Wishbone 总线

Wishbone 总线规范是一种片上系统 IP 核互连体系结构。它定义了一种 IP 核之间公共的逻辑接口，减轻了系统组件集成的难度，提高了系统组件的可重用性、可靠性和可移植性。

Wishbone 总线规范可用于软核、固核和硬核，对开发工具和目标硬件没有特殊要求，可以用多种硬件描述语言来实现。

Wishbone 总线规范的目的是作为一种 IP 核之间的通用接口，因此它定义了一套标准的信号和总线周期，以连接不同的模块，而不是试图去规范 IP 核的功能和接口。

Wishbone 总线结构十分简单，它仅仅定义了一条高速总线。在一个复杂的系统中，可以采用两条 Wishbone 总线的多级总线结构：其一用于高性能系统部分，其二用于低速外设部分，两者之间需要一个接口。这个接口虽然占用一些电路资源，但这比设计并连接两种不同的总线要简单多了。用户可以按需要自定义 Wishbone 标准，如字节对齐方式和标志位(TAG)的含义等等，还可以加上一些其它的特性。

灵活性是 Wishbone 总线的另一个优点。由于 IP 核种类多样，其间并没有一种统一的间接方式。为满足不同系统的需要，Wishbone 总线提供了四种不同的 IP 核互连方式：

- 点到点(Pointto Point)　　(图 1-1)，用于两 IP 核直接互连；
- 数据流(Data Flow)　　(图 1-2)，用于多个串行 IP 核之间的数据并发传输；
- 共享总线(Shared Bus)　　(图 1-3)，多个 IP 核共享一条总线；
- 交叉开关(Crossbar switch)(图 1-4)，同时连接多个主从部件，提高系统吞吐量。

