運用過濾器方法之低耗電分支目標緩衝器的設計

Design of Low-Power Branch Target Buffer Using Filter Scheme

馬永昌
國立台灣海洋大學資訊工程學系
Email: ycmaa@mail.ntou.edu.tw

李冠侖
國立台灣海洋大學資訊工程學系
Email: M96570052@mail.ntou.edu.tw

郭書銘
國立台灣海洋大學資訊工程學系
Email: M96570033@mail.ntou.edu.tw

***摘要*—本篇論文中，我們將快取記憶體過濾器的概念應用到處理器中分支預測器(Branch Predictor)之分支目標緩衝器(Branch Target Buffer)設計。使用過濾器概念可以減少分支目標緩衝器的存取動作以降低動態耗電，本篇論文提出分支目標緩衝器的設計方法，不僅可以維持分支預測的準確度與處理器管線架構高效能運作，並且達到省電的效果。**

我們使用內容可定址記憶體(CAM)來設計過濾器，並且使用 HSPICE、CACTI 等工具來確認過濾器電路不影響處理器管線中指令擷取部分之關鍵電路延遲。根據 SimpleScalar/Wattch 與 SPEC2K 等工具為基礎的實驗結果顯示，在不犧牲分支預測準確度與處理器效能的前提下，分支目標緩衝器動作減少達到 85%，分支預測單元省電達 16% - 55%，分支目標緩衝器省電達 18% - 75%。

***關鍵詞*—低耗電、分支目標緩衝器、過濾器方法、哨兵表**

***Abstract*—In this paper we apply Sentry Tag based filter scheme to the design of branch target buffer (BTB) of branch predictor in modern processors. The filter scheme filtrates unnecessary accesses of branch target buffer to reduce dynamic power consumption. The proposed scheme not only maintains high branch prediction accuracy and thus high pipeline utilization for processors, but also attains considerable power saving.**

We use Content-Addressable Memory (CAM) to design the filter scheme and utilize HSPICE and CACTI tools to make sure the proposed scheme's critical path delay of processor instruction fetch of pipeline is not affected. Based on SimpleScalar/Wattch simulators and SPEC2K benchmarks, we show that our scheme can filter up to 85% of branch target buffer accesses, thus reducing the power consumption for branch prediction unit by 16% - 55%, (the power consumption for branch target buffer by 18% - 75%) without compromising prediction accuracy and processor performance.

***Index Terms*—Low Power, Branch Target Buffer, Filter Scheme, Sentry Table**

一、緒論

(一) 研究動機與目的

近代處理器為追求高效能，均採用高度管線化(deeply pipelined)技術設計處理器。在程式控制流程中，分支指令對於管線化架構的效率有甚大的影響，因此多數高效能處理器均使用動態分支預測技術，且運用分支目標緩衝器來提供有效率的分支處理。

處理器在執行分支指令時，會參考分支目標緩衝器中所記錄的位址，做為程式計數器下一次擷取指令的參考位址。每當執行到分支指令就會查詢一次分支目標緩衝器，若能減少分支目標緩衝器查詢動作，即可降低分支目標緩衝器的動態耗電。

在低耗電的研究當中，許多學者研究的方向大致上分為二個部份：動態耗電(Dynamic/Active power) [1][3][8][21][25][26]以及靜態耗電(Static/Leakage power) [15][20]。在降低動態耗電的相關研究中，許多學者提出減少電路的動作次數 [1][3][8][21]，或者使用過濾器的方法[25][26]，以達到省電的目的。而在降低靜態耗電的相關研究中，有學者提出完全斷電策略[15][20]，當某一部分電路經過一定時間都沒有被使用到，則可以將電路斷電進入沉睡模式，以達到省電效果，當有需要用到此電路時，再將其喚醒。

近年來大部分學者藉由改變電路的架構，或者增加部份電路，過濾分支目標緩衝器的非必要存取動作(Unnecessary Activities)，以達到省

電效果 [1][3][8][21]。其中有學者提出的方法雖然可以達到省電效果，但是會降低分支預測器的準確度以及最長路徑延遲 [8][21]，所以我們想設計一種方法，既不影響分支預測器的準確度，而且又能達到省電效果。我們使用 SimpleScalar [22] 以及 Wattch [9] 模擬一顆 Alpha-21264 [19]處理器，我們發現其中分支目標緩衝器的耗電又佔整體分支預測單元的絕大部分，如圖 1 所示。B-256 即表示分支目標緩衝器為四路，每路有 256 筆內容，採用 Bimodal [13] 分支預測策略，耗電佔整體分支預測單元大約 85%。

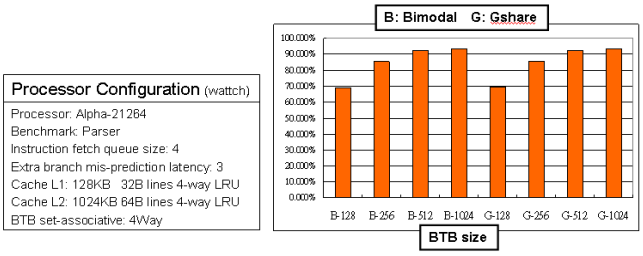

圖 1　分支目標緩衝器耗電在整體分支預測單元之比例

本篇論文應用過濾器方法(Filter Scheme)，在分支目標緩衝器過濾非必要存取動作，節省動態耗電，並進一步分析過濾器結構影響耗電的相關參數，以便進行低耗電分支目標緩衝器之設計。

(二) 研究貢獻

本論文的貢獻主要有以下幾點：

- 減少處理器的耗電：

利用過濾器的方法，減少分支目標緩衝器非必要的存取動作，以達到節省電能的效果。

- 保持處理器效能：

在電路上我們雖然增加額外的過濾器電路，但是並不會影響到整體處理器的效能，因為經過驗證最長路徑延遲(Critical Path Delay)仍然小於指令快取記憶體(I-Cache Memory)的存取時間，不會造成時脈週期延長的後遺症，所以可以保持處理器效能。

- 低硬體成本：

使用過濾器方法所需額外硬體並不多，額外硬體(電晶體數)佔整體分支目標緩衝器約 5% 以下。

(三) 論文內容簡述

本篇論文的其餘內容安排如下。第二節會先介紹相關背景知識與文獻探討，包括分支預測技術(Branch Prediction Techniques)。第三節介紹過濾器方法(Filter Scheme)的原理、硬體電路架構。第四節為模擬實驗結果與討論。最後，第五節提出本篇論文之結論。

二、相關背景

在本節會先介紹分支預測單元(Branch Prediction Unit)架構與功能，包含分支預測器(Branch Predictor)以及分支目標緩衝器(Branch Target Buffer)。

