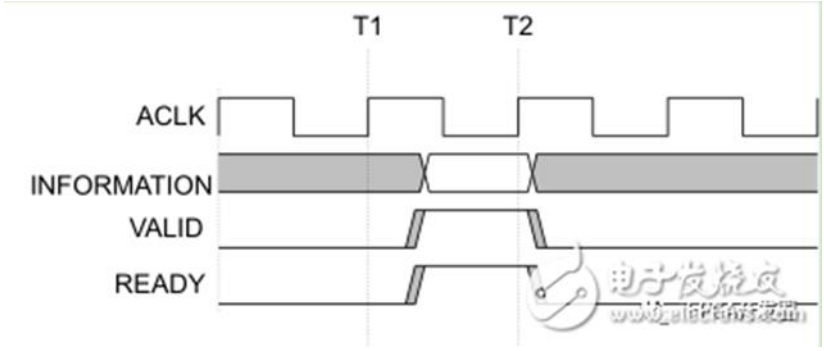



图 6 VALID with READY 握手

#### 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：

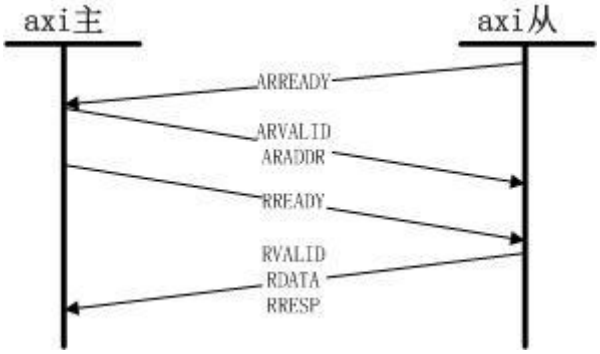



图 7 AXI 读操作

**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：

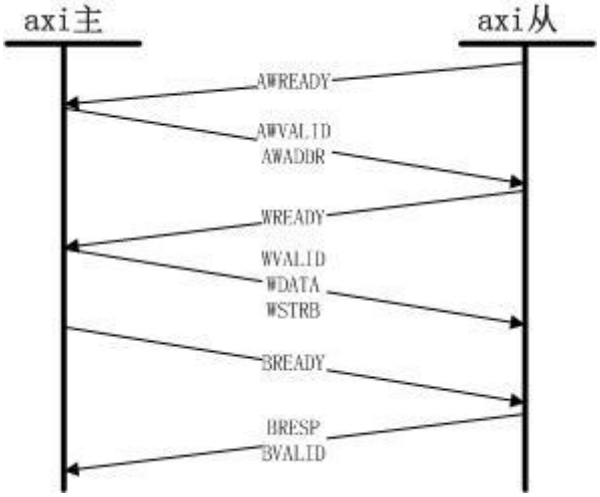



