Avalon MM-Slave

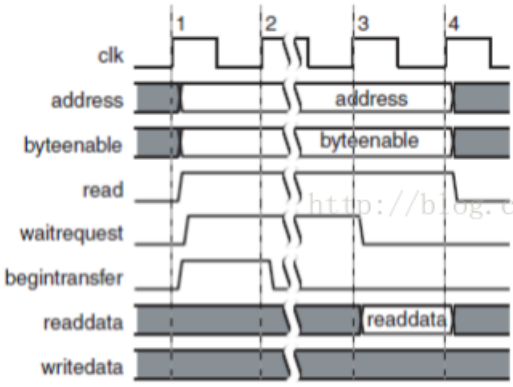
> **FROM: https://blog.csdn.net/chen495277820/article/details/78658688**

1. Avalon MM-Slave

Signals

Classical Transfer Scenarios

Classical Read



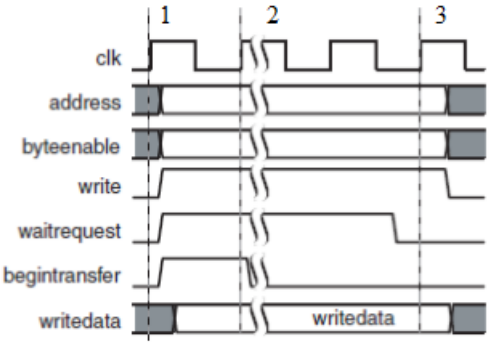
1.在第一个时钟上升沿得到address，byteenable，read，begintransfer信号，同时置位waitrequest（可选）。

2.第二个时钟上升沿begintransfer清零。由于waitrequest置位，进入等待状态。

3.经过若干周期的等待，在一个时钟上升沿waitrequest清零，置位readdata信号。

4.read清零，读过程结束。可开始其他传输过程。

Classical Write



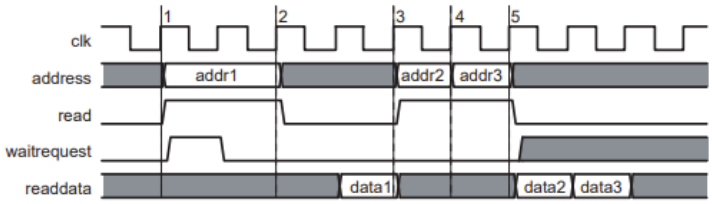
1. 第一个时钟上升沿得到address，byteenable，write，writedata信号，同有效begintransfer，waitrequest（可选）。

2. 第二个时钟上升沿begintransfer清零。由于waitrequest置位，进入等待状态。

3. 经过若干周期的等待，waitrequest清零，在某个时钟上升沿获得writedata。write清零，写过程结束。

Transfer with Fixed Wait Time

Figure 3–6. Slave Pipelined Read Transfer with Fixed Latency of Two Cycles



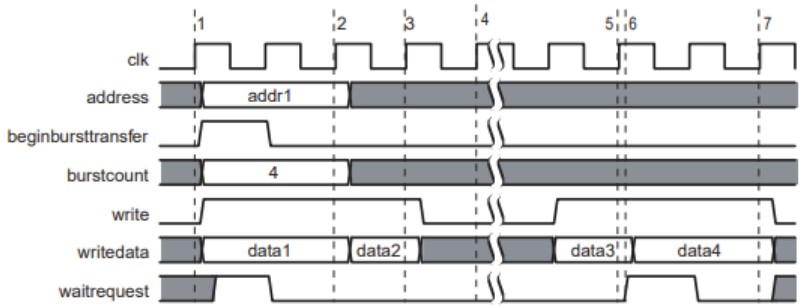
Notes to Figure 3–6:

(1) A master initiates a read transfer by asserting `read` and `addr1`. The slave asserts `waitrequest` to hold off the transfer for one cycle.
(2) The slave deasserts `waitrequest` and captures `addr1` at the rising edge of `clk`. The address phase ends here.
(3) The slave presents valid `readdata` after 2 cycles, ending the transfer.
(4) `addr2` and `read` are asserted for a new read transfer.
(5) The master initiates a third read transfer during the next cycle, before the data from the prior transfer is returned.