(一) 分支預測單元(Branch Prediction Unit)

分支預測單元位於處理器管線的前段部份，當分支位址(Branch Address)進到分支預測單元，會分成二部份平行處理，一部份是分支方向預測器(Direction Predictor)，另外一部份是分支目標緩衝器(Branch Target Buffer)，如圖 2 所示。

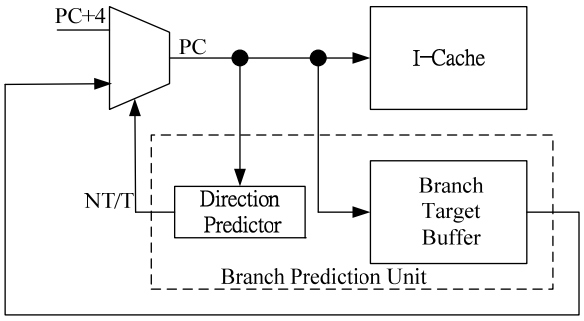

圖 2　分支預測單元(Branch Prediction Unit)

由於高效能處理器多採用管線化設計，分支預測與處理分別在管線化架構的指令擷取階

段(IF)、指令解碼階段(ID)以及指令執行階段(EX)三個層級進行，如圖 3 所示。在指令擷取階段時，判斷分支位址是否能在分支目標緩衝器搜尋命中；若是，則會輸出分支目標位址給程式計數器(PC)，當作下一個指令的位址；反之，若搜尋失誤，則會在指令解碼階段將指令解碼判斷是否為分支指令，若是分支指令，即會在指令執行階段時，將分支指令的位址寫入到分支目標緩衝器。

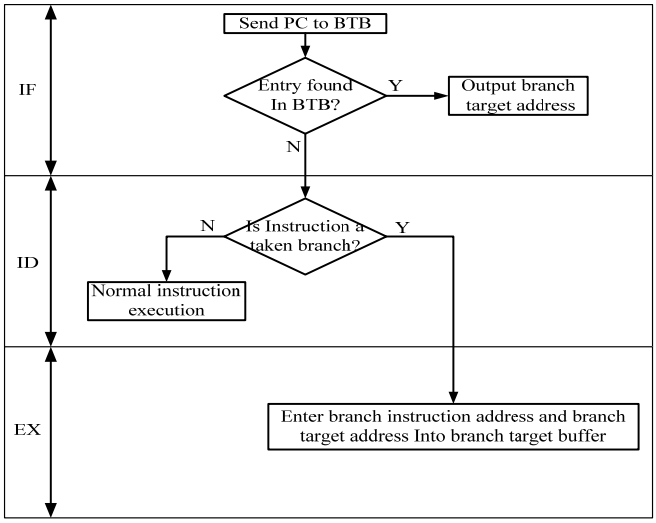

圖 3　分支預測單元在管線化設計中的層級

(二) 方向預測器(Direction Predictor)

分支預測主要的功能就是預測分支指令是否要跳躍(Taken or Not Taken)，主要分支預測策略有靜態(Static)以及使用硬體分支預測器的動態(Dynamic)方式，如：Bimodal、Two-Level [23]、GShare [24]、Combining Branch Predictors [24]…等等。其中以 Yeh [23]與 McFarling [24]分別提出 Two-level 以及 Gshare 和 Combining Branch Predictors 的方法之後，對於整個分支預測器領域有很大的影響，大大提升了分支預測器的預測準確率。

靜態分支預測技術 [14]是由程式設計師或系統軟體(如編譯器)來決定分支是否跳躍，例如規定所有分支指令跳躍(或不跳躍)，或者依照分支指令的種類來決定，如對迴圈控制指令預測皆為跳躍。

動態預測技術使用硬體分支預測器來預測分支指令是否會跳躍，比較具代表性的分支預測器包含 Gselect [24]、Agree [12]、Bi-Mode [6]、Skewed [17]、Filter [18]、YAGS [4]以及使用類神經網路(Neural networks)/感知機(Perceptron)的分支預測器 [7][10][16]。

(三) 分支目標緩衝器(Branch Target Buffer)

分支目標緩衝器(Branch Target Buffer)位於處理器中，分支目標緩衝器會有一個部分用來存放由程式計數器(Program Couner)抓取到的指令位址的數值，再將跳躍分支(taken branch)指令的目標位置儲存起來，如圖 4 所示。

當分支預測器判斷分支指令是跳躍的時候，即可查詢分支目標緩衝器上目標位址的數值，而不用等到處理器管線處理的執行階段才經由運算分支指令位址的結果得到目標位址的數值。由於可以提前預知目標位址的數值，所以可以減少分支指令所造成的效能負擔，並且可以提高程式處理的效能。

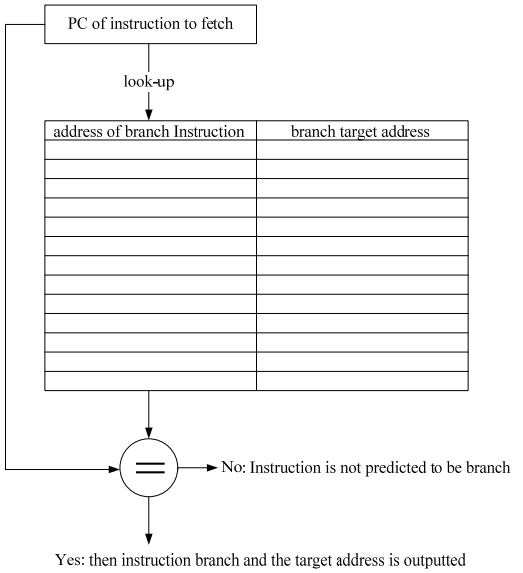

圖 4　分支目標緩衝器(Branch Target Buffer)

三、過濾器方法

在本節一開始介紹傳統分支目標緩衝器構造與動作流程，以及定義非必要的存取動作。再來介紹使用過濾器方法的動機，分析評估所

能改善的過濾率與過濾器方法使用在分支目標緩衝器的原理，以及過濾器所需要的額外硬體電路之設計。

(一) 傳統四路集合關聯式分支目標緩衝器

圖 5 為一個傳統的四路集合關聯式分支目標緩衝器(4-way set-associative BTB)，當程式計數器送出一個分支指令時，會將其分成標籤(Tag)與索引(Index)兩個部份，索引經過一個解碼器電路即可辨認此分支指令會落入那一個集合(Set)中，再將此集合中的四個標籤與分支指令的標籤做平行比對，比對完的結果再送到多工器(MUX)，多工器即可判定是否命中(Hit)或是失誤(Miss)，若是命中即送出目標位址(Target Address)。

