SOC整合實例—網路位址轉換硬體與AMBA整合實作

鄭伊君 曾國坤 林盈達

國立交通大學資訊科學系

新竹市大學路1001 號

TEL:(03) 5712121 EXT. 56667

Email: yjzheng@cis.nctu.edu.tw; kktseng@cis.nctu.edu.tw;
ydlin@cis.nctu.edu.tw

1.摘要

系統單晶片(System on Chip，SoC)是針對特殊的目的所設計的晶片。當其成為目前的一種趨勢時，如何設計成為很重要的研究議題。本篇文章針對目前 SoC 發展中最熱門的 AMBA(Advanced Microcontroller Bus Architecture)整合平台做一番介紹，並以實例解釋將一顆我們研發的網路位置轉換硬體(Network Address and Port Translator, NAPT)的 core 加在 AMBA 平台上，以加快網路處理速度。我們希望藉此過程可以讓讀者了解如何設計一個 AMBA 上的 Master，與整合 SoC 元作的部分訊號轉換流程。由於本實作目前還在進行，目前已完成軟體系統環境的架設和初步的轉換介面設計，尚未完成的其他 NAPT 為了 AMBA 而做的修訂設計，所以本文只介紹目前完成的設計，修訂設計與硬體驗證的部分則待日後完成。

關鍵字： AMBA SoC ARM NAPT

2.簡介

在現今這個半導體產業快速發展的時代，製程進步的速度與日精進，積體電路的整合密度及運算頻率皆如前Intel 總裁摩爾所提出的每18個月加倍的速度（Moore Law）在持續的高速成長。如今，市面上每顆IC裡的邏輯閘數目，動輒幾十百萬顆。如此進步

的製程技術，使得SoC (System on chip)早已不再是夢想，而成為現今半導體產業相繼分食的大餅。根據Dataquest統計，SoC在2001年全球整體產值約198億美元，僅占全球半導體產值12.9%，但到2005年，可望達到417億美元，占整體半導體比重高達21.1%。

既然對半導體業有這麼大的影響，那到底什麼是SoC？簡單來說，就是把過去各擔負不同單一功能的晶片，整合在一顆晶片裡面。過去的一個系統中或許有來自英特爾的微處理器(CPU)執行主要運算功能、有來自威盛的晶片組，負責傳輸訊號到各個介面，還會有nVidia的繪圖晶片，讓動畫功能能流暢執行，另外可能會有瑞昱設計的網路晶片，負責網路的連接，當然，創惟做的USB控制晶片、立錡做的電源管理晶片、驊訊設計的音效晶片等等，林林總總算起來，一片主機板上，至少有8種以上的晶片散布。因此，相當耗費主機板的面積，跟運作時產生的熱能，而SoC也是所謂系統單晶片的概念，就是儘量把這麼多的東西，整合在一顆晶片裡面來完成。

雖然由於IC製程的進步使得系統單一晶片成為一種趨勢，但在SoC的設計上也有其困難之處，例如系統的複雜度便是其中之一。一個SoC的設計包含了處理器、DSP、軟體介面、作業系統、匯流排、數位或類比模組等等。面對這麼複雜的系統設計又必須縮短開發的週期，才能增加市場的競爭力，SOC的設計也出現了許多方式，平台式的設計是其中的主流之一。

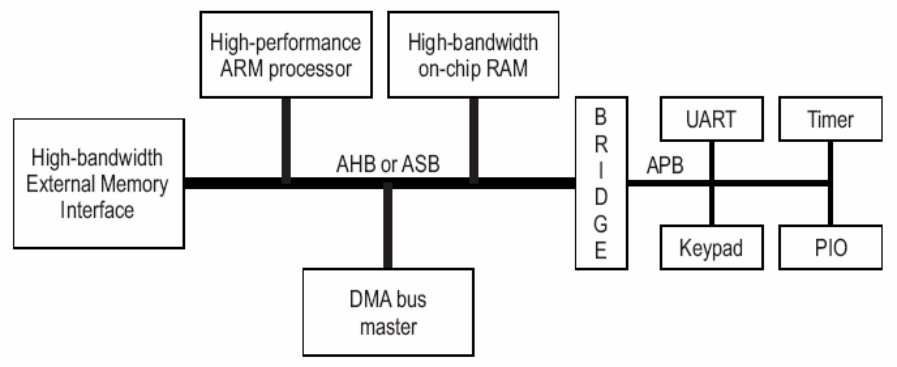
所謂平台式設計就是一個完整的匯流排架構以及一些IP 模組所組合而成，而這個架構也必須考量到新增IP 模組的方便性。目前SoC的處理器市場裡，ARM 表現的相當出色，因SoC強調的是其低成本低電量的優點，而ARM 一開始即是以低耗電量為主要的訴求，自然的成為SoC平台中的寵兒。根據ARM在2001年一月所提出的報告中指出，ARM2000年在32位元嵌入式微處理器的市場佔有率高達76.8% ，由此可見ARM平台熱門的程度。