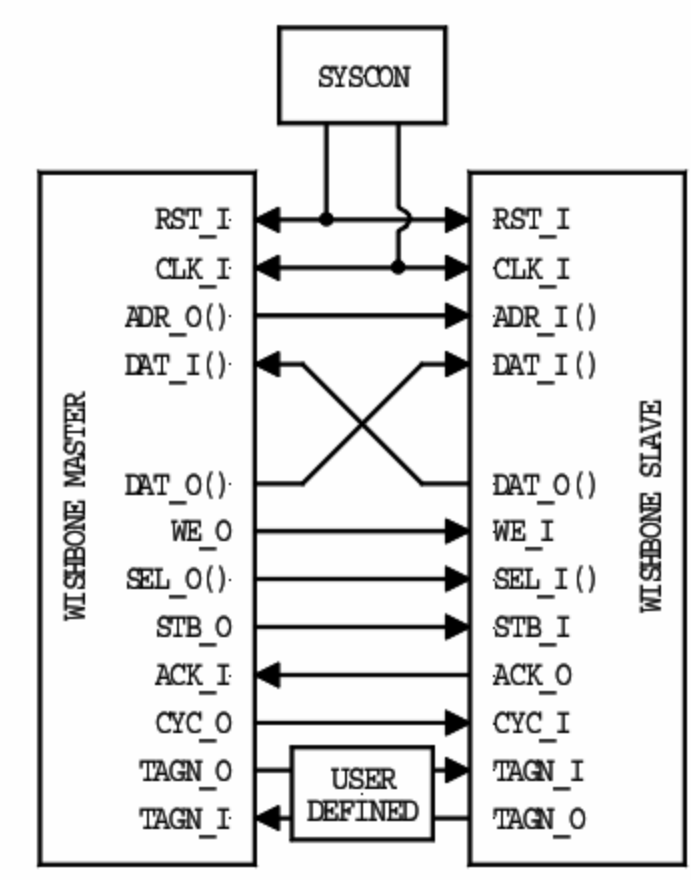


图 1-1　　Wishbone 的 Point to Point 互连结构

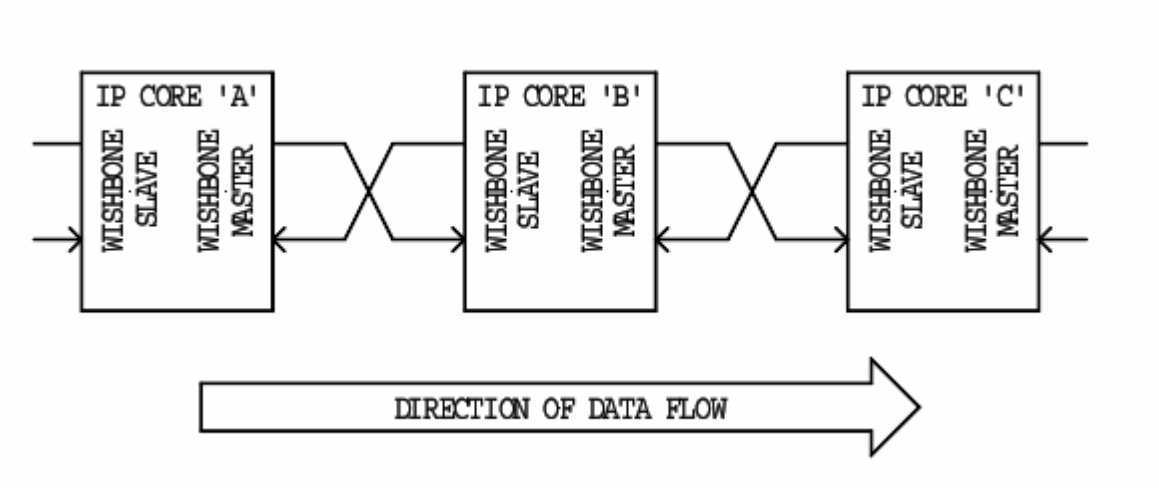


图 1-2　Wishbone 的 Data Flow 互连结构

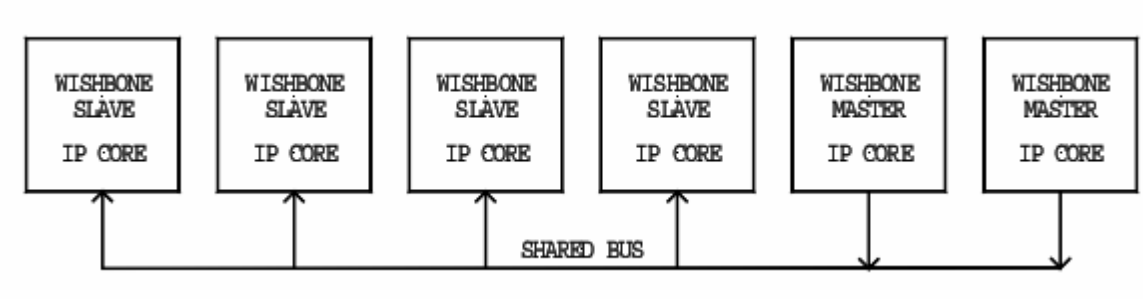


图 1-3　Wishbone 的 Share Bus 互连结构

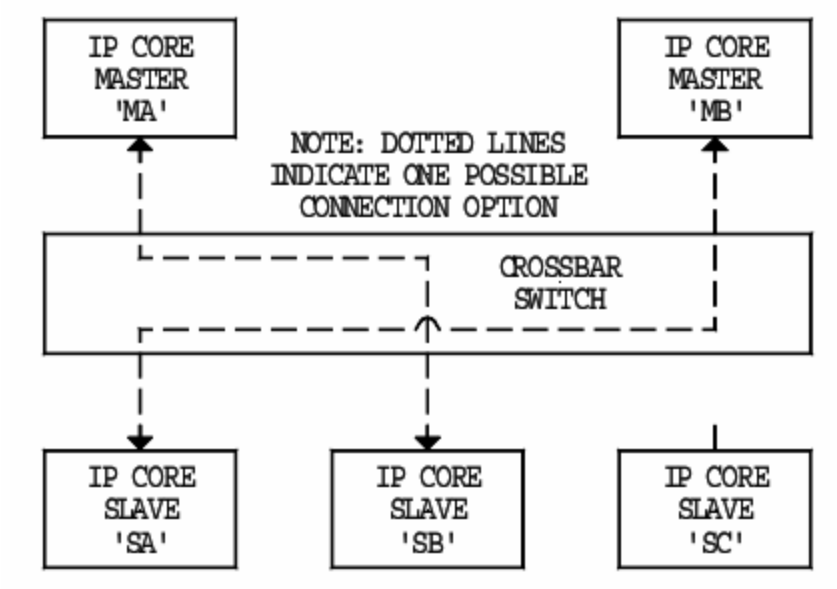


图 1-4　Wishbone 的 Crossbar 互连结构

还有一种片外连接方式，可以连接到上面任何一种互连网络中。比如说，两个有 Wishbone 接口的不同芯片之间就可以用点到点方式进行连接，如图 1-5 所示。

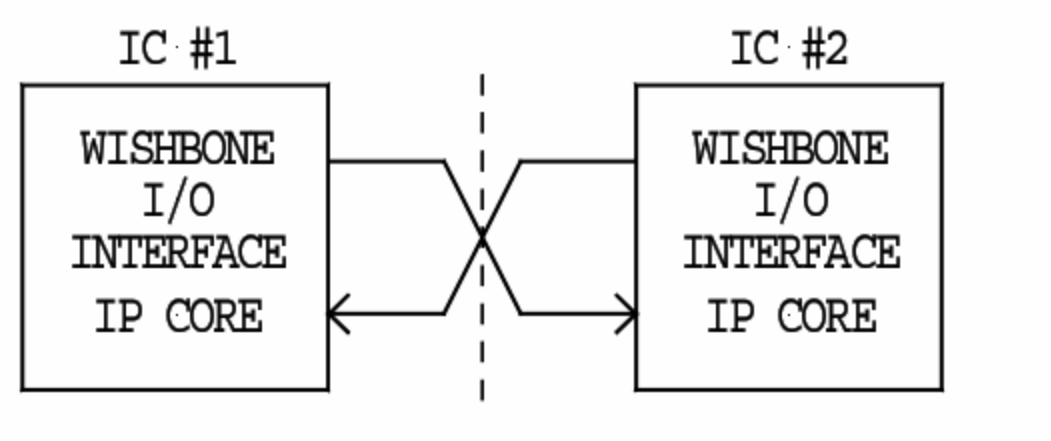


图 1-5　　Wishbone 的 Off-chip 互连结构

Wishbone 总线主要特征如下：

- 所有应用适用于同一种总线体系结构；
- 是一种简单、紧凑的逻辑 IP 核硬件接口，只需很少的逻辑单元即可实现；
- 时序非常简单；
- 主 / 从结构的总线，支持多个总线主设备；
- 8～64 位数据总线(可扩充)；
- 单周期读写；
- 支持所有常用的总线数据传输协议，如单字节读写周期、块传输周期、控制操作及其它的总线事务等；
- 支持多种 IP 核互连网络，如单向总线、双向总线、基于多路互用的互连网络、基于三态的互连网络等；
- 支持总线周期的正常结束、重试结束和错误结束；
- 使用用户白定义标记(TAG)，确定数据传输类型、中断向量等；
- 仲裁器机制由用户白定义；

