精益求精之 Avalon

大家一定发现了，SOPC builder 下面的模块很有限。事实上，作为一个公司也不可能满足全世界的需求。所以在做一些系统设计的时候，不得不做一些新的模块（Component）。而做这些模块的关键在于两方面，首先当然是模块本身的功能，另外的就是模块的接口。功能要靠大家自己努力，谁都帮不了你。但是对接口我们可以稍微看一下。对接口的熟悉，对于模块设计可以说是至关重要的。选择什么样的接口，如何选择。所以我们反而对 Avalon 接口需要花更多的精力，并且是值得的。

Avalon 接口分成两种，一种是 Avalon-MM 接口，偶然我们会叫他美眉接口。另一种是 Avalon-ST 接口，因为出来的时间还不够长，暂时没啥绰号。MM 接口，是通过地址来读写数据，更多的是用在控制逻辑上面。ST 接口是用于点到点的流数据接口，更多的可以用在有高速通过率的模块中间。这两个接口本身并没有矛盾，不是说势不两立的，一个模块中既可以有 MM 接口，甚至几个 MM 接口，也可以同时存在 ST 接口。作为一个点对点的接口定义，Avalon 可以做到高效的接口效果。这与 PCI 之类的总线接口是有本质区别的。PCI 总线可以看作是铁路轨道，当一个火车在轨道上行驶的时候，就不可以有另一个火车同时使用轨道，否则就见鬼了。Avalon 接口更多好像高速公路，你开一个车从你家到别人家里。另一个人可以从他家到另外一个人家里。并不是说，你用了高速公路，就不允许别人用了，除非你是什么国家总统。所以这种接口方式，不会因为总线被占据而延误传输时间。当然，如果当你和另一个人都需要去同一个人家里的时候，你就需要做一些仲裁了，否则，就要撞车。

Avalon-MM 篇

美眉-从端口

美眉接口分为主接口和从接口。无论是读写的操作，都是由主接口发出的指令，然后从接口被动的接受操作。这蛮容易理解的，在美眉面前，美眉就是主接口，追美眉的那个傻老爷们就是从接口。所以我们先介绍一下这个傻老爷们-从接口。

信号	输入/输出	位宽	描述
Address	In	1-32	读写操作的地址
Byteenable/Byteenable_n	In	2^n n=0-7	字节有效信号
Read / Read_n	In	1	读信号
Readdata	Out	8*(2^n) n=0-7	读出去的数据
Write/write_n	In	1	写信号
Writedata	In	8*(2^n)	写进来的数据

		n=0-7	
Begintransfer	In	1	操作开始信号
等待信号			
Waitrequest/waitrequest_n	Out	1	表示无法接受新的读写操作
流水处理信号			
Readdatavalid readdatavalid_n	Out	1	返回数据有效信号
Burst 处理信号			
Burstcount	In	1-32	显示需要 burst 的数据数量
Beginbursttransfer	In	1	Burst 处理开始信号
流控信号			
Readyfordata	Out	1	代表有可以写入的空间
Dataavailable	Out	1	代表有数据可以被读出
复位信号			
Resetrequest	Out	1	可以用来复位整个 Avalon-mm 系统

除了信号，还有一些具体的对端口的设置：

设置名	初始值	范围	描述
ReadLatency	0	0-63	当读操作的延迟为已知的时候，做一些设置。如果接口中有 readdatavalid 的信号就不需要设置
writeWaitTime	0	0-1000	字节有效信号
ReadWaitTime	1	0-1000	读信号
maximumPending ReadTransactions	1	1-64	读出去的数据
BurstOnBurst BoundariesOnly	False	True, false	写信号
LineWrapBursts	False	True,false	写进来的数据
MaxBurstSize	1	64	操作开始信号
BridgesToMaster	无	Avalon-MM master on same component	表示无法接受新的读写操作
AssociatedClock			端口相关的时钟

所谓美眉从端口，无非就是美眉向你提出要求，要你去抽屉里面拿点东西(read)，或者放点东西罢了(write)。美眉的命令当然是要执行的，但是执行是有很多方法来操作的。我们来看看我们可以怎么做。

- **非定时传输套装：**

信号组合：

通用信号	读信号	写信号	参数设置
Clk	Read	Write	
Reset	Readdata	Writedata	
Address			
Waitrequest			
Byteenable			