图 8 AXI 写操作

#### 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：

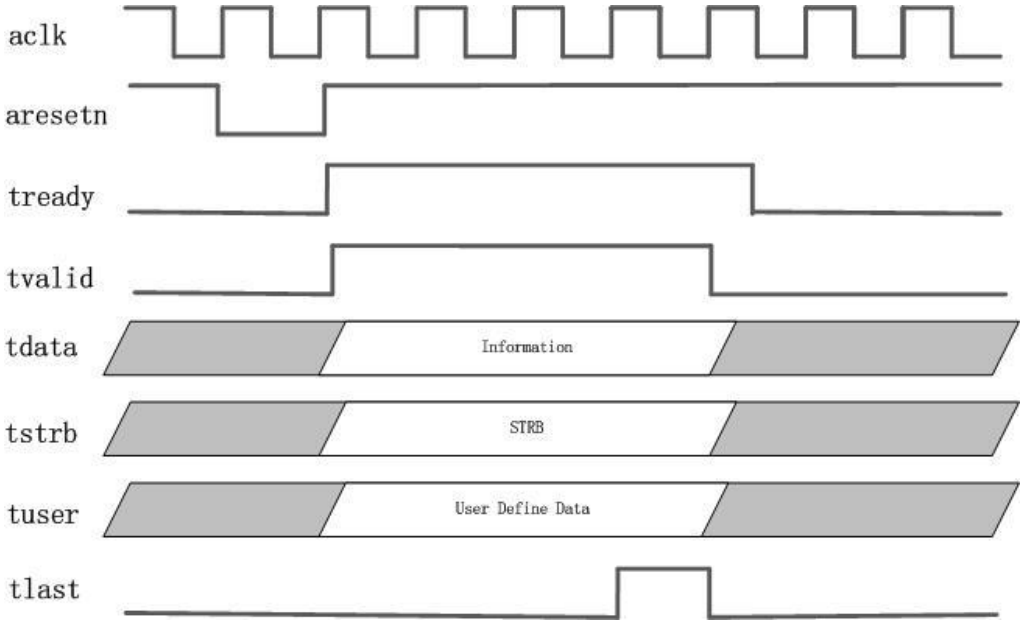



图 9 AXIS 通信时序

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。

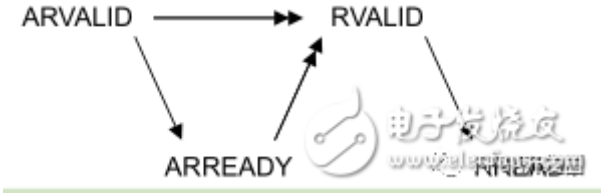



图 10 读传输握手依赖关系