Wishbone 总线的信号和时序

Wishbone 的信号

Wishbone 总线的信号分为 4 类：系统控制信号、主从共有信号、主设备信号和从设备信号。

(1)系统控制信号包括：

- CLK_O 系统时钟输出[CLK_O]，由 SYSCON 模块(系统控制模块)产生，它同步 Wishbone 互连的所有内部信号的有效。INTERCON 模块(主从设备连接模块)连接[CLK_O]和主从设备的[CLK_I]信号。
- RST_O 复位输出[RST_O]，由 SYSCON 模块产生，它强制所有的 Wishbone 接口重新启动。所有的内部自启动状态机将被强制处于初始状态。INTERCON 模块(主从设备连接模块)连接[RST_O]和主从设备的[RST_I]信号。

(2)主从共有信号包括：

- CLK_I 时钟输入[CLK_I]，同步所有的 Wishbone 互连的所有内部信号的有效　Wishbone 输出信号在[CLK_I]的上升沿寄存，在[CLK_I]的上升沿以前处于稳定状态
- DAT_I 数据输入总线[DAT_I()]，被用来传递二进制数据。总线的边界由端口大小决定，端口的最大值是 64 位 （ [DAT_I(63.. 0)] ）。
- DAT_O 数据输出总线[DAT_O()]，被用来传递二进制数据。总线的边界由端口大小决定，端口的最大值是 64 位([DAT O(63.. 0)])。
- RST_I 复位输入，强制 Wishbone 接口重新启动。此外，所有的内部白启动状态机将被强制处于初始状态。这个信号只复位 Wishbone 接口，不需要复位 IP Core 的其他部分。
- TGD_I() 数据标记类型[TGD_I()]，用在主设备和从设备的接口上。它包含与数据输入总线[DAT_I()]有关的信息，并且由[STB_I]信号决定有效性。比如，奇偶保护、错误修正和时序标记信息可以附加在数据总线上。由于新信号的时序已经预先定义，这些标记

位简化了新信号的定义工作。在接口的数据手册中必须定义数据标记的名称和操作。

- TGD_O() 数据标记类型[TGD_O()]，用在主设备和从设备的接口上。它包含与数据输出总线[DAT_O()]有关的信息，并且由[STB_I]信号决定有效性。比如，奇偶保护、错误修正和时序标记信息可以附加在数据总线上。由于新信号的时序已经预先定义 这些标记位简化了新信号的定义。在接口的数据手册中必须定义数据标记的名称和操作。

(3)主设备信号包括：

- ACK_I 应答输入[ACK_I]，当它有效的时候，表明一个总线循环的正常结束。
- ADR_O()地址输出总线[ADR_O()]用于传递二进制地址。总线的高端边界由 IPCore 的地址总线的宽度决定，总线的低端边界由数据端口的宽度和粒度决定。比如一个 32 位的数据端口的字节粒度是[ADR_O(n.. 2)]对于有些情况(比如 FIFO 接口)在接口中没有这个总线。
- CYC_O 循环输出[CYC_O]，当它有效的时候，表明正在进行一个正确的总线循环。这个信号在所有总线循环的持续过程中保持有效。比如在一个块传送过程中会有多次数据传送。[CYC O]在多端口接口(比如双口存储器的接口)中很有用。在这种情况下，[CYC_O]信号需要使用仲裁器提供的共有总线。
- ERR_I 错误输入[ERR_I]，指示一次错误的总线循环的结束。错误源和主设备的反应由 IP Core 的提供方定义。
- LOCK_O 锁定输出[LOCK_O]，当它有效的时候，表明当前的总线循环不能被打断。锁定信号用于声明需要获得对总线完全拥有权。一旦传送开始， INTERCON 模块不会将总线的控制权交给其他主设备，直到当前主设备放弃[LOCK_O]或者[CYC_O]。
- RTY_I 重试输入[RTY_I]，指示接口没有准备好接受或者发送数据，并且循环应当重试。什么时候和如何进行重试由 IP Core 的提供方定义。
- SEL_O()选择输出总线[SEL_O()]，指示在 READ 循环的时候有效数据在 DAT_I()总线中的位置，和在 WRITE 循环的时候有效数据在[DAT_O()]总线中的位置。选择输出总线的边界由端口的粒度决定。比如，如果在一个 64 位的端口中使用 8 位的粒度，那么选择输出总线将有 8 个信号，边界是[SEL O(7.. O)]。每个单独的选择信号确定 64 位数据总线的 8 个有效字节中的一个。
- STB O 选通输出[STB_O]，指示一个有效的数据传输循环。它用于确定像[SEL_O()]这样的接口上的其他信号的有效性。对于每个[STB_O]信号的声明，从设备需要声明[ACK_I]、[ERR_I] 或者[RTY_I]信号。
- TGA_O()地址标记类型[TGA_O()]，包含与地址线[ADR_O()]有关的信息，并且由[STB_O]信号确定有效性。比如，地址尺寸(24 位、32 位)和存储器管理(保护与非保护)信息可以附加在某一个地址上。由于新信号的时序已经预先定义，这些标记位简化了新信号的定义工作。在接口的数据手册中必须定义数据标记的名称和操作。
- TGC_O()循环标记类型[TGC_O()]，包含与总线循环有关的信息，并且由[CYC_O]信号决定有效性。比如，数据传输、中断响应和缓存控制循环由循环标记来唯一确定。它们也可以用来区别 SINGLE，BLOCK 和 RMW 循环。由于新信号的时序已经预先定义，这些标记位简化了新信号的定义。在接口的数据手册中必须定义数据标记的名称和操作。
- WE_O 写使能输出[WE_O]，指示当前的本地总线循环是有个 READ 还是 WRITE 循环。这个信号在 READ 循环中是无效的，在 WRITE 循环中是有效的。