AMBA(Advanced Microcontroller Bus Architecture)為ARM所發表之晶片匯流排(On-Chip Bus)技術，是目前全球應用最廣泛之SOC整合平台。舉凡遵循AMBA的spec所設計的IP 皆可經由ARM processor來幫助其運作，大大縮短了SOC的開發週期，而且在複

雜的設計中，大量的使用已驗證過的模組(ARM和AMBA)，可有效的偵測到系統遇到的瓶頸和問題，這便是平台式設計的精神。

在這一份報告裡，將以實例介紹平台式設計的方式做整合；將一顆由實驗室曾學長所研發的NAPT core 加到AMBA上，考量Master所應有的訊號與NAPT傳送的模式來設計轉換的NAPT主動存取記憶體介面，藉以提供一個獨立的NAT系統。在介紹設計之前我們會先介紹待轉換的二邊訊號介面，然後少介紹目前簡單介面設計的各個部分，最後是未來預定要完成的修訂設計。

2.1 AMBA簡介



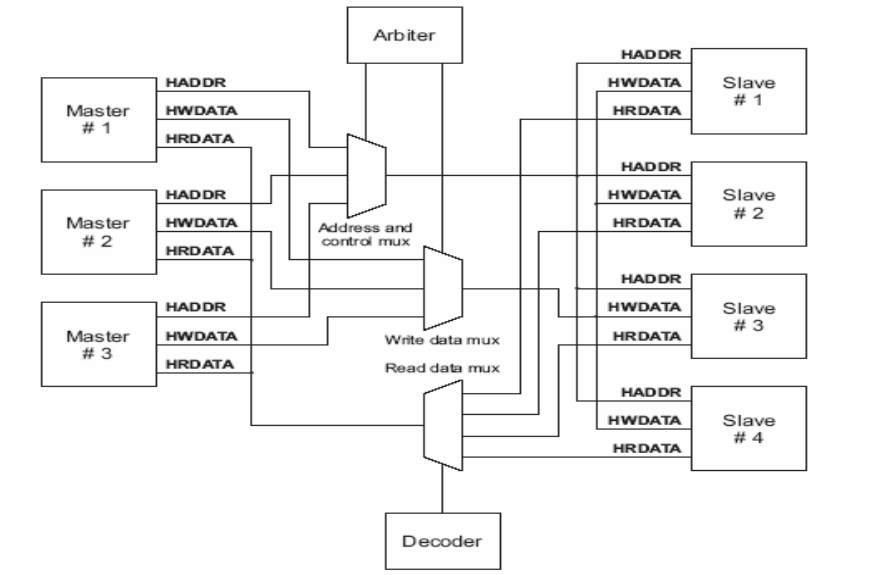
（圖一）AMBA 主架構圖

如圖一所示 AMBA 可細分為 AHB(Advanced High-performance Bus)和 APB(Advanced Peripheral Bus)二個部分。如字面上所示，AHB 有較高速的 performance，適合需要高速運作的 module。而 APB 則適合低耗電量，介面較為簡單的 module 使用。因為 NAPT 要整合入 AHB 中，所以以下我們將針對 AHB 部分作介紹。

