國立中興大學電機工程學系　　（附件一）
專題競賽報告

一種靈活且高效能應用於稀疏卷積神經網路硬體加速器之設計與實作
Implementation for a Flexible and Energy-Efficient Accelerator For Sparse Convolution Neural Network

專題題目說明與價值（限 100 字內）：

本專題針對 SCNN 的特性，設計的加速器可以大幅減少不必要的運算和儲存需求，並進一步減少能耗，並且引入 im2col 的資料重構模式，讓整個系統更加靈活，同時也提出一種交換不同種類的資料儲存的位置來最佳化不同運算的方法。

自評貢獻(與過去前輩成果不同之處)（限 100 字內）：

本專題實作並改良了 Eyeriss v2 架構的類神經網路加速器，並進行一系列 cell-based design flow 得到 GDSII，我們的設計的加速器的結果在 MAC 只使用 Eyeriss v2 的一半的條件下，也能有用有相近的推理速度，及更低的面積與功耗。

專題組員：

姓名	E-mail	負責項目說明	專題貢獻度(%)
陳柏淳	booo.c.chen@gmail.com	硬體架構設計與合成驗證、AI 模型訓練及輕量化	100%

【說明】上述表格之專題內貢獻度累計需等於 100%。

目錄

一、 摘要（含關鍵詞）

二、 專題研究動機與目的

三、 專題重要貢獻

四、 設計原理、研究方法與步驟

五、 實現與實驗結果

六、 效能評估與成果

七、 結論

八、 參考文獻

一、摘要：

深度神經網絡（DNN）已通過在圖像分類、辨識等任務中展示出色的性能，革命性地改變了許多領域。然而，DNN 的計算、低功耗和記憶體需求巨大，這強調了需要專門的硬體加速器（ASIC）以達到快速且低功耗的運算。

我們提出了一種基於 Eyeriss v2[1][2]架構的通用型神經網絡硬體加速器架構。結合了 im2col+GEMM 的數據重構方案，簡化了神經網絡中的數據處理同時增加了硬體架構的靈活性，我並且在 NoC 層面引入了脈衝陣列(systolic array)的設計，有效地解決了原始 HM-NoC 硬體架構中 combination loop 的問題，此外我們也在資料流的設計中對於 input feature map 和 weight 使用相同的硬體模組，並且在不同 layer 之間交換 input feature map 和 weight 的儲存位置以最大化的復用(reuse)資料。我們使用 OpenROAD 工具，將我們設計的硬體加速器合成至 GDSII。在 NanGate 45nm CMOS 的製程工藝，系統推理 Mobilenet(convolution only)的結果達到每秒 1559.7inference/sec (batch size=1)，此外也有將硬體實作於 FPGA 平台做驗證。

Index term—deep neural network accelerator、Deep neural network、Energy efficient Accelerator、data flow processing、spatial architecture 、Eyeriss v2、systolic array、im2col、GEMM、FPGA

二、專題研究動機與目的：

深度神經網絡（DNN）已成為現代計算模式發展的基石，並且在識別複雜模式方面的無與倫比能力，DNN 應用到移動裝置中，作為我們日常生活中的主要計算界面，是一個重要的前沿領域。然而，在移動平台上有效運行 DNN 的任務呈現出顯著的挑戰。儘管移動設備的計算能力不斷增強，但仍會受到功率、記憶體和處理能力的限制。特別是在執行資源密集型 DNN 操作時，導致推理速度受阻和功耗增加[3]。這一困境突顯了專門的硬體加速器的迫切需求，這些加速器旨在滿足 DNN 的強大計算需求，同時不損害能源效率或實時性能。

隨著深度學習的持續進步，對高性能且低功耗加速器的需求日益增加。為應對這一日益增長的需求，已經出現了許多加速器架構，每一種都針對特定的操作、性能指標和功耗考慮而設計。減輕 DNN 計算負擔的一個有前景的策略是利用模型稀疏性。通過利用非零權重較少的稀疏模型，我們可以顯著減少計算和記憶體開銷，在移動平台上提供獨特優勢。