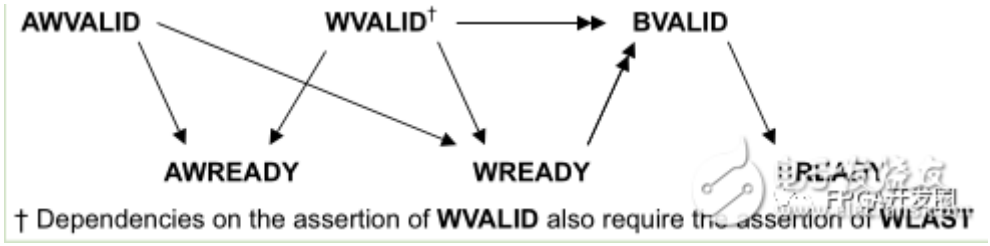



图 11 写传输握手依赖关系

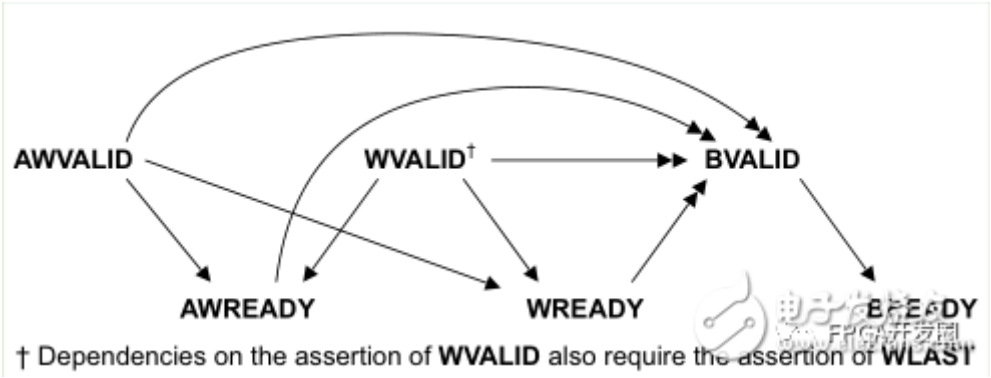



图 12 从机写响应握手依赖关系

## 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：

- wraping 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。

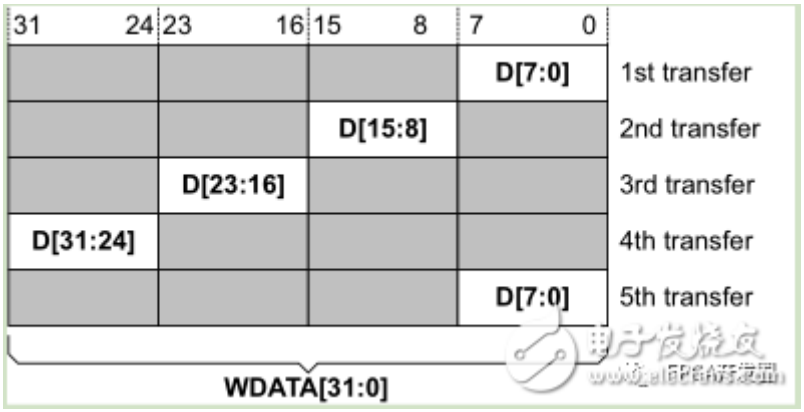