■ AHB 多 Master 與 Slave 之間的關係

現在我們針對AHB的部分 (圖一的左側)做介紹，由圖一所示，在AHB上有許多的module，主要可分為主動要求使用Bus的Master和被動的slave。AHB可以多個Master共用Bus，如ARM 處理器或DMA controller。他們之間連結的方式，如圖二所示是以集中多工互連(central multiplexer interconnection)的方式設計。一開始Master會先向

arbiter發出使用Bus的要求，同時放出位址和表現傳輸類型的控制訊號。當arbiter收到若干個Mater的要求訊號來，會依內定的優先順序來決定將Bus使用權交給誰，一經決定，Arbiter會將該Master所送出的訊號經由多工器Route到所有的slave上。再由Decoder經Master送出來的訊號來判定那一個slave該enable。



(圖二)AMBA上以多工器連結Master和slave的情形

■ **AHB 訊號**

在介紹 AHB 的傳輸 cycle 之前，我們先來解釋一些接下來設計時會提到的訊號的作用、含義和表示情形，方便讀者對於之後 cycle 介紹的了解。

訊號名稱	作用	來源
HCLK	匯流排時脈	clock source
HRESETn	重新啟動	reset controller
HADDR[31:0]	位址	master
HTRANS[1:0]	傳輸型態	master
HWRITE	傳輸方向	master
HSIZE[2:0]	傳送大小	master

HBURST[2:0]	burst 型態	master
HPROT[3:0]	保護控制	master
HWDATA[31:0]	寫入資料匯流排	master
HSELx	slave 選擇訊號	decoder
HRDATA[31:0]	讀出資料匯流排	slave
HREADY	傳輸完成表示訊號	slave
HRESP[1:0]	傳輸回應	slave

(表一)AMBA訊號表

除了HTrans和HResp是讓AMBA上的Master與slave傳達訊息所用，表一中大部分的訊號都如一般的BUS訊號，我們便不再贅述，有興趣可參考AMBA spec. v2。底下我們介紹二者的訊號內容表示後，以一個cycle圖舉個例說明他們溝通的方式：

