精益求精之 Avalon

大家一定發現了，SOPC builder 下面的模組很有限。事實上，作爲一個公司也不可能滿足全世界的需求。所以在做一些系統設計的時候，不得不做一些新的模組（Component）。而做這些模組的關鍵在於兩方面，首先當然是模組本身的功能，另外的就是模組的介面。功能要靠大家自己努力，誰都幫不了你。但是對介面我們可以稍微看一下。對介面的熟悉，對於模組設計可以說是至關重要的。選擇什麼樣的介面，如何選擇。所以我們反而對 Avalon 介面需要花更多的精力，並且是值得的。

Avalon 介面分成兩種，一種是 Avalon-MM 介面，偶然我們會叫他美眉介面。另一種是 Avalon-ST 介面，因爲出來的時間還不夠長，暫時沒啥綽號。MM 介面，是通過位址來讀寫資料，更多的是用在控制邏輯上面。ST 介面是用於點到點的流資料介面，更多的可以用在有高速通過率的模組中間。這兩個介面本身並沒有矛盾，不是說勢不兩立的，一個模組中既可以有 MM 介面，甚至幾個 MM 介面，也可以同時存在 ST 介面。作爲一個點對點的介面定義，Avalon 可以做到高效的介面效果。這與 PCI 之類的匯流排界面是有本質區別的。PCI 匯流排可以看作是鐵路軌道，當一個火車在軌道上行駛的時候，就不可以有另一個火車同時使用軌道，否則就見鬼了。Avalon 介面更多好像高速公路，你開一個車從你家到別人家裏。另一個人可以從他家到另外一個人家裏。並不是說，你用了高速公路，就不允許別人用了，除非你是什麼國家總統。所以這種介面方式，不會因爲匯流排被佔據而延誤傳輸時間。當然，如果當你和另一個人都需要去同一個人家裏的時候，你就需要做一些仲裁了，否則，就要撞車。

Avalon-MM 篇

美眉-從埠

美眉介面分爲主介面和從介面。無論是讀寫的操作，都是由主介面發出的指令，然後從介面被動的接受操作。這蠻容易理解的，在美眉面前，美眉就是主介面，追美眉的那個傻老爺們就是從介面。所以我們先介紹一下這個傻老爺們-從介面。

信號	輸入/輸出	位寬	描述
Address	In	1-32	讀寫操作的位址
Byteenable/Byteenable_n	In	2^n n=0-7	位元組有效信號
Read / Read_n	In	1	讀信號
Readdata	Out	8*(2^n) n=0-7	讀出去的資料
Write/write_n	In	1	寫信號
Writedata	In	8*(2^n) n=0-7	寫進來的數據

Begintransfer	In	1	操作開始信號
等待信號			
Waitrequest/waitrequest_n	Out	1	表示無法接受新的讀寫操作
流水處理信號			
Readdatavalid readdatavalid_n	Out	1	返回資料有效信號
Burst 處理信號			
Burstcount	In	1-32	顯示需要 burst 的資料數量
Beginbursttransfer	In	1	Burst 處理開始信號
流控信號			
Readyfordata	Out	1	代表有可以寫入的空間
Dataavailable	Out	1	代表有資料可以被讀出
重定信號			
Resetrequest	Out	1	可以用來重定整個 Avalon-mm 系統

除了信號，還有一些具體的對埠的設置：

設置名	初始值	範圍	描述
ReadLatency	0	0-63	當讀操作的延遲爲已知的時候，做一些設置。如果介面中有 readdatavalid 的信號就不需要設置
writeWaitTime	0	0-1000	位元組有效信號
ReadWaitTime	1	0-1000	讀信號
maximumPending ReadTransactions	1	1-64	讀出去的資料
BurstOnBurst BoundariesOnly	False	True, false	寫信號
LineWrapBursts	False	True,false	寫進來的數據
MaxBurstSize	1	64	操作開始信號
BridgesToMaster	無	Avalon-MM master on same component	表示無法接受新的讀寫操作
AssociatedClock			埠相關的時鐘

所謂美眉從埠，無非就是美眉向你提出要求，要你去抽屜裏面拿點東西(read)，或者放點東西罷了(write)。美眉的命令當然是要執行的，但是執行是有很多方法來操作的。我們來看看我們可以怎麼做。