是这么一种情况，美眉要你做事情的同时，你给她一块牌子，叫做 Waitrequest (置高)。就是说你等着，我去拿，或者我去放。等到你按照她的要求拿好了东西，给她的同时，把牌子拿回来（置低）。这样她就拿到东西高高兴兴走了。同样的，当你把东西放好以后，回来把，牌子收回来，她也就满意了。在你完成你的任务之前，美眉会一直傻傻的等待着 (保持读写状态)。

- **定延迟传输操作套装**

信号组合

通用信号	读信号	写信号	参数设置
Clk	Read	Write	ReadWaitTime
Reset	Readdata	Writedata	WriteWaitTime
Address			
Byteenable			

在这种状况下，美眉对你是非常了解了，她很确定的知道，你拿东西(readwaittime)和放东西（writewaittime）的时间会有多长。所以我们这里就不需要那个 waitrequest 的牌子了。她告诉你需要做什么，然后等待响应的时间。她就知道事情完成了。该拿的东西应该可以送到了，该放的东西也已经放好了。

- **非定时流水套装**

信号组合：

通用信号	读信号	参数设置
Clk	Read	maximumPending ReadTransactions
Reset	Readdata	
Address	readdatavalid	
Waitrequest		
Byteenable		

这个情况只适用于拿东西，不适合于放东西。这个美眉会比较残暴，她没耐心等你把东西拿来才发出下一个要求。她会一直发要求，虽然她并不知道可能会花多少时间去拿来。所以她可以每个时钟都发出拿东西的请求，告诉你地址，你就不断的去拿来。由于可能回来的时间是不同的，所以你需要提醒一下她东西来了，所以当我们把东西拿来的时候，我们同时给她一个 readdatavalid 的牌子。这样她就知道她要的东西来了。在这里我们需要一个前提，就是

东西是一样一样去拿的。换句话说就是后发出的请求和回来的东西的顺序一定是相同的。否则后到的地址，先拿来东西，美眉要翻脸了。

这样当然效率会高很多了，但是我们需要有一点反抗精神。我们不能太纵容她了。所以在我们很忙的时候，或者想罢工的时候，就毫不留情的给她一块 Waitrequest 的牌子说，等着，我现在没空。在这段时候，对她的要求不予理睬，让她眼巴巴的等着。

在这个模式下面有一个名字超长的设置: maximumpendingreadtransactions， 它是对模块的一个附加说明。说明这个模块最大能接受的流水量。也就是说最大的可以容忍的美眉的要求。超过这个要求的话，那只好说对不起了。

- **定时流水套装**

通用信号	读信号	参数设置
Clk	Read	maximumPending ReadTransactions
Reset	Readdata	`ReadLatency`
Address	readdatavalid	
Byteenable		

这是一个对你比较了解的残暴的美眉（天哪）。她知道你会用多少时间来拿东西（readwaittime）。所以她在发出拿指令以后，过几个时钟就可以拿到东西。这个时候我们就不需要那个 readdatavalid 的牌子了。但是 waitrequest 还是要的，为了保护我们自己的权益。她拿着那块牌子的时候，我们什么都不做，所以她需要等待的时间其实是 waitrequest + readwaittime。

- **批处理套装**

这是一个比较内向的美眉，她不喜欢不厌其烦的告诉每次操作的抽屉位置。她只是告诉你第一次的位置，和希望拿（放）多少东西就好了。所以管他叫批处理套装。

通用信号	读信号	写信号	参数设置
Clk	Read	Write	LineWrapBursts
Reset	Readdata	Writedata	MaxBurstSize
Address			
Waitrequest			
BurstCount			
BeginBurstTransfer			
Byteenable			

批处理写套装

在批处理写套装时，会有一个 beginbursttransfer 命令，随着这个命令，会告诉你地址，数据，已经要求批处理的数量（burstcount）。Waitrequest 对 beginbursttransfer 无效，她不管你是不是给她那个牌子，她会告诉你她需要批处理的信息。但是地址，burstcount,写信号以及写的

东西，必须要保持到 waitrequest 撤销的时候。在第一个数据以后的传递中，她不再需要告诉你地址和数量，只要告诉你要放（write）和要放什么东西(writedata)就好了。你就应该很自觉自愿的从命。