HTrans

00: IDLE 當Master沒有data需要傳送卻取得Bus使用權時所使用狀態

01: BUSY 讓Master在burst中間插入Idle狀態，通常是Master來不及做出下一個burst所需的訊號時使用

10: NONSEQ 表示接下來所傳的address與之前的並無關,也許是第一個Burst的address

11: SEQ 表示接下來所傳的address與之前的有關，通常是Burst的第二以後的access

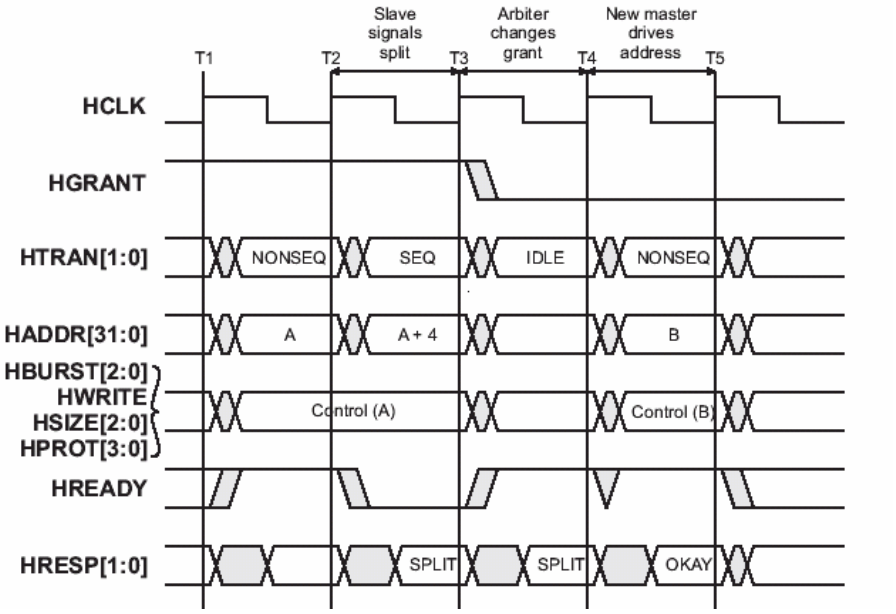
HResp

00: OKAY 傳輸完成

01: ERROR 發生錯誤

10: RETRY 傳送未完成，等待下一次的Grant

11: SPLIT 傳送未完成，暫時 Block 不傳輸

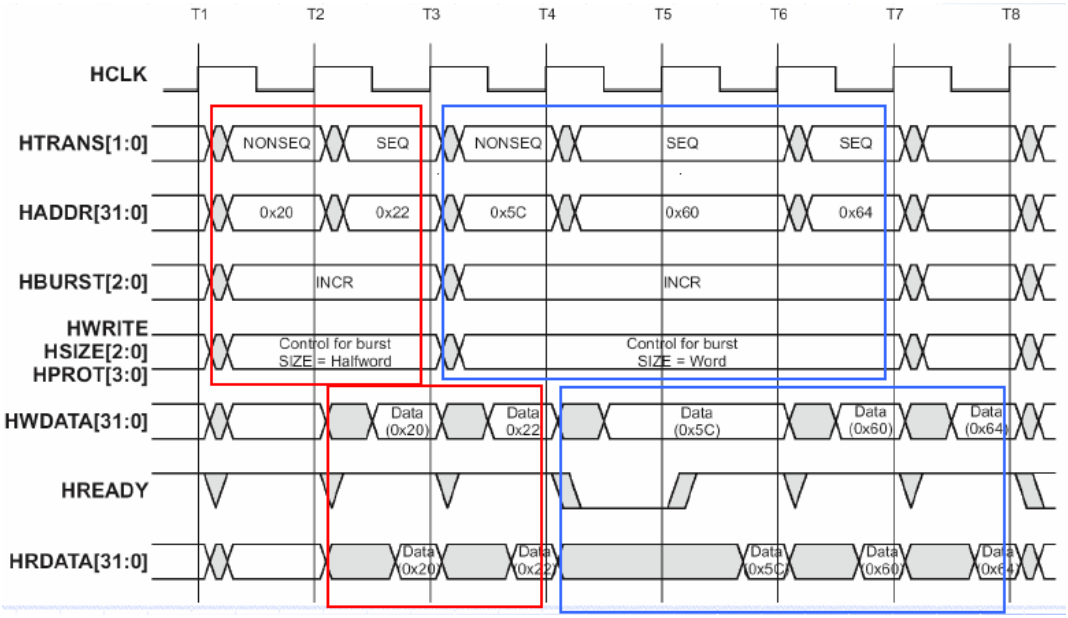


(圖三)Bus 在 split 回應底下 handover 的情形

圖三是Master A在slave回應split時交出Bus使用權，再由Master B搶到Bus使用權的cycle示意圖，T1時間點放出address，T2.T3 cycle 的後面回對這筆address的回應：2cycle的split。在第一個回應的cycle末(T3上)Master 需偵測到 split 而插入 idle 的狀態，來清除原來會出現的A+4的控制訊號。Idle cycle都是一個cycle完成。下一個Master(B)必會在IDLE之後放出address與控制訊號。

- **Pipeline訊號傳輸模式**

AHB為了達到其高performance採用pipeline的傳輸模式，Data在Address的下一個cycle出來的同時，下一筆傳輸資料的Address也在此時同時送出，如圖四所示。