- **非定時傳輸套裝：**

信號組合：

通用信號	讀信號	寫信號	參數設置

Clk	Read	Write	
Reset	Readdata	Writedata	
Address			
Waitrequest			
Byteenable			

是這麼一種情況，美眉要你做事情的同時，你給她一塊牌子，叫做 Waitrequest (置高)。就是說你等著，我去拿，或者我去放。等到你按照她的要求拿好了東西，給她的同時，把牌子拿回來（置低）。這樣她就拿到東西高高興興走了。同樣的，當你把東西放好以後，回來把，牌子收回來，她也就滿意了。在你完成你的任務之前，美眉會一直傻傻的等待著 (保持讀寫狀態)。

- **定延遲傳輸操作套裝**

信號組合

通用信號	讀信號	寫信號	參數設置
Clk	Read	Write	ReadWaitTime
Reset	Readdata	Writedata	WriteWaitTime
Address			
Byteenable			

在這種狀況下，美眉對你是非常瞭解了，她很確定的知道，你拿東西(readwaittime)和放東西（writewaittime）的時間會有多長。所以我們這裏就不需要那個 waitrequest 的牌子了。她告訴你需要做什麼，然後等待回應的時間。她就知道事情完成了。該拿的東西應該可以送到了，該放的東西也已經放好了。

- **非定時流水套裝**

信號組合：

通用信號	讀信號	參數設置
Clk	Read	maximumPending ReadTransactions
Reset	Readdata	
Address	readdatavalid	
Waitrequest		
Byteenable		

這個情況只適用於拿東西，不適合於放東西。這個美眉會比較殘暴，她沒耐心等你把東西拿來才發出下一個要求。她會一直發要求，雖然她並不知道可能會花多少時間去拿來。所以她可以每個時鐘都發出拿東西的請求，告訴你位址，你就不斷的去拿來。由於可能回來的時間是不同的，所以你需要提醒一下她東西來了，所以當我們把東西拿來的時候，我們同時給她一個 readdatavalid 的牌子。這樣她就知道她要的東西來了。在這裏我們需要一個前提，就是東西是一樣一樣去拿的。換句話說就是後發出的請求和回來的東西的順序一定是相同的。否

則後到的地址，先拿來東西，美眉要翻臉了。

這樣當然效率會高很多了，但是我們需要有一點反抗精神。我們不能太縱容她了。所以在我們很忙的時候，或者想罷工的時候，就毫不留情的給她一塊 Waitrequest 的牌子說，等著，我現在沒空。在這段時候，對她的要求不予理睬，讓她眼巴巴的等著。

在這個模式下面有一個名字超長的設置: maximumpendingreadtransactions， 它是對模組的一個附加說明。說明這個模組最大能接受的流水量。也就是說最大的可以容忍的美眉的要求。超過這個要求的話，那只好說對不起了。

- **定時流水套裝**

通用信號	讀信號	參數設置
Clk	Read	maximumPending ReadTransactions
Reset	Readdata	ReadLatency
Address	readdatavalid	
Byteenable		

這是一個對你比較瞭解的殘暴的美眉（天哪）。她知道你會用多少時間來拿東西（readwaittime）。所以她在發出拿指令以後，過幾個時鐘就可以拿到東西。這個時候我們就不需要那個 readdatavalid 的牌子了。但是 waitrequest 還是要的，爲了保護我們自己的權益。她拿著那塊牌子的時候，我們什麼都不做，所以她需要等待的時間其實是 waitrequest + readwaittime。

- **批次處理套裝**

這是一個比較內向的美眉，她不喜歡不厭其煩的告訴每次操作的抽屜位置。她只是告訴你第一次的位置，和希望拿（放）多少東西就好了。所以管他叫批次處理套裝。

通用信號	讀信號	寫信號	參數設置
Clk	Read	Write	LineWrapBursts
Reset	Readdata	Writedata	MaxBurstSize
Address			
Waitrequest			
BurstCount			
BeginBurstTransfer			
Byteenable			

批次處理寫套裝

在批次處理寫套裝時，會有一個 beginbursttransfer 命令，隨著這個命令，會告訴你位址，資料，已經要求批次處理的數量（burstcount）。Waitrequest 對 beginbursttransfer 無效，她不管你是不是給她那個牌子，她會告訴你她需要批次處理的資訊。但是地址，burstcount,寫信號以及寫的東西，必須要保持到 waitrequest 撤銷的時候。在第一個資料以後的傳遞中，她不