批处理读套装

与放东西一样，拿东西的指令也是由 beginbursttransfer 发出的。依然告诉你读的初始地址。但是在读的时候，不需要等到数据返回，可以继续发起下一次操作（another beginbursttransfer）。而拿回来东西的时候，还需要给一个 readdatavalid 的牌子告诉一下。

- **流控**

我们也需要提高一些服务质量。有些情况是美眉不知道的。比如说，我这个抽屉里现在是空的，还是满的。如果是空的，就不能再来拿东西了，如果是满的，就不能在塞东西了。所以我们需要用 davaavailable 来表示，现在抽屉里东西不是空的，你可以来拿。用 readyfordata 来表示抽屉不是满的，你可以放东西。

美眉-主端口：

了解了从端口的状况，其实我们对于主端口也可以知道个大概了。这里需要有一些说明，那就是主端口和从端口之间，并不是直接连接的，中间会有一个叫 Avalon fabric switch 的东西。他从中进行调整和一些自动加入的控制逻辑。这样可以保证一个从端口可以更多的适用于不同的主端口，而不需要太多的考虑对方的状况。而很多主端口的功能，通过 switch 来实现，从从端口看起来，是看不到的。好比很多美眉内心的想法，我们是体会不出来的，而她们会通过一些方式来暗示达到她们的要求。而我们就是傻呼呼的照做就好了。我们这里就主要说一下两个信号： arbiterlock, flush.

Arbiterlock, 这个信号的意义很简单，就是锁定仲裁。好比你在帮好几个美眉做事情，大家说好了要分享你的。而偏偏这个美眉比较霸道，于是她恶狠狠的举起这块 arbiterlock 的牌子，那么在一段时间里面，你就被她独享了。而其他的美眉只能可怜巴巴的等着。

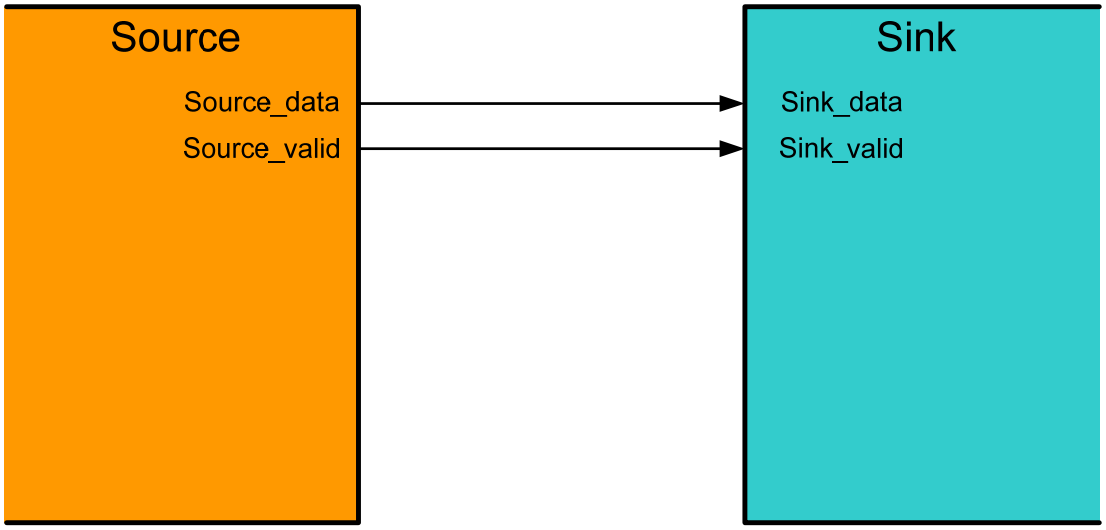
Flush：这个信号是使用在流水读的状态下。这一定是一个善变的美眉了。她给你发出去一堆取东西的要求。那你得一个个做吧，然后她突然之间发飙说，我不要了，举起这个 flush 的牌子。于是所有在这之前发出的要求全部取消。还是那句话，这个牌子她不是举给你看的，而是举给 switch 看的。Switch 会把你取回来的东西放弃掉。所以你其实还是很可怜的一样一样的去拿来的。