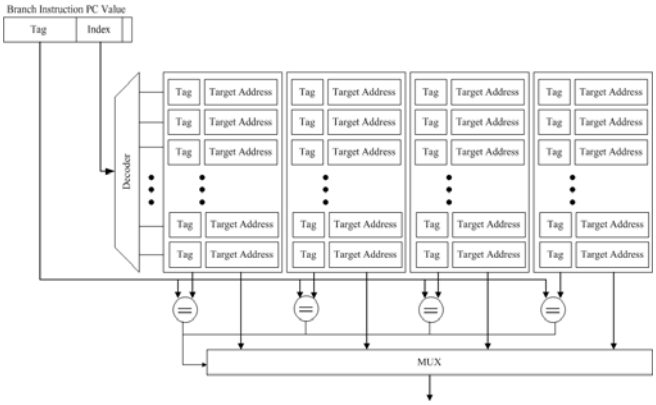

圖 5　傳統四路集合關聯式分支目標緩衝器

由於上述中，我們提到傳統四路集合關聯式分支目標緩衝器的構造，現在來分析它的特性。每當程式計數器傳送一個分支位址(Branch Address)時，在分支目標緩衝器都要同時到每一個路(Way)中取出標籤來比對。在這裡我們定義非必要存取動作，如果以命中來說，則會有三個路的動作是在做非必要的存取動作；倘若以失誤來說，則是四個路的動作是非必要的存取動作，而這些非必要的存取動作將會導致耗電。我們觀察到在分支目標緩衝器中的非必要存取動作會導致耗電，所以我們利用 [25]的過濾器方法，來改善非必要存取動作的發生次數，進一步達到省電的效果。

(二) 過濾器方法使用在分支目標緩衝器的原理

圖 6 為一個傳統分支位址，使用過濾器方法來設計一個新的分支目標緩衝器硬體架構。我們改變傳統分支位址的表示方式，在標籤這一個欄位中移出部份的位元(bit)並且給予一個新的名稱哨兵位元(Sentry bit) [25]。圖 7 是使用哨兵位元之後的分支位址結構，其中 S 代表的即是哨兵位元，圖 8 為使用哨兵位元的分支目標緩衝器架構。

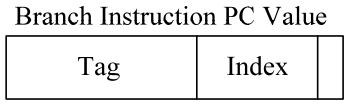

圖 6　傳統分支位址結構

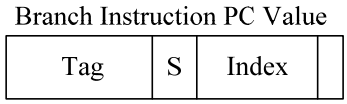

圖 7　使用哨兵位元之後的分支位址結構

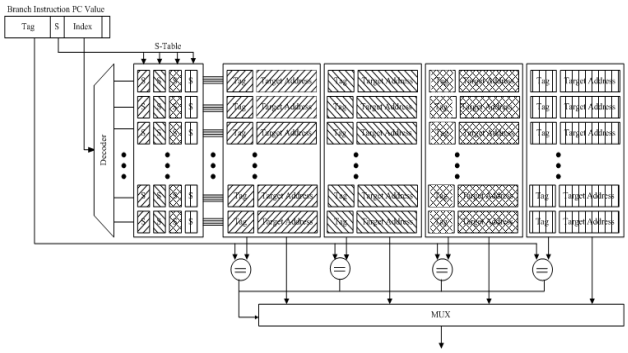

圖 8　使用哨兵位元之後分支目標緩衝器架構

過濾器方法的精神是過濾掉在每一個路中的非必要存取動作，現在介紹哨兵位元在過濾器方法中的功能與概念。哨兵位元是由標籤裡面擷取的一部份，所以哨兵位元是標籤的一個子集合(Subset)。當一個分支位址進到分支目標緩衝器的時候，我們設計讓哨兵位元這一部份的標籤與分支位址相對應的位元先做比對的動作。假如在哨兵位元這一部份比對命中，則表示整體標籤有可能會存在分支目標緩衝器中，所以會繼續比較剩餘標籤(即是不包含哨兵位元的位元)，再判定整體標籤是否會是命中或失

誤；倘若在哨兵位元這一部分發生失誤，則表示整體的標籤比對一定是失誤，故不需要再到分支目標緩衝器去比對剩餘的標籤是否相等，即表示可以過濾掉非必要的存取動作。

(三) 過濾率與平均存取動作的分析

在上一節中我們談到過濾器方法的概念，現在來分析過濾率與平均存取動作，由概念得知當哨兵位元的位元數取的越多，則過濾率會是越好，因為要比對的數目變多，排列組合的結果也跟著變多。例如：哨兵位元從標籤中取一個位元，當相對應分支位址的位元值是'1'，此時在哨兵位元中的值可能是'0'或是'1'，即表示過濾率大約是 50 %；倘若哨兵位元從標籤中取二個位元，當相對應分支位址的位元值為 "10"，此時在哨兵位元中的值可能是"00"、"01"、"10"和"11"其中的一種情況，即表示過濾率大約是 75 %。

平均存取動作的分析

但是在實際分支目標緩衝器過濾率的分析還要考慮到分支目標緩衝器的命中率(Hit Rate 或 HR)以及分支目標緩衝器關聯度是多少個路，下面使用機率統計方法來定義平均存取動作次數。

分支目標緩衝器命中時：

在一個 W 路分支目標緩衝器中，哨兵位元有 B 位元，當一個分支指令在分支目標緩衝器搜尋命中時，即是在整體標籤在其中一個路完全命中，也代表這一個路在哨兵表(Sentry table)[3]當中的 B 個位元也是命中的，所以基本上就會有一個路是需要動作的。而其他 W-1 路在哨兵表當中的 B 個位元可能命中也可能是失誤。由於哨兵位元擷取整體標籤的其中 B 位元，故有 2^B 種狀況，"00…00"、"00…01"、…、"11…10"和"11…11"，這 W-1 路的動作次數為 $\frac{1}{2^B}$，當分支目標緩衝器為命中時，定義動作次數為：

$$(1+(W-1)\times\frac{1}{2^B}) \qquad (1)$$

分支目標緩衝器失誤時：

在一個 W 路分支目標緩衝器中，哨兵位元為 B 位元，當一個分支指令在分支目標緩衝器搜尋失誤時，即是在整體標籤在其 W 路都無完全命中，也代表這 W 路在哨兵表當中的 B 個位元可能命中也可能是失誤。由於哨兵位元擷取整體標籤的其中 B 個位元，故有 2^B 種狀況，"00…00"、"00…01"、…、"11…10"和"11…11"，這 W 路的動作次數為 $\frac{1}{2^B}$，分支目標緩衝器失誤時，動作次數為：

$$W\times\frac{1}{2^B} \qquad (2)$$

利用(1)與(2)再加上命中率，我們可以定義出平均存取動作的次數，如下列：

$$W_{Ave}=HR\times(1+(W-1)\times\frac{1}{2^B})+(1-HR)\times W\times\frac{1}{2^B}$$