A. Im2col+GEMM 面臨的挑戰:

Im2col 結合 GEMM 操作已成為卷積神經網絡（CNN）[4]加速的普遍方法，如 Fig.1 所示。Im2col+GEMM 的主要優勢在於它能將可能不規則且對硬體實現具挑戰性的卷積操作轉換為可在硬體平台上有效執行的規則矩陣乘法。這種重構顯著提高了速度和並行性，然而這種轉換帶來了許多挑戰，例如在轉換時會導致數據擴展，對 On-chip memory 的容量要求非常高。此外，在龐大的矩陣中管理數據流並確保效率也很複雜，需要創新的架構解決方案。

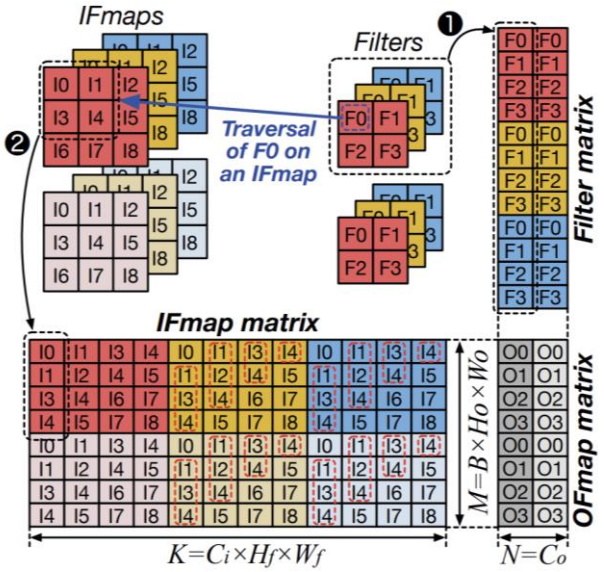


Fig. 1 Im2col GEMM converted from the convolution[5]

B. 稀疏 DNN 面臨的挑戰

稀疏 DNN 在資源受限環境中帶來了一系列優勢。通過消除與零或接近零的權重相關的計算，稀疏 DNN 大大減少了計算工作量和記憶體提取，加快了推理時間並節省了能源。儘管這些好處引人注目，但在硬體上實現稀疏 DNN 並非易事。非零權重分布的不規則性可能導致數據訪問模式不可預測，使硬體設計變得複雜，並可能與管理稀疏性的開銷抵消了計算優勢。

C. Eyeriss v2 架構面臨的挑戰

Eyeriss v2 在 Eyeriss 的成功基礎上推出，旨在解決其前身的限制。它最突出的特點之一是靈活的架構，通過 row stationary plus（RS+）data flow，實現對不同類型和大小的 CNN 層的適應性，確保了更好的資源利用。它還結合了更高效的 Hierarchical mesh NoC (HM-NoC)，進一步優化了數據運動。然而，儘管有這些改進，Eyeriss v2 仍面臨挑戰。一個問題是其 HM-NoC 硬體中的組合迴路存在，另一個問題則是在不同的神經網絡模型上的 PE 使用率、資料復用率以及模型稀疏不一等讓系統不穩定的問題，這些都阻礙了其推理模型時的可靠度。