Avalon-ST 篇

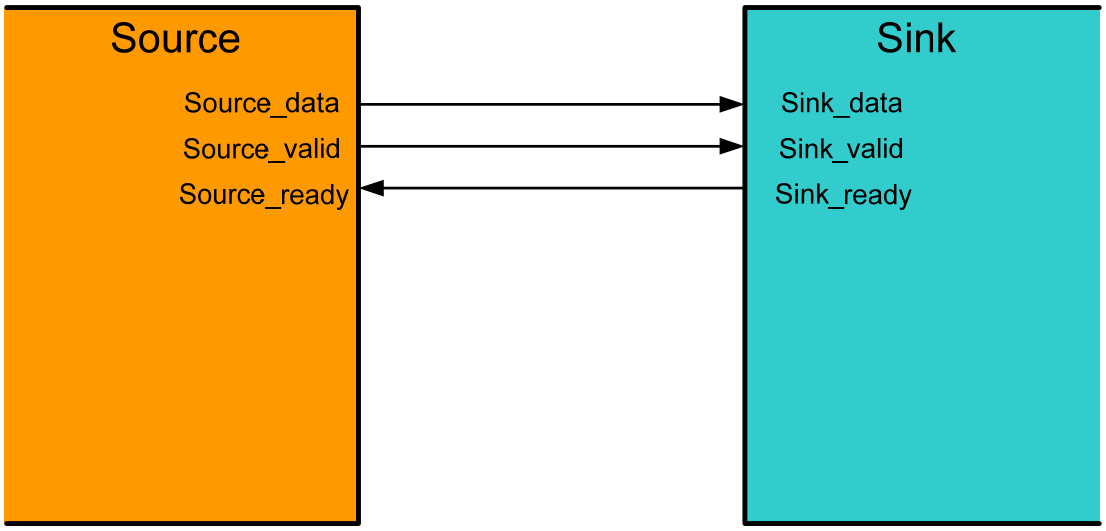
与美眉接口不同，Avalon-ST 更多的适用于一些传递速度要求比较高，没有地址需求的应用方面。比如一些 DSP，包处理方面的应用，FIR, FFT 什么的。更多的是对数据的一种传递。有这样一些想法

- 点对点的传输
- 多通道的传输
- 包传输
- 自动的接口调整

看上去很可怕的样子，其实很简单，我们看这么几张图就全部了解了。



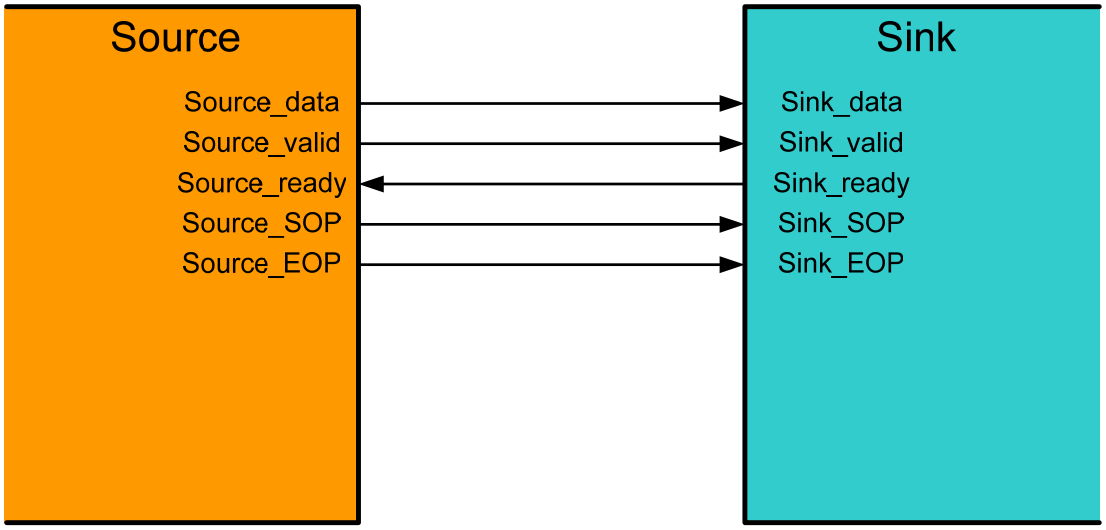
一个问题的两个方面，一个接口的两个部分。我们有一个 Source 和一个 Sink 部分。Source 是源，数据从这个接口发出，然后由 sink 来接受。数据就是从 data 这个信号发出来的，而 valid 信号表示当前这是一个有效信号