再需要告訴你位址和數量，只要告訴你要放（write）和要放什麼東西(writedata)就好了。你就應該很自覺自願的從命。

批次處理讀套裝

與放東西一樣，拿東西的指令也是由 beginbursttransfer 發出的。依然告訴你讀的初始地址。但是在讀的時候，不需要等到資料返回，可以繼續發起下一次操作（another beginbursttransfer）。而拿回來東西的時候，還需要給一個 readdatavalid 的牌子告訴一下。

- **流控**

我們也需要提高一些服務品質。有些情況是美眉不知道的。比如說，我這個抽屜裏現在是空的，還是滿的。如果是空的，就不能再來拿東西了，如果是滿的，就不能在塞東西了。所以我們需要用 davaavailable 來表示，現在抽屜裏東西不是空的，你可以來拿。用 readyfordata 來表示抽屜不是滿的，你可以放東西。

美眉-主埠:

瞭解了從埠的狀況，其實我們對於主埠也可以知道個大概了。這裏需要有一些說明，那就是主埠和從埠之間，並不是直接連接的，中間會有一個叫 Avalon fabric switch 的東西。他從中進行調整和一些自動加入的控制邏輯。這樣可以保證一個從埠可以更多的適用於不同的主埠，而不需要太多的考慮對方的狀況。而很多主埠的功能，通過 switch 來實現，從從埠看起來，是看不到的。好比很多美眉內心的想法，我們是體會不出來的，而她們會通過一些方式來暗示達到她們的要求。而我們就是傻呼呼的照做就好了。我們這裏就主要說一下兩個信號： arbiterlock, flush.

Arbiterlock, 這個信號的意義很簡單，就是鎖定仲裁。好比你在幫好幾個美眉做事情，大家說好了要分享你的。而偏偏這個美眉比較霸道，於是她惡狠狠的舉起這塊 arbiterlock 的牌子，那麼在一段時間裏面，你就被她獨享了。而其他的美眉只能可憐巴巴的等著。

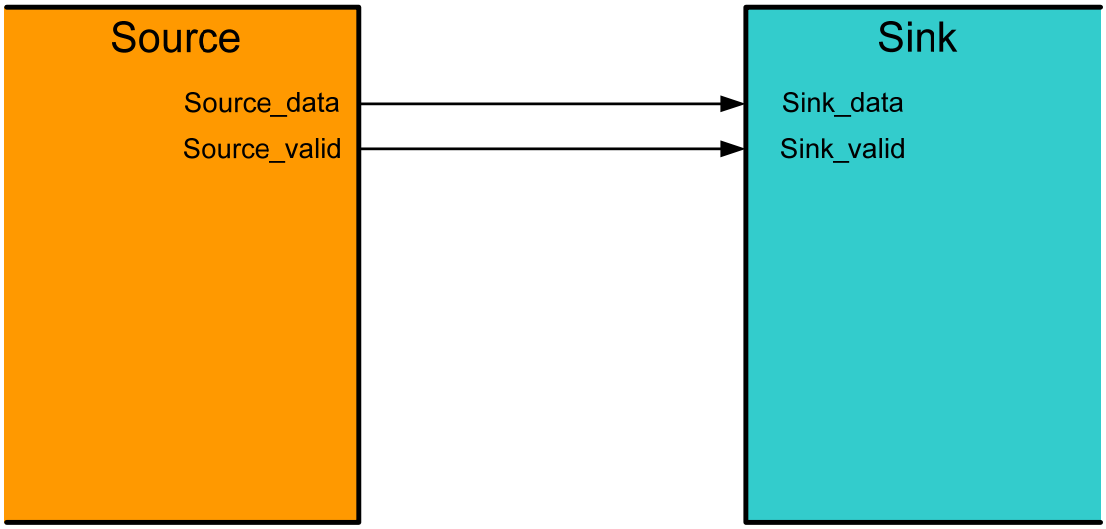
Flush：這個信號是使用在流水讀的狀態下。這一定是一個善變的美眉了。她給你發出去一堆取東西的要求。那你得一個個做吧，然後她突然之間發飆說，我不要了，舉起這個 flush 的牌子。於是所有在這之前發出的要求全部取消。還是那句話，這個牌子她不是舉給你看的，而是舉給 switch 看的。Switch 會把你取回來的東西放棄掉。所以你其實還是很可憐的一樣一樣的去拿來的。