（圖四）AMBA傳輸訊號-解釋Pipeline訊號傳輸模式

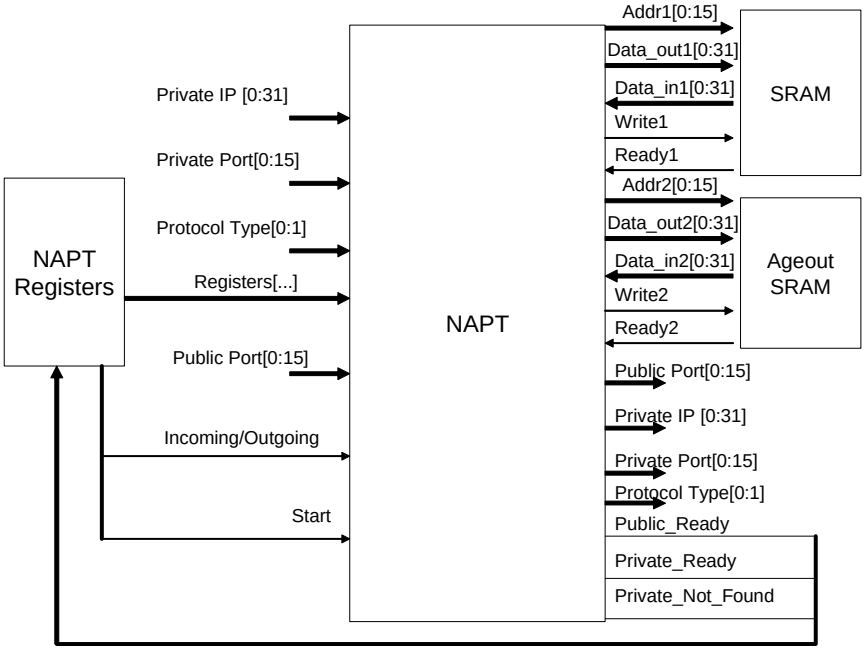
其中以 Hready 來表示是否完成傳輸，若 Hready 顯示為未完成傳輸(低電位)，則 Master 必須 hold 住目前的 address，讓 Slave 可以在下一個 cycle 完成傳輸。

2.2 NAPT 簡介

介紹完 AMBA 後，接下來要介紹的是我們要整合入 AMBA 的這顆 softcore－NAPT。

NAPT為一種虛擬的IP位址轉換功能， 其主要功能是， 可以讓多台的內部電腦共用一個Public IP位址， 每一個內部的電腦連線(Connections)會有一個唯一的Public Port來作區別。NAPT若使用Public Port編號來區別內部電腦的連線，在使用一個public IP 位址的情況下， 最多可以建立 2^{16} 個連線。 若每個private IP 位址可以有 32 個連線， 則最多可同時支援 2048 個IP 位址。 因此此方法為解決IP位址不足的一個方式。而我們所要加入的這顆NAPT是一顆提供新的private ip和public port對應的演算法的NAPT core，不同於原來的方法是直接利用Private IP的最後一個位元組，來當作Public Port的第一個位元組，若是內部同一台電腦的不同連線，則依連線建立的時間順序，來決定其Public Port的第二個位元組，去得到所要轉換的Public Port編號。而是提供一

個較有彈性、較適合硬體實現加速的方法。以下將暫略其內部設計，解釋其外部訊號的傳輸模式。



(圖五)NAPT 主要架構圖

圖五為 NAPT 的大概示意圖，NAPT 主要由一個包含主要控制的 FSM(Finite State Machine)Main Block 與二塊記憶體所組成。二塊 SRAM 各自有不同的 Bus 來達到同時 Access 的功能。其中，Ageout SRAM 是算 Ageout 的 IP 或 port，判定其已斷線，進而將它刪除，由 counter 所 maintain，定期的去將 counter 的值減一，當其為 0 時即將其刪除。由於需定期的更新 counter 的內容，所以會有很大量、頻繁的 access 次數。

■ **NAPT 外接 RAM 部分的訊號**

訊號名稱	作用	方向
clk	時脈	輸入
reset	重設	輸入
addr_napt2table	從 NAPT 到 memory 的 address	輸出
data_napt2table	從 NAPT 到 memory 的 data	輸出
data_table2napt	從 memory 到 NAPT 的 data	輸入

write_en_napt2table	NAPT 對 memory 做寫	輸出
read_en_napt2table	NAPT 對 memory 做讀	輸出
napt2table_ready	memory 動作完成可輸出	輸入