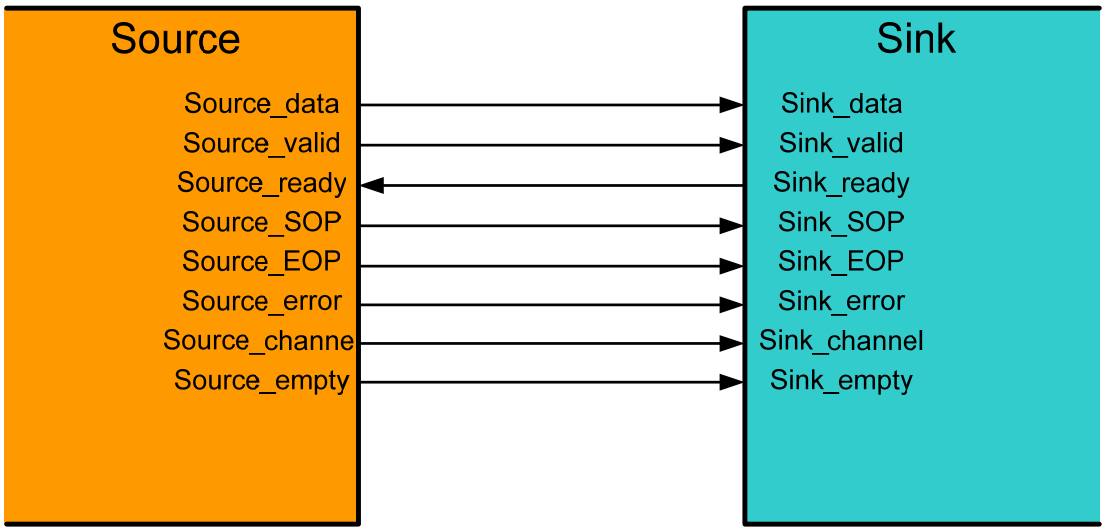
Sink 并不是什么时候都可以接受数据的，所以它通过 ready 信号告诉 source，我有没有准备好接受数据我们称之为反压。这样的效果就是可以一直把信息传递到前面，然一切操作可以停止下来。这里面有一个非常重要的参数 ReadyLatency，这代表说，在这个 ready 信号起来以后几个时钟 sink 可以接受数据。我们分两种情况来看：

ReadyLatency=0，这个时候呢，在 ready 为高的同时 sink 就可以开始接受数据。即便在 ready 为低的时候，source 依然可以把 valid 设为高，但是这个时候，sink 是不接受数据的，只有当 ready 和 valid 同时为高的时候，sink 才接受数据。

ReadyLatency 不是 0 的时候。代表 ready 为高以后的几个时钟才能接受数据。这个状况下，source 必须严格控制 valid 信号。在 ready 信号为低，以及为高的前几个时钟内（readlatency）都不得为高。而在 ready 信号由高变低的几个时钟内(readlatency),valid 信号还是可以为高的。这样，sink 端口就只需要监视 valid 信号就好了，控制上会更简单一些。



包处理：在很多应用中，我们会用到包这么个概念。我们用 SOP 和 EOP 来指示一个包。SOP 就是 Start of Packet, EOP 就是 end of packet.好了，我想不需要我啰嗦了。



为了把故事讲完整了，把其他信号也加进来。

Channel：用于多通道处理，指示当前数据是属于什么 channel 的
Error: 错误信号，可以用来传递错误指示，表示当前信号的一个状态。这个信号可以灵活运用，并不是一定要用来传递错误哦。
Empty: 在 EOP 的时候，指示数据中的哪几个 Byte 是有效数据，很像 Mask。

其实 Avalon 接口并不是什么非常复杂的东西，但是应用到具体的模块中，就需要大家自己灵活掌握，发挥创意。对他们更深入的了解对于更好地把握传输，控制是非常重要的，对于系统的理顺和增强也是很有意思的。当然还是不太需要拘泥于具体的形式。