(4)从设备信号包括：

- ACK_O 应答输出[ACK_O]，当它有效的时候，表明一个总线循环的正常结束。
- ADR_I()地址输入总线[ADR_I()]，用于传递二进制地址。总线的高端边界由 IP core 的地址总线的宽度决定，总线的低端边界由数据端口的宽度和粒度决定。比如，一个 32 位的数据端口的字节粒度是[ADR O(n.. 2)]。对于有些情况(比如 FIFO 接口)，在接口中没有这个总线。
- CYC_I 循环输入[CYC_I]，当它有效的时候，表明正在进行一个正确的总线循环。这个信号在所有总线循环的持续过程中保持有效。比如，在一个块传送过程中会有多次数据传送。[CYC_I]信号在第一次数据传输的时候有效，并且一直保持到最后一个数据传输。
- ERR_O 错误输出[ERR_O]，指示一次错误的总线循环的结束。错误源和主设备的反应由 IP Core 的提供方定义。

。LOCK_I 锁定输入[LOCK_I]，当它有效的时候，表明当前的总线循环不能被打断。一个收[LOCK_I]信号的从设备只能被一个主设备访问，直到[LOCK_I]或者[CYC_I]无效。

- RTY_O 重试输出[RTY_O]，指示接口没有准备好接受或者发送数据，并且循环应当重试。什么时候和如何进行重试由 IP Core 的提供方定义。
- SEL_I()选择输入总线[SEL_I()]，指示在 WRITE 循环的时候有效数据在 DAT_I()总线中的位置，和在 READ 循环的时候有效数据在[DAT_O()]总线中的位置。选择输出总线的边界由端口的粒度决定。比如，如果在一个 64 位的端口中使用 8 位的粒度，那么选择输出总线将有 8 个信号，边界是[SEL_I(7.. 0)]。每个单独的选择信号确定 64 位数据总线的 8 个有效字节中的一个。
- STB_I 选通输入[STB_I]，当它有效的时候，指示这个从设备被选中。当[STB_I]有效的时候，一个从设备应当只对其他的 Wishbone 信号做出反应。对于每个[STB_I]信号的声明，从设备需要声明[ACK_O]、[ERR_O]或者[RTY_O]信号。
- TGA_I 地址标记类型[TGA_I()]，包含与地址线[ADR_I()]有关的信息，并且由[STB_ O]信号确定有效性。比如，地址尺寸(24 位、32 位)和存储器管理(保护与非保护)信息可以附加在某一个地址上。由于新信号的时序已经预先定义，这些标记位简化了新信号的定义工作。在接口的数据手册中必须定义数据标记的名称和操作。
- TGC_I()循环标记类型[TGD_I()]，包含与总线循环有关的信息，并且由[CYC_I]信号决定有效性。比如，数据传输、中断响应和缓存控制循环由循环标记来唯一确定。它们也可以用来区别 SINGLE，BLOCK 和 RMW 循环。由于新信号的时序已经预先定义，这些标记位简化了新信号的定义。在接口的数据手册中必须定义数据标记的名称和操作。
- WE_I 写使能输入[WE_I]，指示当前的本地总线循环是有个 READ 还是 WRITE 循环。这个信号在 READ 循环中是无效的，在 WRITE 循环中是有效的。

Wishbone 典型的总线循环

典型的 Wishbone 的总线循环有 5 中：SINGLE READ、SINGLE WRITE、BLOACK READ、BLOACKWRITE 和 RMW。

SINGLEREAD 循环流程如下：

- 时钟沿 O：

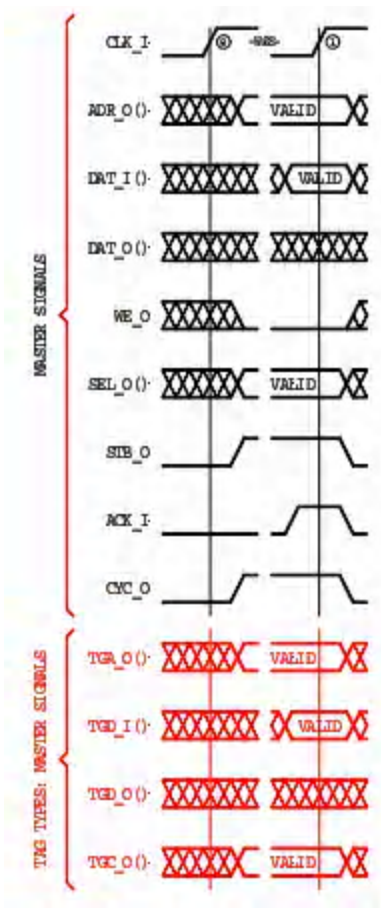
◆主设备在[ADR_O()]和[TGA_O()]上提供一个有效的地址。
◆主设备无效[WE_O]以指示一个 READ 循环。
◆主设备提供选择信号[SEL_O()]以指示数据的位置。
◆主设备声明[CYC_O]和[TGC_O()]以指示循环的开始。
◆主设备声明[STB_O]以指示一次传输的开始。