$$=HR+\frac{HR\times W}{2^B}-\frac{HR}{2^B}+\frac{W}{2^B}-\frac{HR\times W}{2^B}$$

$$=\frac{W}{2^B}+(1-\frac{1}{2^B})\times HR \qquad (3)$$

W_{Ave}：所有關聯度的平均存取動作
HR：命中率
W：分支目標緩衝器的關聯度
B：從標籤中取得的哨兵位元的位元數目

過濾率的分析

接下來要分析以哨兵表為過濾方法的分支目標緩衝器之過濾率，再這裡過濾率的定義是：

$$FR = \frac{W - W_{Ave}}{W} \quad (4)$$

FR：過濾率(Filter Rate)
W：傳統分支目標緩衝器的平均存取動作
WAve：使用過濾器方法分支目標緩衝器的平均存取動作

(四) 過濾器方法硬體電路設計

這一節開始討論過濾器方法所需要的硬體電路，由於在前二節有討論過傳統分支目標緩衝器架構以及動作流程，由圖 5 增加哨兵表之後的電路示意圖，如圖 8 所示。將原本的分支位址中的標籤分割出一小部份當作哨兵位元，把這些分割出來的哨兵位元收集起來成為哨兵表，當分支位址到來時，會先在哨兵表中做比對動作，比對結果如果是命中，則會繼續往右半部份相對應的分支目標緩衝器繼續做標籤比對；倘若結果是失誤，則相對應的分支目標緩衝器則是無動作。

哨兵表實際上是一個內容可定址記憶體(CAM)的架構，因為內容可定址記憶體有快速比對的功能，而且使用有規律性的內容可定址記憶體電路來實作較為輕鬆，如圖 9 所示。針對內容可定址記憶體電路的詳細分析以及模擬，我們會在第四章節的時候討論。

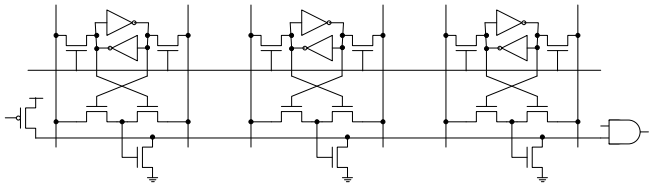

圖 9　CAM 電路的詳細架構

四、模擬實驗結果與討論

本節介紹本論文所提使用過濾器方法之分支目標緩衝器設計，模擬評估環境結果及討論實驗相關數據與成果。本篇論文使用 SimpleScalar [22] 以及 Wattch [9] 模擬 Alpha-21264 [19]架構，相關組態設定如表 1 所示。使用 SimpleScalar 所執行的標竿程式是 SPCE2000 [27]，每一組標竿程式皆執行五億個指令。在電路延遲的分析上則使用 CACTI [11] 與 HSPICE [2]，因為 CACTI 是針對於快取記體設計的分析軟體，快取記憶體與分支目標緩衝器的構造並不完全相同，而且在 CAM 的分析上也不能完全依賴 CACTI，所以我們選用 HSPICE。在功率方面的計算則是沿用 CACTI 以及 Wattch，Wattch 當中也包含了舊版的 CACTI 功率計算方式，所以我們也用較新的 CACTI 當作輔助來幫忙計算功率消耗。

表 1　相關組態參數設定

Processor Core	
Instruction Window	RUU=64, LSQ=32
Issue width	4 instructions per cycle
Functional Units	4 Int ALU, 1 Int mult/div
	2 FP ALU, 1 FP mult/div
Memory Hierarchy	
L1 D-cache size	64KB, 2-way, 32B block /
	32KB, 2-way, 32B block /
	16KB, 2-way, 32B block
L1 I-cache size	64KB, 4-way, 32B block /
	32KB, 4-way, 32B block /
	16KB, 4-way, 32B block
L1 latency	1 cycle
L2	Unified, 1MB, 4-way, LRU
	11 cycle latency
Memory latency	200 cycle
TLB Size	64-entry, 30 cycle miss penalty
Branch Predictor	
Branch target buffer	256-entry, 4-way /
	512-entry, 4-way /
	1024-entry, 4-way /
	2048-entry, 4-way /
	4096-entry, 4-way /
	512-entry, 8-way /
	1024-entry, 8-way /
	2048-entry, 8-way /
	4096-entry, 8-way
Return-address-stack	32-entry
Prediction Strategy	Gshare

(一) 對最長路徑的影響

本節主要探討使用過濾器方法所需額外的電路，以及哨兵表所造成的電路延遲效果，是否會影響到管線化中指令擷取所需的時脈週期數。圖 10 是管線化中指令擷取階段的電路，路徑 1 是指令快取記憶體的延遲路徑，路徑 2 是分支目標緩衝器加上哨兵表之後的電路延遲路徑，而接下來我們要比較兩路徑的延遲時間。

我們使用 CACTI 工具來計算指令快取記憶體與分支目標緩衝器的電路延遲時間，再利用 HSPICE 模擬哨兵表的延遲時間。哨兵表每一個路我們使用 3bit 的哨兵位元來儲存資料，所以會有 3 個 CAM 架構的 SRAM，如圖 11 所示。應用 3bit 的哨兵表是參考張延任教授 [26]之前分