图 13 窄传输示例 1

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。

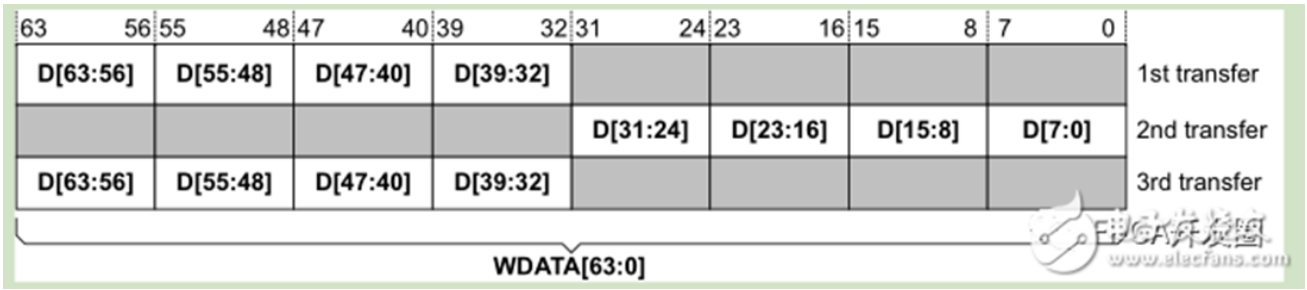



图 14 窄传输示例 2

#### 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。

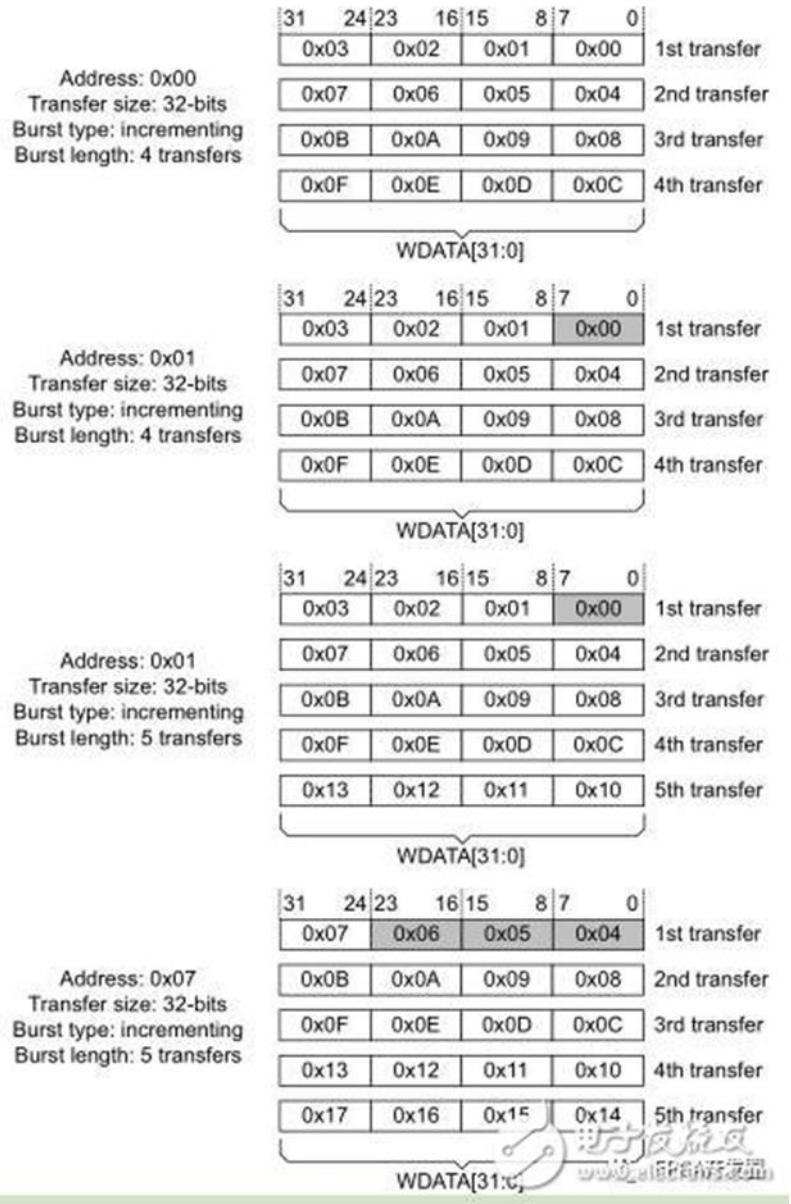



图 15 对齐非对齐传输示例 1-32bit 总线