Avalon-ST 篇

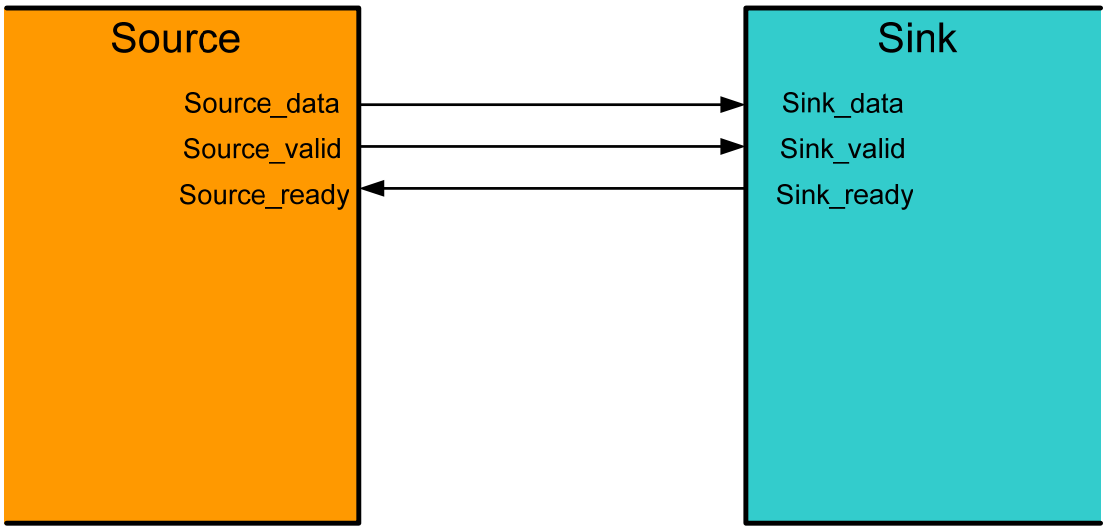
與美眉介面不同，Avalon-ST 更多的適用於一些傳遞速度要求比較高，沒有位址需求的應用方面。比如一些 DSP，包處理方面的應用，FIR, FFT 什麼的。更多的是對資料的一種傳遞。有這樣一些想法

- 點對點的傳輸
- 多通道的傳輸
- 包傳輸
- 自動的介面調整

看上去很可怕的樣子，其實很簡單，我們看這麼幾張圖就全部瞭解了。



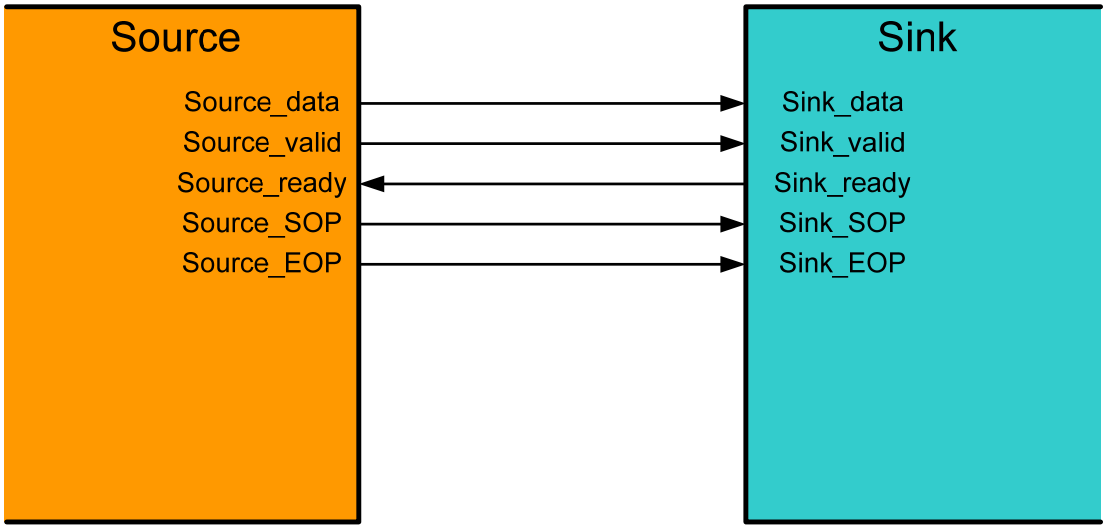
一個問題的兩個方面，一個介面的兩個部分。我們有一個 Source 和一個 Sink 部分。Source 是源，資料從這個介面發出，然後由 sink 來接受。資料就是從 data 這個信號發出來的，而 valid 信號表示當前這是一個有效信號