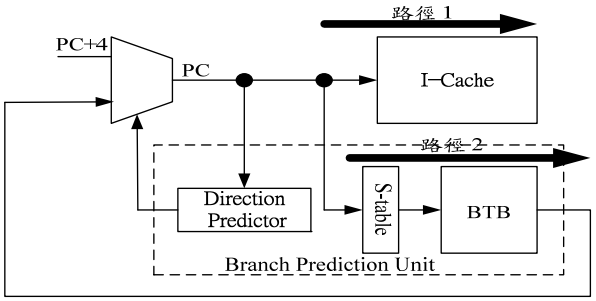

圖 10　管線化中指令擷取階段的電路

析的結果，由圖 12 可以觀察出，在大於 3bit 之後過濾率曲線斜率開始平緩，我們沿用其研究成果，所以我們選擇使用 3bit 當作哨兵表的位元數。

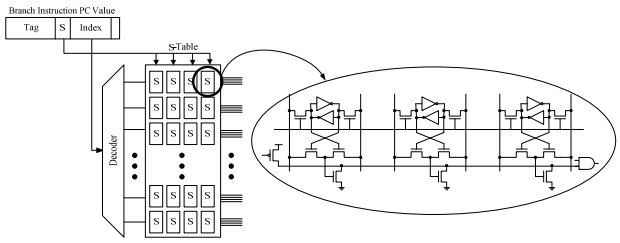

圖 11　哨兵表結構圖

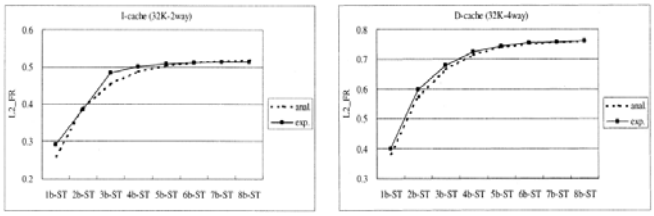

圖 12　哨兵表的位元數與過濾率之間關係圖

我們使用 Virtuoso 工具 [2]畫出哨兵位元電晶體階層的硬體電路圖，如圖 13 所示。再將電路圖轉換成 HSPICE 可以模擬的 SP 類型檔案。最後使用 HSPICE 模擬電晶體階層的硬體電路延遲時間，如圖 16 所示。我們發現哨兵表的延遲時間大約為 0.08 nS 左右，與之前張延任教授 [2]分析出來的延遲時間大略相同。

我們另外使用 CACTI 工具模擬不同組態的指令快取記憶體與分支目標緩衝器(包含哨兵表)的延遲時間，例如：64K 快取記憶體延遲時間大約為 1.779 ns， 4Way-512 的分支目標緩衝器(包含哨兵表)延遲時間大約為 1.581 ns，如圖 14 所示。最後把所有分支目標緩衝器(包含哨兵表)和快取記憶體的延遲時間做統整，如圖 15 與表 2 所示。0.18 μm-cache 代表快取記憶體的延遲時間，分別表示快取記憶體為 1KB、2KB、4BK、8KB、16KB、32KB、64KB，故呈現曲線狀，另外每一個橫軸代表分支目標緩衝器(包含哨兵表)的延遲時間。

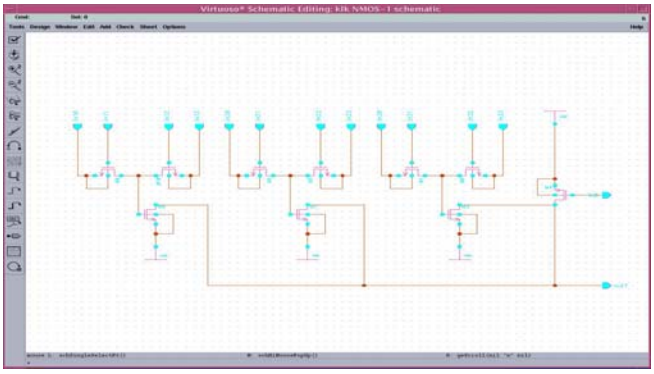

圖 13　使用 Virtuoso 工具畫出 CAM 電路架構

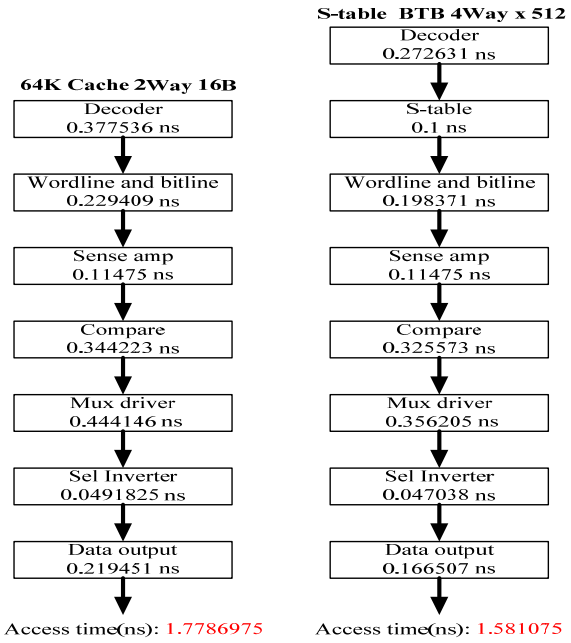

圖 14 使用 CACTI 工具執行結果

表 2　不同組態的快取記憶體與分支目標緩衝器之間延遲時間關係

0.18 μm-cache						
1KB	2KB	4KB	8KB	16KB	32KB	64KB
1.13128 ns	1.16289 ns	1.2284 ns	1.26678 ns	1.40946 ns	1.58684 ns	1.7787 ns
0.18 μm-BTB						
4way-64	4way-128	4way-256	4way-512	4way-1024		
1.30373 ns	1.37246 ns	1.46485 ns	1.58107 ns	1.79956 ns		
8way-64	8way-128	8way-256	8way-512	8way-1024		
1.34377 ns	1.40504 ns	1.49063 ns	1.67605 ns	1.97576 ns		

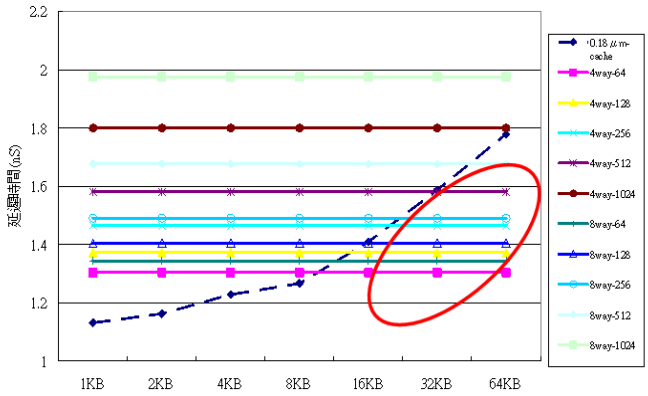

圖 15　不同組態的快取記憶體與分支目標緩衝器之間延遲時間關係

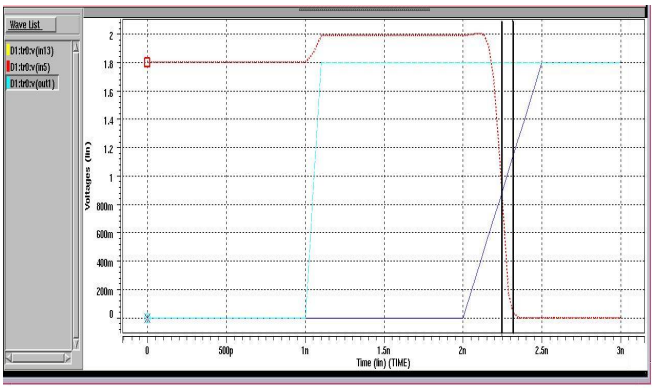

圖 16　HSPICE 模擬結果波形圖

(二) 哨兵表額外面積

在本節我們討論使用哨兵表之後，硬體電路增加的效應。哨兵表的內容是由原本的分支目標緩衝器的標籤中取出，並且放置到哨兵表中，之後分支目標緩衝器的標籤即減少被取出的位元數目。在面積計算的方面，我們使用電晶體數目來評估。使用哨兵表的方法大約會使得電晶體增加 4% ～ 5%，依據不同的分支目標緩衝器組態，所增加的電晶體數目也不相同。