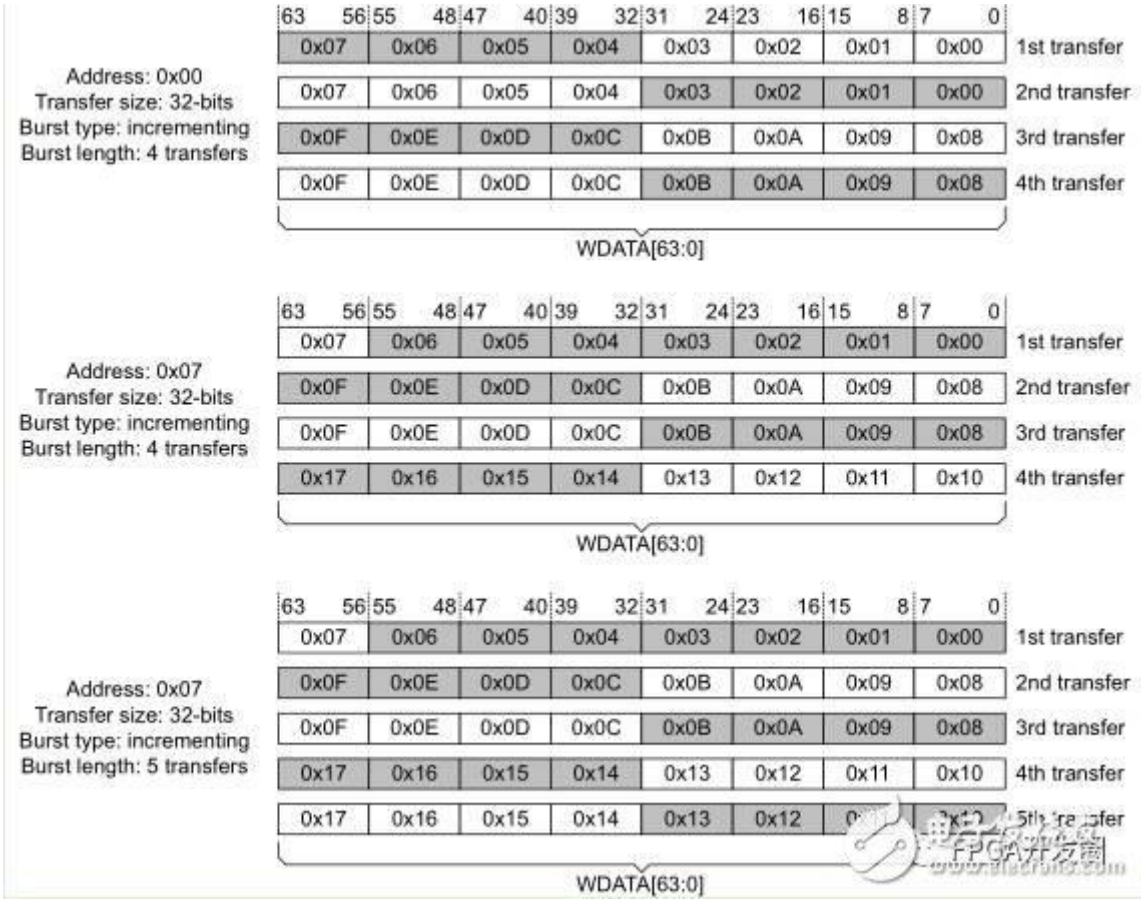



图 16 对齐非对齐传输示例 2-64bit 总线

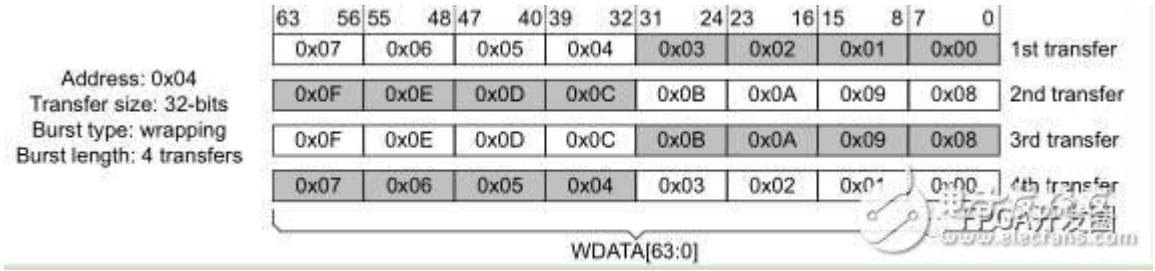



图 17 对齐的回环传输示例

### 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；

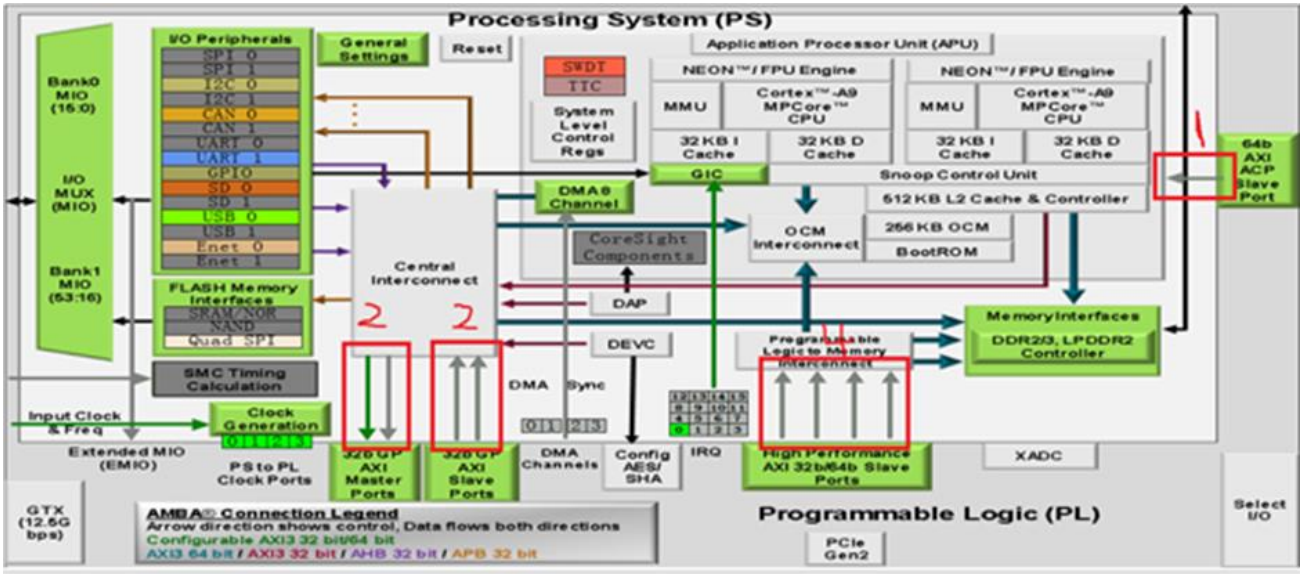
DECERR(‘b11)：译码错误。一般由互联组件给出，表明没有对应的从机地址。