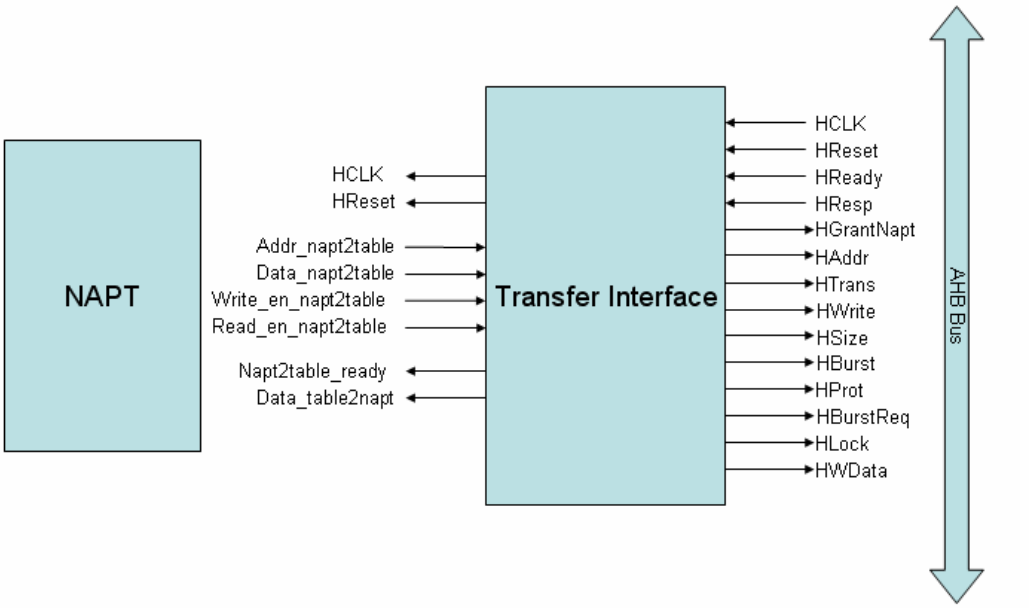
(表二)NAPT 外接 RAM 部分的 I/O

因為目前我們要做的部分是讓 NAPT 可以對 AMBA 上的記憶體作讀寫,所以目前我們只介紹 NAPT 做讀寫的訊號部分。當 NAPT 要做讀,只要先將 addrs 放到 addr_napt2table 上,再將 read_en_napt2table 拉高則假定記憶體開始做讀,直到 napt2table_ready 拉高則表示記憶體動作完成。才會進行下一個動作。其中 address 和二排 data bus 都是固定 32bit 且 NAPT 並沒有針對 AMBA 設計 Burst 的傳輸模式所以每筆 address 都是獨立的。

3.NAPT 整合入 AMBA 平台的訊號轉換設計

介紹完會用到的訊號傳輸模式,可以了解到 NAPT 與 AMBA 上除了訊號腳的不同之外,傳輸模式更是相當大的不同,AMBA 的 pipeline 傳輸模式與 NAPT 的 handshaking 之間的等待 cycle 的處理是轉換訊號主要設計及改善的地方,如何正確的轉換進而配合 AMBA 為了提高效能所做的訊號傳輸設計在未來的修訂設計要努力的目標。