在此我們先定義一些變數，如下：

N_{Tag}：標籤的位元數目

N_{Addr}：位址的位元數目

N_{Way}：集合關聯度的數目

N_{Set}：每一路有的集合數目

$N_{S\text{-}bit}$：哨兵位元的數目

在原始的分支目標緩衝器是以 SRAM 6T 的架構組成，所以電晶體總數為：

$$N_{BTB}：(N_{Tag}+N_{Addr}) \times N_{Way} \times N_{Set} \times 6 \qquad (5)$$

由於我們使用 CAM 9T 的架構設計哨兵表(以下簡稱 S-table)，所以在計算電晶體數目時，要分成三個部份計算：

$N_{S\text{-}table}：N_{S\text{-}bit} \times N_{Way} \times N_{Set} \times 9$

$N_{BTB}：(N_{Tag} - N_{S\text{-}bit} + N_{Addr}) \times N_{Way} \times N_{Set} \times 6$

$N_{ANDarray}：N_{Way} \times N_{Set} \times 6 \qquad (6)$

範例 1：

N_{Addr}：32　　N_{Set}：256　　N_{Way}：4　　$N_{S\text{-}bit}$：3

$N_{Tag} = N_{Addr} - \log_2 N_{Set} - \log_2 N_{Way} = 32 - 8 - 2 = 22$

Orig. BTB

$(22 + 32) \times 4 \times 256 \times 6 = 331776$

Filter BTB

$N_{S\text{-}table}$：$3 \times 4 \times 256 \times 9 = 27648$

N_{BTB}：$(22 - 3 + 32) \times 4 \times 256 \times 6 = 313344$

$N_{ANDarray}$：$4 \times 256 \times 6 = 2048$

Total：347136

電晶體額外增加

$(347136 - 331776) \div 331776 = 4.6296\%$

由範例 1 可知當原始分支目標緩衝器為 4W-512 的時候，採用過濾器方法改善動態耗電的時候，會額外增加大約 4.62%的電晶體數目，詳細各組態所增加電晶體數目請參考圖 17。

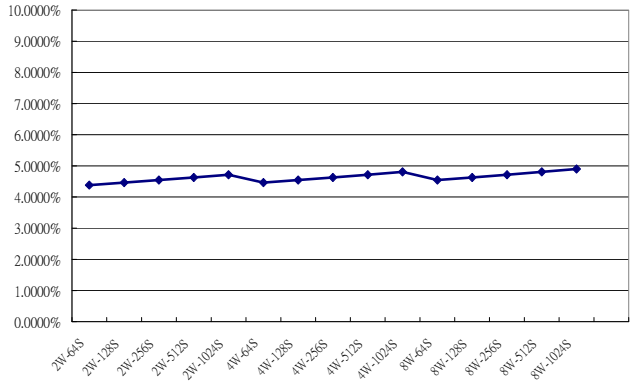

圖 17　不同組態分支目標緩衝器使用過濾器方法所增加硬體電路

(三) 哨兵表過濾效能

觀察圖 15 可以發現在某一些組態(快取記憶體大小與分支目標緩衝器)時，使用過濾器方法並不會影響到整體的最常路徑延遲。我們將這些組態的組合，使用 SimpleScalar 與 Wattch 執行 SPEC2000 的標竿程式，得到個別的過濾率。我們發現在不同指令快取記憶體大小的情況之下，如果分支目標緩衝器的大小若是相等時，則其分支目標緩衝器過濾效果是相同的，例如指令快取記憶體：64KB、分支目標緩衝器：4W-256 會與指令快取記憶體：32KB、分支目標緩衝器：4W-256 過濾效果相同，圖 18 與圖 19 列出所有實際模擬的結果。

(四) 功率節省效果

在這個章節討論耗電相關的問題，在 Wattch 中定義了許多參數，Wattch 提供的耗電模組從 0.4μm ～ 0.1μm 製程，在本論文中我們使用 0.18μm 製程為基礎，模擬在 0.18μm 功率消耗，Wattch 的耗電模組在定義分支預測單元以及分支目標緩衝器如下列公式(7)(8)。

$$bpred\_power = power\_btb + power\_local\_predict + power\_global\_predict + power\_chooser + power\_ras \quad (7)$$

$$power\_btb = array\_decoder\_power + array\_wordline\_power + array\_bitline\_power + senseamp\_power \quad (8)$$

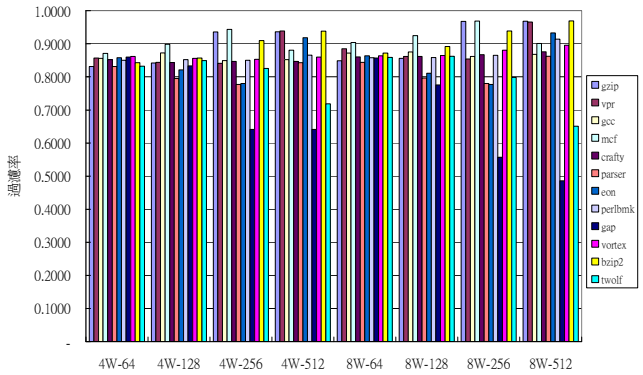

圖 18 整數標竿程式在不同組態時的實際過濾率

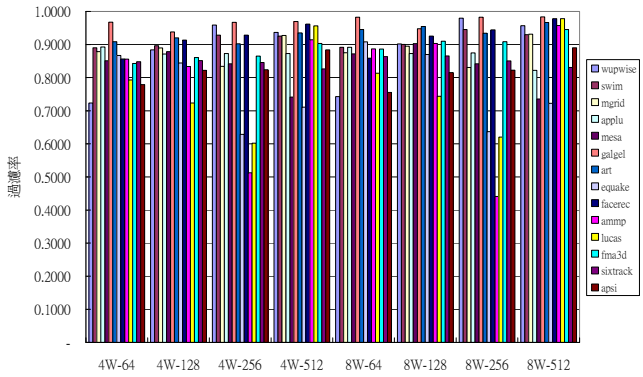

圖 19 浮點數標竿程式在不同組態時的實際過濾率

我們定義新的參數，使用過濾器方法後，分支目標緩衝器的耗電 (Filter_Scheme_power_btb) 以及分支預測單元的耗電 (Filter_Scheme_bpred_power)，公式如下。

$$Filter\_Scheme\_power\_btb = array\_decoder\_power + (array\_wordline\_power * filter\ rate) + (array\_bitline\_power * filter\ rate) + (senseamp\_power * filter\ rate) + S\text{-}table\_power \quad (9)$$