- 准备沿 I：

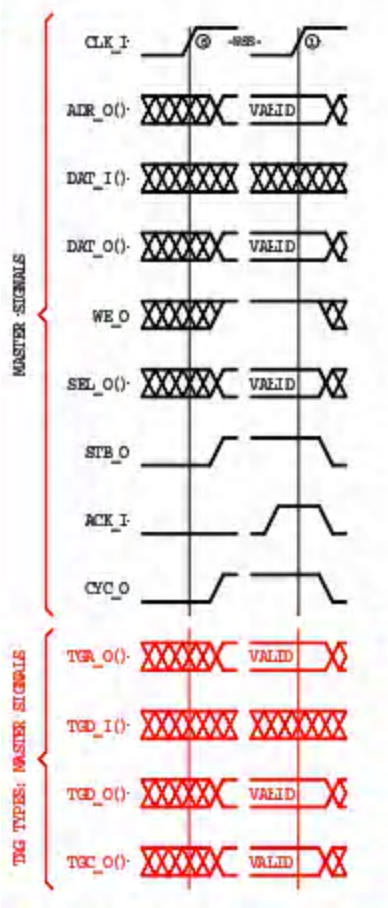
◆从设备译码输入并相应的声明[ACK_I]。
◆从设备在[DAT_I()]和[TGD_I()]提供有效的数据。
◆从设备根据[STB_O]声明[ACK_I]以指示数据有效。
◆主设备监视[ACK_I]并且准备锁存[DAT_I()]和[TGD_I()]上的数据。

- 时钟沿 I：

◆主设备锁存[DAT_I()和[TGD_I()]上的数据。
◆主设备无效[STB_O]和[CYC_O]以指示循环的结束。
◆从设备根据[STB_O]的无效而无效[ACK_I]。



SINGLEREAD 循环 图 1-6



SINGLEWRITE 图 1-7

SINGLEWRITE 循环流程如下：

• 时钟沿 0：

◆主设备在[ADR_O()]和[TGA_O()]上提供一个有效的地址。

◆主设备在[DAT_O()]和[TGD_O()]上提供有效的数据。

◆主设备声明[WE_O]以指示一个 WRITE 循环。

◆主设备提供选择信号[SEL_O()]以指示数据的位置。

◆主设备声明[CYC_O]和[TGC_O()]以指示循环的开始。

◆主设备声明[STB_O]以指示一次传输的开始。

• 准备沿 I：

◆从设备译码输入并且响应的声明[ACK_I]。

◆从设备准备锁存[DAT_O()]和[TGD_O()]上的数据。

◆从设备根据[STB_O]声明[ACK_I]以指示主设备锁存数据。

◆主设备监视[ACK_I]并且准备结束循环。

• 时钟沿 I：

◆从设备锁存[DAT_O()]和[TGD_O()]上的数据。

◆主设备无效[STB_O]和[CYC_O]以指示循环结束。

◆从设备根据[STB_O]的无效而无效[ACK_I]。

BLOACKREAD 循环流程如下：

• 时钟沿 0：

◆主设备在[ADR_O()]和[TGA_O()]提供一个有效的地址。

◆主设备无效[WE_O]以指示一个 READ 循环。

◆主设备提供选择信号[SEL_O()]以指示数据的位置。

◆主设备声明[CYC_O]和[TGC_O()]以指示循环开始。

◆主设备声明[STB_O]以指示一次传输的开始。

• 准备沿 1：

◆从设备译码输入并且声明[ACK_I]。
◆从设备在[DAT_I()]和[TGD_I()]上提供有效的数据。
◆主设备监视[ACK_I]并且准备锁存[DAT_I()]和[TGD_I()]。
・时钟沿 I：
◆主设备[DAT_I()]和[TGD_I()]上的锁存数据。
◆主设备提供新的[ADR_O()]和[TGA_O()]。
◆主设备提供新的选择信号[SEL_O()]以指示数据的位置。
・准备沿 2：
◆从设备译码输入并相应的声明[ACK_I]。
◆从设备在[DAT_I()和[TGD_I（）]上提供有效的数据。
◆主设备监视[ACK_I]并且准备锁存[DAT_I（）]和[TGD_I()]。
・时钟沿 2：
◆主设备锁存[DAT_I()]和[TGD_I()]上的数据。
◆主设备无效[STB_O]以加入一个等待状态(WSM)。
・准备沿 3：
◆从根据[STB_O]设备无效[ACK_I]。
・时钟沿 3：
◆主设备提供新的[ADR_O()]和[TGA_O()]。
◆主设备提供新的选择信号[SEL_O()]以指示数据的位置。
◆主设备声明『STB_O]。
・准备沿 4：
◆从设备译码输入并相应的声明[ACK_I]。
◆从设备在[DAT_I()]和[TGD_I()]上提供有效的数据。
◆主设备监视[ACK_I]并且准备锁存[DAT_I()]和[TGD_I()]。
・时钟沿 4：
◆主设备锁存[DAT_I()]和[TGD_I()]上的数据。
◆主设备提供[ADR_O()]和[TGA_O()]。
◆主设备提供新的选择信号[SEL_O()]以指示数据的位置。
・准备沿 5：
◆从设备译码输入并相应的声明[ACK_I]。
◆从设备在[DAT_I()]和[TGD_I()]上提供有效的数据。
◆主设备监视[ACK_I]并且准备锁存[DAT_I()]和[TGD_I()]。
・时钟沿 5：
◆主设备锁存[DAT_I()]和[TGD_I()]上的数据。
◆从设备无效[ACK_I]以加入一个等待状态。
・准备沿 6：
◆从设备译码输入并相应的声明[ACK_I]。
◆从设备在[DAT_I()]和[TGD_I()]上提供有效的数据。
◆主设备监视[ACK_I]并且准备锁存[DAT_I()]和[TGD_I()]。
・时钟沿 6：
◆主设备锁存[DAT_I()]和[TGD_I()]上的数据。
◆主设备通过无效[STB_O]和[CYC_O]结束循环。