接下來圖六就是參考 EASY(Example AMBA SYstem)所設計的一個簡單的轉換介面,其右接 AHB 左接 NAPT 的存取訊號。

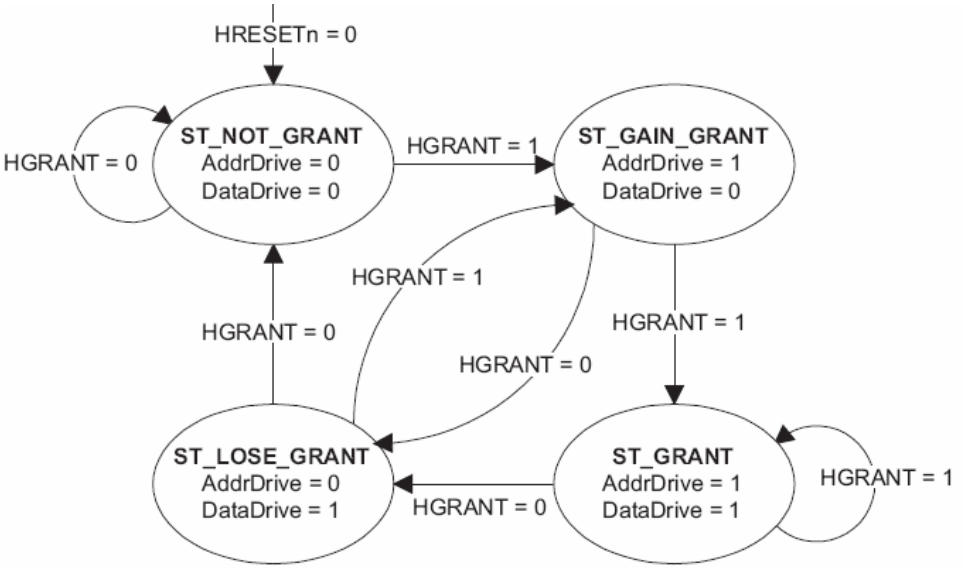


(圖六)介面架構圖

因為 NAPT 主要動作為向記憶體查詢資料，ip 和 port 的轉換與收送封包，皆是透過記憶體來動作以 Master 的角色做轉換是比較恰當的。既然是 Master 就少不了的必須要設計下列幾個模組：

■ Granted 狀態機

如前 AHB 所介紹，Master 需要向 Arbiter 要求使用 Bus 得到 Grant 後才能開始使用 Bus 傳送資料。圖七即為參考 EASY 中 TIC 的狀態機示意圖：其中我們也用 AddrDrive 和 DataDrive 為其內部的控制變數，只有在 HReady 是 high 時此狀態機才會變換否則維持同狀態。



(圖七)Grant 狀態圖

- AHB address 產生

因為有時 Master 需照著 Slave 反應傳輸的結果來決定下一個送出去的 address。如果是標準的 read/write cycle 而且都來得及的話就直接用下一個 address。如果 slave 的回應是 split 或 retry 就要重送上一筆 address，所以至少要有二個 address register。加上要從 NAPT 抓 address 下來，所以需再準備一個 register 去 latch。因為此顆 NAPT 並不支援 pileline 傳輸的作法，所以不考慮設計 address 由 increment 產生的功能。

- 資料讀取控制

要確定此時的 read_en_napt2table 為致能狀態

- 控制訊號產生部分

訊號名稱	作用	值
HTRANS[1:0]	傳輸型態	Nonsequential 後一直接 IDLE 直到 HReady high 的下一個 cycle 如果遇到 response 是 split 也要插入 IDLE state 來清掉控制訊號
HWRITE	傳輸方向	NAPT 的 write_en_napt2table

HSIZE[2:0]	傳送大小	32bit(010)
HBURST[2:0]	burst 型態	single
HPROT[3:0]	保護控制	xxxx ,通常 slave 都沒用到所以不給值

(表三)控制訊號給定內容

HTrans 我們做了三個假設：

1.default 都是 idle

2.因為 NAPT 不做 pipeline read 所以送 address 出去之後都插 IDLE 等其完成

3.假定 NAPT 皆來得及產生下一筆 address 否則要考慮 busy

HWrite 此處設定雖為 NAPT 的 write_en_napt2table 但因為 low 表示 read，所以必須確定 write_en_napt2table＝～read_en_napt2table

- split與retry偵測 假設在傳輸過程當中，NAPT失去Bus使用權，則splitRetry要一直被hold，直到重新取得使用權，完成傳輸為止。

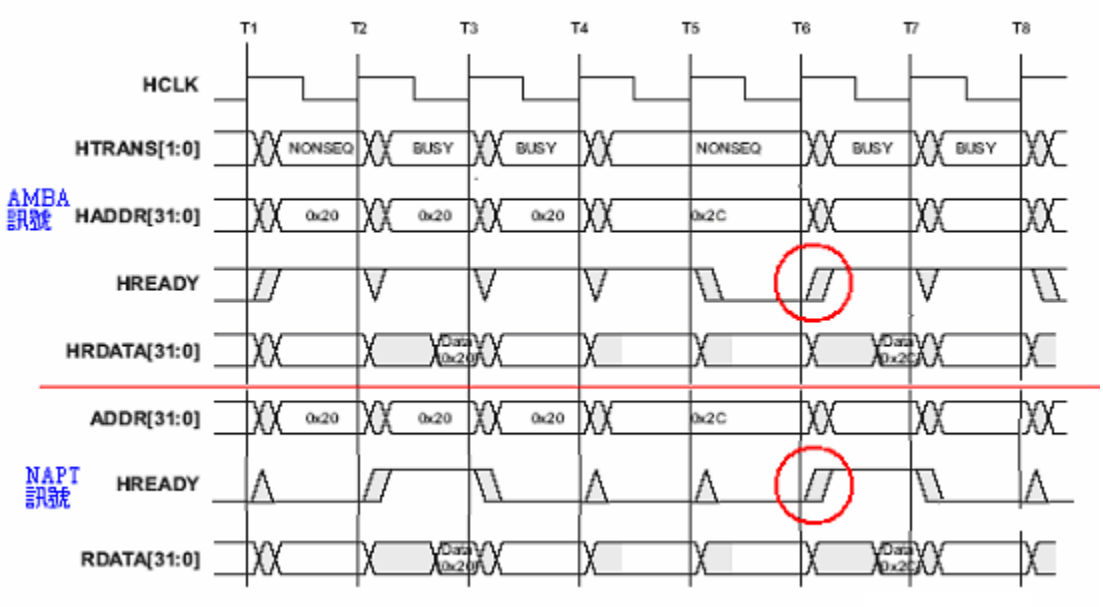
以上討論清楚 Master 對 AHB 的訊號給定及各個 Block 的大概架構內容，接下來解釋一下形成的 NAPT 訊號收送的流程：