Filter_Scheme_bpred_power =
Filter_Scheme_ power_btb +
power_local_predict +
power_global_predict +
power_chooser + power_ras (10)

我們將由(7)(10)定義出分支預測單元省電效率(%)

bpred_power_save =
1-(*Filter_Scheme_bpred_power/bpred_power*)
(11)

我們將由(8)(9)定義出分支目標緩衝器省電效率(%)

power_btb _save =
1－(*Filter_Scheme_ power_btb/power_btb)*
(12)

最後的模擬結果得到的數據如圖 20、圖 21 圖 22 與圖 23 所示。由圖 20 到圖 23 中我們可以發現，當分支目標緩衝器的組態為八路的時候，其省電的效能會比四路的分支目標緩衝器還優秀。原因是八路的分支目標緩衝器比四路分支目標緩衝器多一倍的耗電量，在相同的分支預測器組態之下，分支目標緩衝器在整體分支預測單元的耗電量上就佔有不同的比例。利用過濾器方法的時候，分支預測單元的省電效能會與所能節省耗電的部份所佔整體比例相關，即是分支目標緩衝器耗電比例越大，則過濾器方法省電效能也會越好。

我們也發現在同樣路數的時候，越多筆數(entry)的情況之下，省電效能也會下降，原因是解碼器電路的耗電量在整體分支預測單元中增加了。在越多通道的分支目標緩衝器中，解碼器電路耗電量在整體分支預測單元中所佔比例也會提高，因為過濾器方法不能減少解碼器電路的動作次數，所以越多通道的時候，使用過濾器方法的效能會下降。

在總筆數數目為 2K 的時候，根據我們實驗結果，我們會推薦使用 8 路 256 集合的分支目標緩衝器作為設計方向。

使用過濾器方法的分支目標緩衝器省電效率在 18% - 75% 之間，分支預測單元的省電效率在 16% - 55% 之間，詳細數據如圖 24 與圖 25 所示。圖 24 與圖 25 皆是所有的標竿程式取算術平均數所得到的圖。

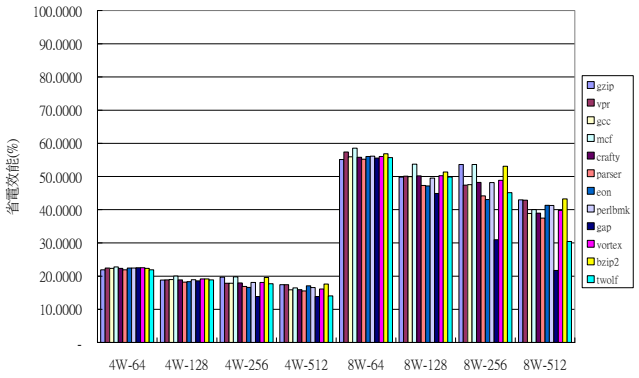

圖 20　分支預測單元整體省電效率-INT(%)

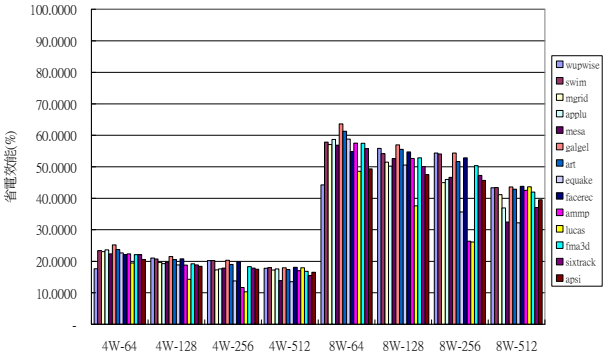

圖 21 分支預測單元整體省電效率-FP(%)

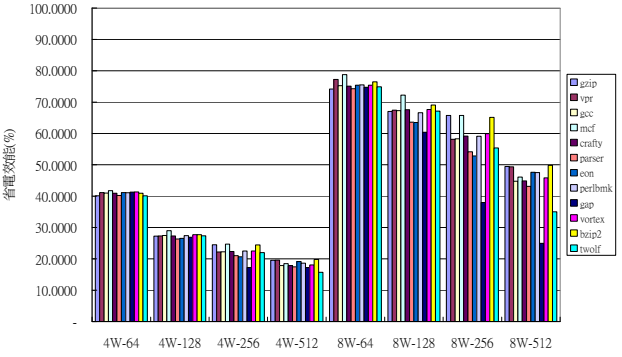

圖 22 分支目標緩衝器整體省電效率-INT(%)

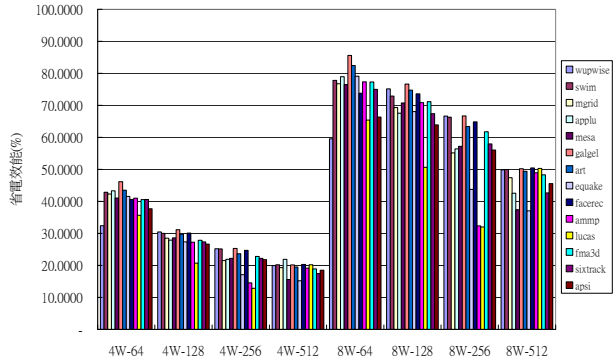

圖 23 分支目標緩衝器整體省電效率-FP(%)