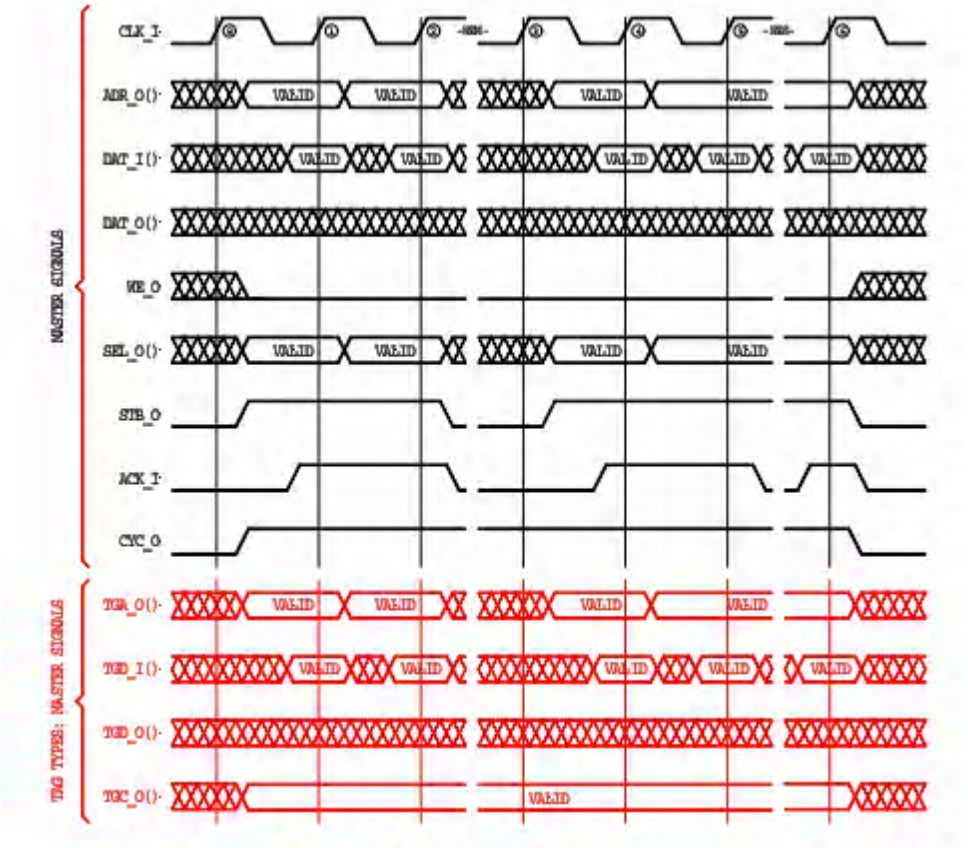


图 1-8　Wishbone 的 BLOCKREAD 循环

BLOACKWRITE 循环流程如下：

・时钟沿 0：

◆主设备提供[ADR_O()]和[TGA_O()]。

◆主设备声明[WE_O]以指示一个 WRITE 循环。

◆主设备提供选择信号[SEL_O()]以指示数据的位置。

◆主设备声明[CYC_O]和 [TGC_O()]以指示循环开始。

◆主设备声明[STB_O]。

・准备沿 I：

◆从设备译码输入并相应的声明[ACK_I]。

◆从设备准备锁存[DAT_O()]和[TGD_O()]上的数据。

◆主设备监视[ACK_I]并且准备结束当前的数据传输。

・时钟沿 I：

◆从设备锁存[DAT_O()]和[TGD_O()]上的数据。

◆主设备提供[ADR_O()]和[TGA_O()]。

◆主设备提供新的选择信号[SEL_O()]以指示数据的位置。

・准备沿 2：

◆从设备译码输入并相应的声明[ACK_I]。

◆从设备准备锁存[DAT_O()]和[TGD_O()]上的数据。

◆主设备监视[ACK_I]并且准备结束当前的数据传输。

・时钟沿 2：

◆从设备锁存[DAT_O()]和[TGD_O()]上的数据。

◆主设备无效[STB_O]以加入一个等待状态(WSM)。

・准备沿 3：

◆从设备根据[STB_O]无效[ACK_I]。

● 时钟沿 3：

◆主设备提供[ADR_O()]和[TGA_O()]。

◆主设备提供选择信号[SEL_O()]以指示数据的位置。

◆主设备声明[STB_O]。
・准备沿 4：
◆从设备译码输入并相应的声明[ACK_I]。
◆从设备准备锁存[DAT_O()]和[TGD_O()]上的数据。
◆主设备监视[ACK_I]并且准备结束数据传输。
・时钟沿 4：
◆从设备锁存[DAT_O()]和[TGD_O()]上的数据。
◆主设备提供[ADR_O()]和[TGA_O()]。
◆主设备提供新的选择信号[SEL_O()]以指示数据的位置。
・准备沿 5：
◆从设备译码输入并相应的声明[ACK_I]。
◆从设备准备锁存[DAT_O()]和[TGD_O()]上的数据。
◆主设备监视[ACK_I]并且准备结束数据传输。
・时钟沿 5：
◆从设备锁存[DAT_O()]和[TGD_O()]上的数据。
◆从设备无效[ACK_I]以加入一个等待状态。
・准备沿 6：
◆从设备译码输入并相应的声明[ACK_I]。
◆从设备准备锁存[DAT_O()]和[TGD_O()]上的数据。
◆主设备监视[ACK_I]并且准备结束数据传输。
・时钟沿 6：
◆从设备锁存[DAT_O()]和[TGD_O()]上的数据。
◆主设备通过无效[STB_O]和[CYC_O]结束循环。