1. NAPT 送出 read_en_napt2table high 和 address 訊號。
2. TIC 看到 read 訊號會當成 request 開始對 arbitor 發出 request,因為不一定會馬上取得 grant 所以先，內部開始跑狀態機，處理 grant 的訊號與 address 和 data 的 drive。
3. 取得 Grant 之後，TIC 將 addr 導到 AMBA 上，開始一般的 read/write cycle。
4. NAPT 會等到 ready 上來才會送出下一個 address，所以在第二個 grant 送的 Address 仍是之前的 address 接下來會插 IDLE state 使 AHB 上不依原來的 address 去動作。

5. 因為 AMBA 上的 Hready 是表示 data 可以在這個 cycle 被完成(讀出或寫入)，告訴 Master 可以送出下一個 address，但是 NAPT 的 ready 一但放出來 data 會馬上被 latch，所以傳給 NAPT 的 ready 必須晚 AMBA 上的 Hready 一個 cycle。

6. 當 ready 致能，NAPT latch 住 data 後，下一個 cycle 就會放出下一筆資料的 address。如此重覆的存取記憶體

圖八是 cycle 設計圖:



(圖八)cycle 設計圖

4. 未來工作

NAPT 目前尚在設計階段，計畫完成度是將其以 FPGA 做體驗證。設計部分已完成大部分，軟體模擬所使用的系統環境是 ARM 提供的 EASY。已加上之前所介紹的大部分設計，尚希望完成以下的設計：

1. NAPT 存取 ip /port 的 slave 設計：目前假定是由系統上主要的 processor 將封包 decode 出 ip/port 放到一個我們設計的暫存器檔(Register File)，當然此暫存器

檔會針對 AMBA 的傳輸模式做設計。善加利用 Burst 來做加速。

2. 加一個 module 作 Buffer 使得 NAPT 可以使用到 AHB pipeline 設計的優點來增進 NAPT 的速度。或直接改 NAPT 的 address 和 data 不要同時放出而是差一個 cycle，讓 data 晚一個 cycle 放出或讀入。

3. Address 其實不用每次都抓 32bit 因為其實在 NAPT 的演算法中，並非 32bit 全部都使用，反而大多使用最後的 8bit 加上 AMBA 有支援不同的 size 存取，所以可以從 NAPT 改此處的設定，降低耗電量。

預計十一月底要加上上三點設計的軟體模擬測試，十二月開始進行硬體合成，修訂 FPGA 合成器無法合成的語法。一月中 study 如何看合成後的報表做最後的效能提升修正，以 Benchmark 測試的部分為主要修正的目標。以上是暫定完成的目標，以後尚可進行的工作，可將這個已轉換成支援 AMBA 的硬體和 processor 一起真正的收送封包作 NAT 的測試。

5.參考資料

[1]NAPT IP core Concept from Kuo-Kun

[2] **AMBA** Specification 2.0.

[3] ARM **Technical** Training Course 2003

[4] **AMBA University Kit Technical Reference Manual**.

[5] **AMBA University Kit** User Guide.

[6] The Example **AMBA** System **Technical Reference Manual**.