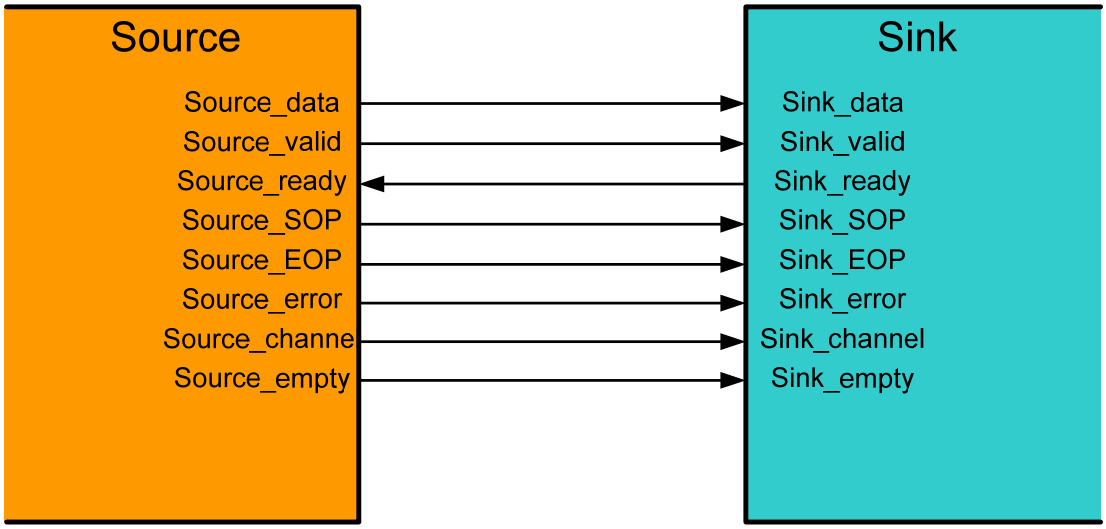
Sink 並不是什麼時候都可以接受資料的，所以它通過 ready 信號告訴 source，我有沒有準備好接受資料我們稱之爲反壓。這樣的效果就是可以一直把資訊傳遞到前面，然一切操作可以停止下來。這裏面有一個非常重要的參數 ReadyLatency，這代表說，在這個 ready 信號起來以後幾個時鐘 sink 可以接受資料。我們分兩種情況來看：

ReadyLatency=0，這個時候呢，在 ready 爲高的同時 sink 就可以開始接受資料。即便在 ready 爲低的時候，source 依然可以把 valid 設爲高，但是這個時候，sink 是不接受資料的，只有當 ready 和 valid 同時爲高的時候，sink 才接受資料。

ReadyLatency 不是 0 的時候。代表 ready 爲高以後的幾個時鐘才能接受資料。這個狀況下，source 必須嚴格控制 valid 信號。在 ready 信號爲低，以及爲高的前幾個時鐘內（readlatency）都不得爲高。而在 ready 信號由高變低的幾個時鐘內(readlatency),valid 信號還是可以爲高的。這樣，sink 埠就只需要監視 valid 信號就好了，控制上會更簡單一些。



包處理：在很多應用中，我們會用到包這麼個概念。我們用 SOP 和 EOP 來指示一個包。SOP 就是 Start of Packet, EOP 就是 end of packet.好了，我想不需要我囉嗦了。



爲了把故事講完整了，把其他信號也加進來。

Channel：用於多通道處理，指示當前資料是屬於什麼 channel 的
Error: 錯誤信號，可以用來傳遞錯誤指示，表示當前信號的一個狀態。這個信號可以靈活運用，並不是一定要用來傳遞錯誤哦。
Empty: 在 EOP 的時候，指示資料中的哪幾個 Byte 是有效資料，很像 Mask。

其實 Avalon 介面並不是什麼非常複雜的東西，但是應用到具體的模組中，就需要大家自己靈活掌握，發揮創意。對他們更深入的瞭解對於更好地把握傳輸，控制是非常重要的，對於系統的理順和增強也是很有意思的。當然還是不太需要拘泥於具體的形式。