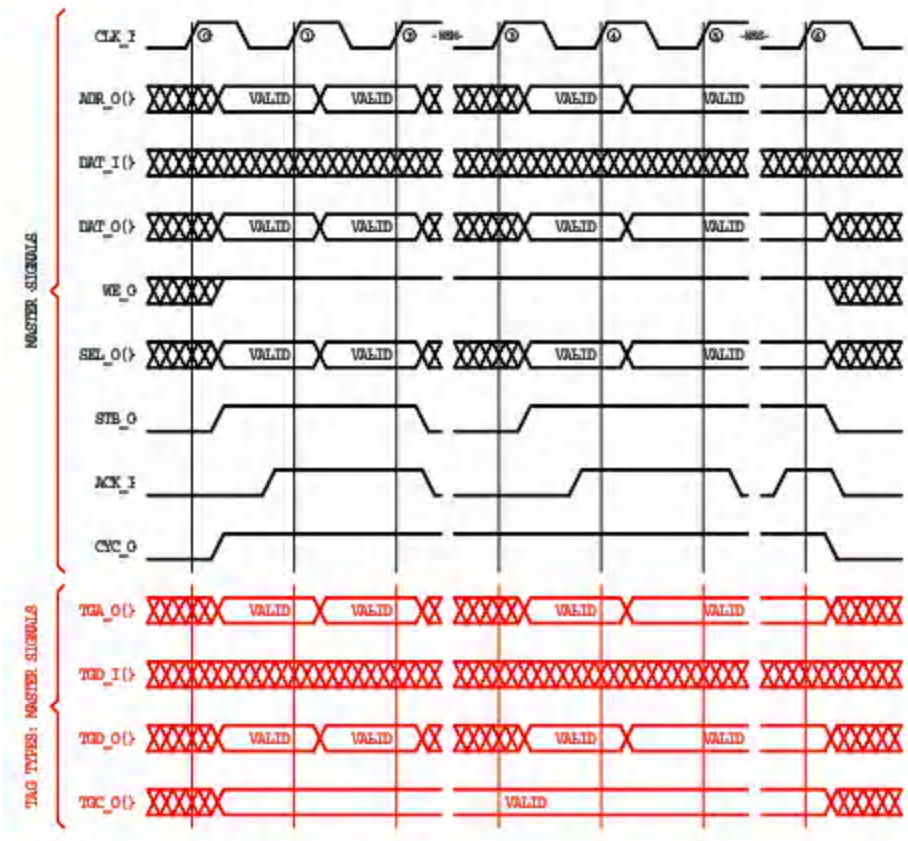


图 1-9 Wishbone 的 BLOCKWRITE 循环

RMw 循环流程如下：
・时钟沿 0：
◆主设备提供[ADR_O()]和[TGA_O()]。

◆主设备无效[WE_O]以指示一个 READ 循环。
◆主设备提供选择信号[SEL_O()]以指示数据的位置。
◆主设备声明[CYC_O]和[TGC_O()]以指示循环开始。
◆主设备声明[STB_O]。
·准备沿 I：
◆从设备译码输入并相应的声明[ACK_I]。
◆从设备在[DAT_I()]和[TGD_I()]上提供有效的数据。
◆主设备监视[ACK_I]并且准备锁存[DAT_I()]和[TGD_I()]。
·时钟沿 I：
◆主设备锁存[DAT_I()]和[TGD_I()]上的数据。
◆主设备无效[STB_O]以加入一个等待状态(WSM)。
·准备沿 2：
◆从设备根据[STB_O]无效[ACK_I]。
◆主设备声明[WE_O]以指示一个 WRITE 循环。
◆注意：主设备在这个时候可以加入任意数量的等待状态。
·时钟沿 2：
◆主设备提供 WRITE 数据[DAT_O()]和[TGD_O()]。
◆主设备提供新的选择信号[SEL_O()]以指示数据的位置。
◆主设备声明[STB_O]。
·准备沿 3：
◆从设备译码输入并相应的声明[ACK_I]。
◆从设备准备锁存[DAT_O()]和[TGD_O()]上的数据。
◆主设备监视[ACK_I]并且准备结束数据传输。
·时钟沿 3：
◆从设备锁存[DAT_O()]和[TGD_O()]上的数据。
◆主设备无效[STB_O]和[CYC_O]指示循环结束。
◆从设备根据[STB_O]的无效而无效[ACK_I]。