The address phase for fixed latency slave read transfers is identical to the variable latency case. After the address phase, a pipelined slave port with fixed read latency takes a fixed number of clock cycles to return valid `readdata`, as indicated by the `readWaitTime` property

Burst Transfer

Burst Write



Notes to Figure 3–7:

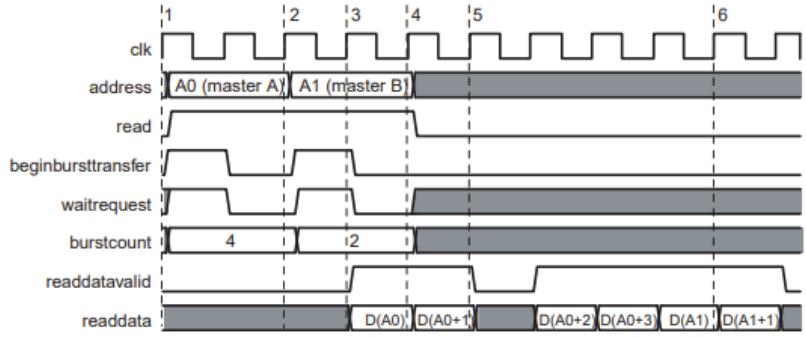
(1) The master asserts `address`, `burstcount`, `write`, and drives the first unit of `writedata`. The slave immediately asserts `waitrequest`, indicating that it is not ready to proceed with the transfer.
(2) `waitrequest` is low; the slave captures `addr1`, `burstcount`, and the first unit of `writedata` . On subsequent cycles of the transfer, `address` and `burstcount` are ignored.
(3) The slave port captures the second unit of data at the rising edge of `clk`.
(4) The burst is paused while `write` is deasserted.
(5) The slave captures the third unit of data at the rising edge of `clk`.
(6) The slave asserts `waitrequest`. In response, all outputs are held constant through another clock cycle.
(7) The slave captures the last unit of data on this rising edge of `clk`. The slave write burst ends.

从接口写突发传输遵循如下规则：

1. 在突发写传输达到规定的数量之前，**主从对一直会被锁定**。
2. `write` 信号被置位时才获取 `writedata` 。
3. 可以用 `waitrequest` 信号来延迟突发传输。此时， `writedata` ， `write` 和 `byteenable` 保持不变

Burst Read

Figure 3–8. Slave Read Burst



Notes to Figure 3–8:

(1) Master A asserts `address` (`A0`), `burstcount`, and `read` after the rising edge of `clk`. The slave asserts `waitrequest`, causing all inputs except `beginbursttransfer` to be held constant through another clock cycle.
(2) The slave captures `A0` and `burstcount` at this rising edge of `clk`. A new transfer could start on the next cycle.
(3) Master B drives `address` (`A1`), `burstcount`, and `read`. The slave asserts `waitrequest`, causing all inputs except `beginbursttransfer` to be held constant. The slave could have returned read data from the first read request at this time, at the earliest.
(4) The slave presents valid `readdata` and asserts `readdatavalid`, transferring the first word of data for master A.
(5) The second word for master A is transferred. The slave deasserts `readdatavalid` pausing the read burst. The slave port can keep `readdatavalid` deasserted for an arbitrary number of clock cycles.
(6) The first word for master B is returned.