# 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。

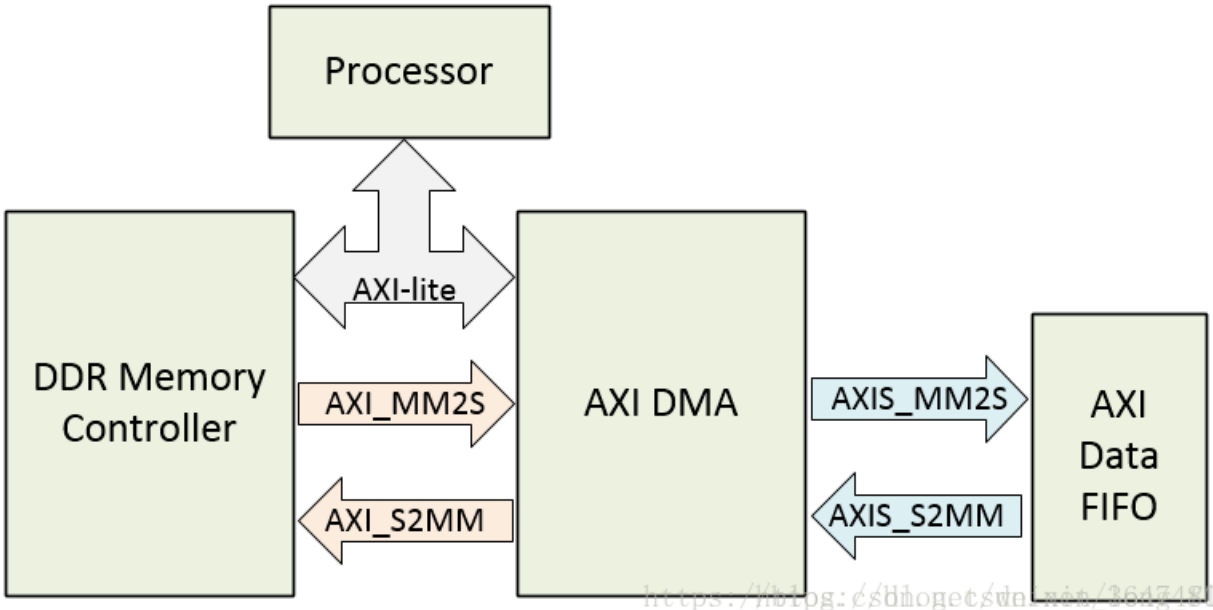
有人会问，为什么高性能接口不做成主机接口呢？这样可以由 ARM 发起高速数据传输。答案是高性能接口根本不需要 ARM CPU 来负责数据搬移，真正的搬运工是位于 PL 中的 DMA 控制器。

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。