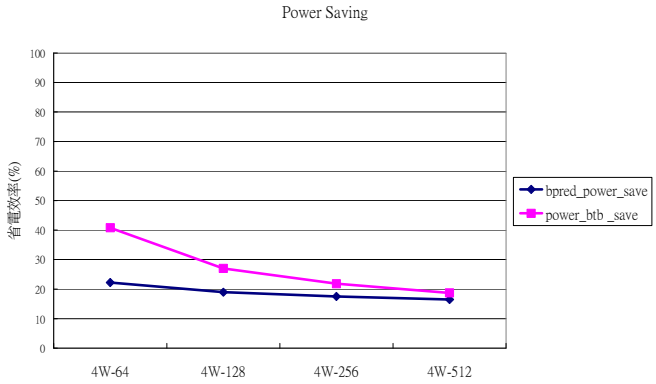

圖 24 分支目標緩衝器為四路時各組態省電效率

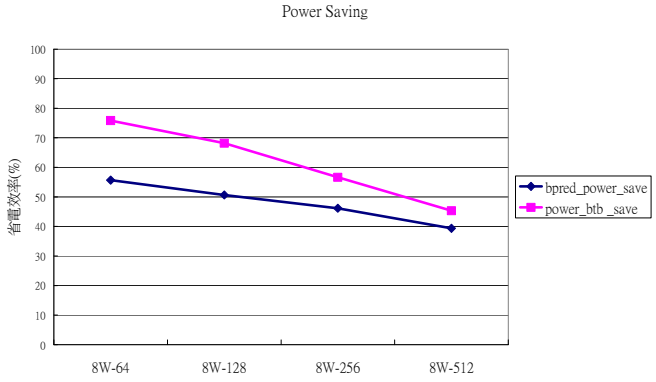

圖 25 分支目標緩衝器為八路時各組態省電效率

五、結論與未來研究方向

本篇論文從硬體的方面來考量，利用簡單的過濾器架構，減少分支目標緩衝器的存取動作以降低動態耗電，本篇論文提出分支目標緩衝器的設計方法，在不犧牲分支預測準確度與處理器效能並且用低硬體成本(小於5%)的前提下，分支目標緩衝器動作減少達到85%，分支預測單元省電達16% - 55%，分支目標緩衝器省電達18% - 75%。

我們發現在不同的分支目標緩衝器組態之下，使用過濾器方法的省電效率也都不盡相同，原因是過濾器方法不能過濾掉解碼器的動作次數。所以未來研究可以朝向減少解碼器動作次數，並且觀察程式行為，利用分支指令的特性，減少解碼器電路與分支目標緩衝器電路的動作次數，達到更好的省電的效果。

六、參考文獻

[1] 胡耀中，“低耗電分支目標緩衝器”，國立交通大學資訊工程學系碩士論文，2005.

[2] 財團法人國家實驗研究院-晶片系統設計中心, http://www.cic.org.tw/cic_v13/

[3] 喬偉豪，“低耗電分支目標緩衝器”，國立交通大學資訊工程學系博士論文，2008.

[4] A. Eden and T. Mudge, “The YAGS Branch Prediction Scheme”, In Proceedings. 31st International Symposium on Microarchitecture, pp. 69–77, Nov. 1998.

[5] B. S. AMRUTUR and M. A. HOROWITZ, “Fast low-power decoders for RAMs,” IEEE Journal of Solid-State Circuits 36, pp. 1506–1515, 2001.

[6] C.-C. Lee, I.-C. K. Chen, and T. N. Mudge, “The bimode branch predictor,” in Proceedings of the 30th Annual ACM/IEEE International Symposium on Microarchitecture, pp. 4 - 13, Dec 1997.

[7] D. A. Jimenez and D. Burger, “Low - Power, High-Performance Analog Neural Branch Prediction,” Microarchitecture, MICRO-41, 41st IEEE/ACM International Symposium, pp. 447–458, Nov 2008.

[8] D. Parikh, K. Skadron, Y. Zhang, M. Barcella, and M. Stan, “Power issues related to branch prediction,” In Proceedings. HPCA-8, pp. 233–44, Feb. 2002.

[9] D. Brooks, “Wattch Version 1.02,” in http://www.eecs.harvard.edu/~dbrooks/wattch-form.html

[10] D. A. Jimenez and Calvin Lin, “Dynamic branch prediction with perceptrons,” In Proceedings of the 7th International Symposium on High Performance Computer Architecture, pp. 197–206, January 2001.

[11] D. Tarjan. et al., “Cacti 3.2 web interface,” in http://quid.hpl.hp.com:9081/cacti/

[12] E. Sprangle, R. Chappell, M. Alsup, and Y. Patt, “The Agree predictor: Amechanism for reducing negative branch history interferee-ence,” In Proceedings of the 24th Annual International Symposium on Computer Architecture, May 1997.

[13] I. Bate and R. Reutemann, “Efficient Integration of Bimodal Branch Prediction and Pipeline Analysis,” In IEEE Conference on Real-Time Computing Systems and Applications, pp. 39–44, Hong Kong, China. 2005.

[14] J. Lee and A. J. Smith, “Branch prediction strategies and branch target buffer design,” IEEE Computer, pp. 6–22, January 1984.

[15] K. Flautner, N . S. Kim, S. Martin, D. Blaauw, and T. Mudge, “Drowsy caches: simple techniques for reducing leakage power,” In Proceeding. International. Symposium. Computer Architecture, Alaska, pp. 219-230, May 2002.

[16] M. Monchiero and G. Palermo, “The Combined Perceptron Branch Predictor,” Technical Report n. 2004.35, Dept. of Electronics & information Technical Report, Milan, Polytech Institute, Milan, Italy, 2004.

[17] P. Michaud, A. Seznec, and R. Uhlig, “Trading conflict and capacity aliasing in conditional branch predictors,” In Proceedings of the 24th Annual International Symposium on Computer Architecture, pp. 292-303, May 1997.

[18] P. Chang, M. Evers, and Y. Patt, “Improving branch prediction accuracy by reducing pattern history table interference,” In Proceedings of the International Conference Parallel Architecture and Compilation Techniques, pp. 48–57, October 1995.

[19] R. Kessler, “The Alpha 21264 microprocessor,” IEEE Micro, 19(2): pp. 24–36, March/April 1999.

[20] S. Kaxiras, Z. Hu, and M. Martonosi, “Cache decay: Exploiting generational behavior to reduce cache leakage power,” Proceeding. of International Symposium. Computer Architecture, pp. 240-251, 2001.

[21] Shuai Wang, Jie Hu, and Sotirios G. Ziavras, “BTB Access Filtering: A Low Energy and High Performance Design,” In Proceeding. of the IEEE Computer Society Annual Symposium on VLSI (ISVLSI 2008), pp. 81 - 86, April 7-9, 2008.

[22] Todd M. et al., “SimpleScalar 3.0,” in http://www.simplescalar.com/

[23] T.Y. Yeh and Y. N. Patt, “Two-level adaptive training branch prediction,” In Proceedings of the 24th Annual International Symposium on Microarchitecture, pp. 51-61, November 1991.

[24] S. McFarling, “Combining branch predictors,” Digital Equipment Corporation, WRL Tech. Note TN-36, 1993.

[25] Y. J. Chang, S. J. Ruan, and F. P. Lai, F, “Sentry Tag: An Efficient Filter Scheme for Low Power Cache,” In Proceeding. Seventh Asia-Pacific Computer Systems Architectures Conference (ACSAC2002), Melbourne, Australia, pp. 135-140, 2002.

[26] Y. J. Chang, S. J. Ruan, and F. P. Lai, “Design and analysis of low power cache using two-level filter scheme,” IEEE Transactions on Very Large Scale Integration Systems, vol. 11, pp. 568–580, Aug 2003.

[27] Standard Performance Evaluation Corporation, http://www.specbench.org/osg/cpu2000/.