从接口突发读传输和有可变延迟的流传输类似。唯一的不同点就是一次突发传输只传输一个地址。

从接口读突发传输遵循如下规则：

1. `burstcount`位N，则从设备回复N个words的数据。
2. 可以用 `readdatavalid`来延迟读突发传输。

Address Aligned

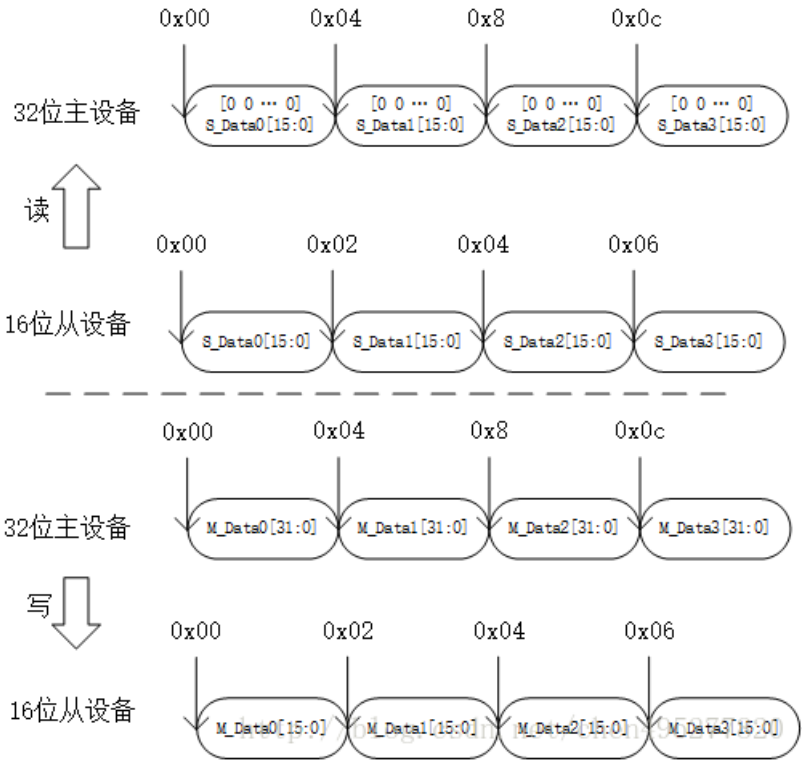
数据宽度不同的主从接口通讯存在地址对齐的问题。Avalon内存映射接口能够解决地址对齐的问题，所以任何宽度的接口都可以进行互联。此时存在三种情况

主从接口宽度相等

这种情况是我们最希望看到的情况，此时接口间可以很方便的进行通讯

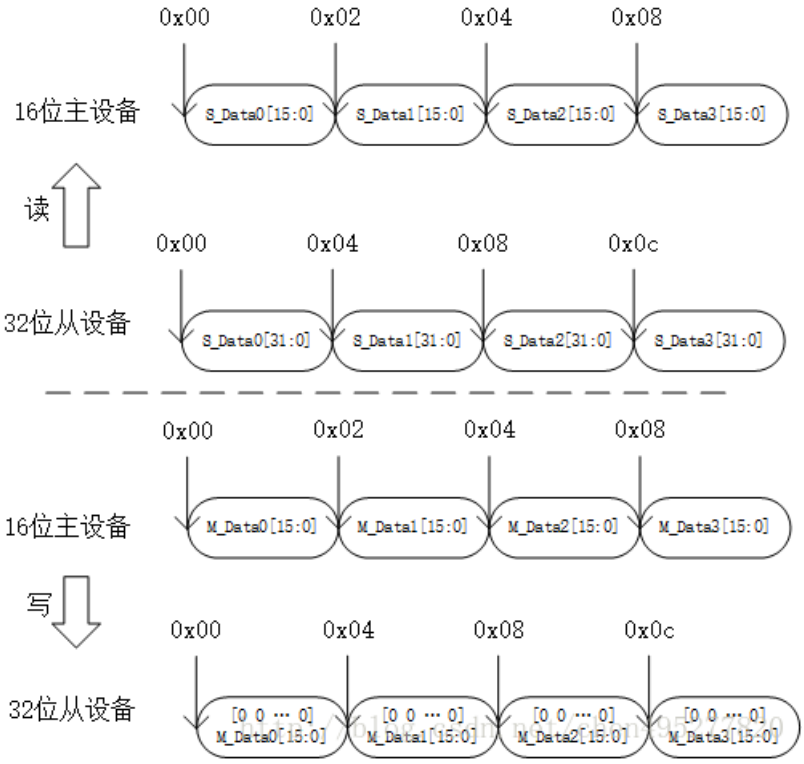
主接口宽度大于从接口

举例说明，当32位的主接口要读取16位从接口，那么总线会执行2次读传输（32位数据），将32位的数据传输给主接口。、



主接口宽度小于从接口

举例说明，16位的主接口要读取32位从接口的数据，此时总线会选取合适的byte来压缩数据。16位的主接口要向32位从接口写数据时数据，通过控制byteenable信号来写数据


0x00
0x02
0x04
0x08
16位主设备
S_Data0[15:0]
S_Data1[15:0]
S_Data2[15:0]
S_Data3[15:0]
读
0x00
0x04
0x08
0x0c
32位从设备
S_Data0[31:0]
S_Data1[31:0]
S_Data2[31:0]
S_Data3[31:0]
0x00
0x02
0x04
0x08
16位主设备
M_Data0[15:0]
M_Data1[15:0]
M_Data2[15:0]
M_Data3[15:0]
写
0x00
0x04
0x08
0x0c
32位从设备
[0 0 … 0]
M_Data0[15:0]
[0 0 … 0]
M_Data1[15:0]
[0 0 … 0]
M_Data2[15:0]
[0 0 … 0]
M_Data3[15:0]