图 1-9 Wishbone 的 BLOCKWRITE 循环

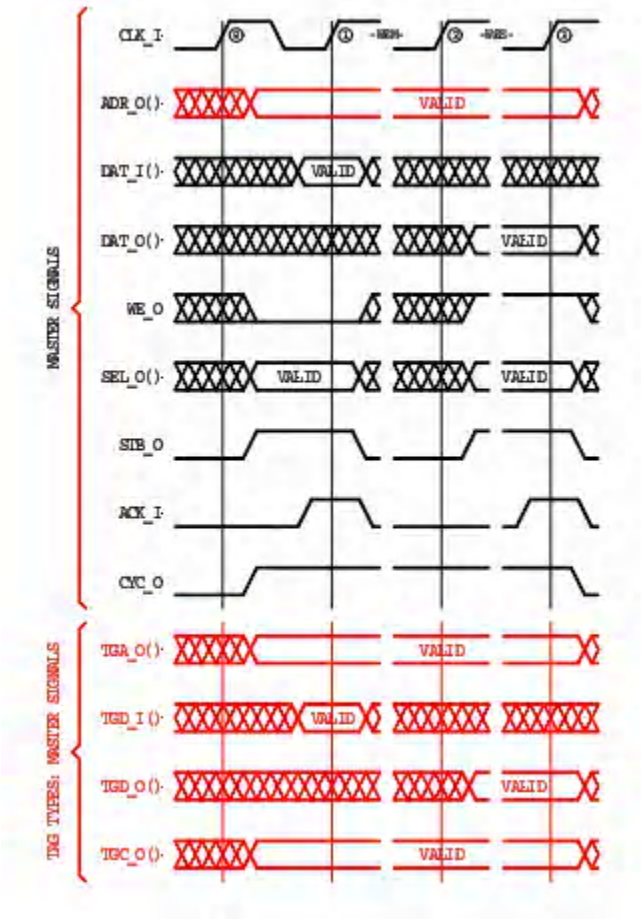


图 1-10 Wishbone 的 RMW 循环

正是由于 Wishbone 总线目的是作为一种 IP 核之间的通用接口，所以我们必须掌握它就象 Linux 的驱动程序一样重要，你可以先不阅读 Linux 源代码，但你需要学习驱动程序的设计，下面列两个 Wishbone 接口的列子供参考。

Wishbone 的 Master 的写操作（一个简单例子）

```
module Wishbone_master(
 wb_clk_i,wb_rst_i,
    wb_adr_o,wb_dat_i,wb_dat_o,
    wb_sel_o,wb_we_o, wb_stb_o,wb_cyc_o,
    wb_ack_i);
input        wb_clk_i;
input        wb_rst_i;
input        wb_ack_i;
input[31:0]  wb_dat_i;
output[31:0] wb_adr_o;
output[31:0] wb_dat_o;
output[3:0]  wb_sel_o;
output       wb_we_o;
output       wb_stb_o;
output       wb_cyc_o;
reg[31:0]    wb_adr_o;
reg[31:0]    wb_dat_o;
reg[3:0]     wb_sel_o;
reg          wb_we_o;
reg          wb_stb_o;
```

```
reg          wb_cyc_o;
reg[3 : 0]   State;
parameter   sta0  = 4'b0001;
parameter   sta1  = 4'b0010;
parameter   sta2  = 4'b0100;
reg[7:0]    temp;
reg         init_sel;
always @(posedge wb_clk_i or posedge wb_rst_i)
begin
 if(wb_rst_i)
 begin
  wb_cyc_o  <= 1'b0;
  wb_stb_o  <= 1'b0;
 end
 else if(init_sel)
 begin
   wb_adr_o <= 0;
   wb_dat_o <= 0;
   wb_sel_o <= 4'b1111;
   wb_cyc_o <= 1'b1;
   wb_stb_o <= 1'b1;
   wb_we_o  <= 1'b1;
 end
 else if(wb_ack_i)
 begin
  wb_cyc_o <= 1'b0;
  wb_stb_o <= 1'b0;
end
end

always @(posedge wb_clk_i or posedge wb_rst_i)
begin
  if(wb_rst_i)
    begin
       init_sel   <= 1'b0;
       temp       <= 8'd0;
       State      <= sta01
    end
  else
  case(State)
    sta0:
       begin
        if(temp == 8'd1)  init_sel   <= 1'b1;
        if(temp == 8'd2)  init_sel   <= 1'b0;
        if(wb_ack_i)
           State  <= sta0;
        else temp <= temp + 1'b1;
       end
    sta1:
       begin
```

```
         State      <= sta1;
      end
    default :
    begin
        State <= sta0;
    end
  endcase
end
endmodule
```

Wishbone 的 Slave 的简单外设(内存)

```
module wb_mem(
  wb_clk_i, wb_rst_i,
  wb_dat_i, wb_dat_o, wb_adr_i, wb_sel_i, wb_we_i, wb_cyc_i,
  wb_stb_i, wb_ack_o, wb_err_o
);
parameter     aw = 8;
parameter    dw = 8;
input           wb_clk_i;
input           wb_rst_i;
input[31:0]  wb_dat_i;
output[31:0] wb_dat_o;
input[31:0]  wb_adr_i;
input[3:0]   wb_sel_i;
input           wb_we_i;
input           wb_cyc_i;
input           wb_stb_i;
output       wb_ack_o;
output       wb_err_o;
assign      wb_err_o = 1'b0;
reg[31:0]   wb_dat_o;
reg         wb_ack_o;
reg[dw-1:0] mem0[(1<<aw)-1:0];
always @(posedge wb_clk_i or posedge wb_rst_i)
    if(wb_rst_i)
        wb_ack_o<= 1'b0;
    else if(wb_cyc_i & wb_stb_i )
        wb_ack_o<= 1'b1;
    else
        wb_ack_o<= 1'b0;
always@(posedge wb_clk_i)
begin
   wb_dat_o[7: 0]  <= mem0[wb_adr_i[((aw)-1):2]];
  if(wb_cyc_i & wb_stb_i & wb_we_i & wb_sel_i[0])
    mem0[wb_adr_i[((aw)-1):2]] <= wb_dat_i[7 : 0];
end
endmodule
```