三、專題重要貢獻：

針對在移動設備上部署 DNN 所面臨的挑戰，此研究進一步發揮了為稀疏模型量身定制的硬體加速器的潛力。我們的貢獻列舉如下：

i. 改進 Eyeriss v2 架構：

我們復現了著名的神經網絡硬體加速器 Eyeriss v2。在認識到其優勢並識別改進領域後，我們透過在不同 layer 之間交換 weight 和 input feature map 所使用的硬體位置，這種資料流可以提升其性能與資料的復用率，同時我們也設計了完整的硬體架構，包含 CSC 壓縮、im2col 轉換的模組以及將整體系統用 pipeline 的形式實現。

ii. CSC 壓縮搭配 Im2col+GEMM 數據重構：

在傳統 DNN 架構中，主要的計算量和記憶體的頻寬都用於卷積運算。通過採用 im2col+GEMM 方法，將卷積操作轉換為矩陣乘法。這種重構不僅簡化了計算，還優化了記憶體訪問模式，此外我們在 im2col 轉換到儲存到 GLB 之前接上 CSC encoder 並且引入 pipeline 的架構來實現從而減少記憶體占用並提高計算效率，此外由於 im2col 主要的缺點在於需要大量的 buffer 來儲存重構後的數據，但在我們的設計中，buffer 儲存的是經過 CSC 壓縮之後的數據，因此這種方法可以克服原本的問題。

iii. NoC 層面的脈衝陣列設計：

原始的 HM-NoC（Hierarchical Mesh-Network on Chip）由於其硬體架構中存在 combination loop 的問題。這可能導致在操作不同傳送資料模式時數據移動的效率低下以及電路潛在不穩定的風險。我們修改後的架構解決了這一問題，確保了加速器中順暢、高效且穩定的運作。我們的方法是在這一層面引入了脈衝陣列架構。脈衝陣列憑借其規律穩定數據傳播，提供了同步、可預測和高效的數據流機制。這種架構設計可以成功地解決問題。

iv. 稀疏 GEMM PE 設計：

我們改良後的 PE 現在支持更廣泛的計算模式，專門為 im2col+GEMM 數據重構技術量身定制。這確保 PE 不僅能夠適應常規 GEMM 操作，還能有效處理 im2col 方法重塑的數據。通過融入這種設計範式，我們優化了加速器內部 Global Local Buffer（GLB）的使用。經 im2col 轉換重構的數據現在以更高效率存儲在 GLB 中，減少了冗餘數據運動，提高了吞吐量，進一步增強了系統的能源效率，此外也針對 CSC 壓縮的個格式做了一些修改，以簡化 PE 中的控制邏輯電路。

v. 交換 loacl memory 儲存模式(weight stationary plus)：

我們提出的 PE 中存在 Former 和 Later 的 SPad，在 PE 內部 reuse Former SPad 的資料，在 PE 之間則是 reuse Later 的資料，在 PE 內部 psum 則是縱向累加。而在不同種類的運算模式下，Former 和 Later 的 SPad 儲存的資料種類會交換，在 convolution layer 的運算時，Former SPad 儲存的是 weight(PE 內 reuse weight)，Later SPad 儲存的則是 iact(PE 之間 reuse iact)。在 Fully connected layer 的運算時，Former SPad 儲存的是 iact(reuse iact)，Later SPad 儲存的則是 weight，在不同運算時都可以最大化 reuse 資料以達到 Energy-Efficient，我們稱這種 dataflow 以及 PE 內部交換儲存資料的模式稱為 weight stationary plus (WS+)。

四、設計原理、研究方法與步驟：

A. ARCHITECTURE OVERVIEW

在原始 Eyeriss v2 設計的基礎上，我們的架構引入了幾項關鍵改進，其中包含了 CSC encoder、im2col converter、quantizer，以此更加完善了整個硬體加速器系統的運作，如圖 Fig.2 所示。在核心配置中，cluster array 以 8x2 集群群組的排列構建，可根據計算需求進行調整。這種靈活性確保我們的設計能適應多種神經網絡配置。

Encoder group 主要負責處理資料。其中 Im2col converter 將圖像數據重構成列，然後再將重構後的資料傳進 CSC encoder 直接進行壓縮，最後再將壓縮後的資料傳進 GLB 中，從而減少 SRAM 的占用。當 cluster array 運算完後，他會將 psum 從 GLB 中讀取出來，傳到 quantizer 中，將數據表示從 21 -bit 降低到 8 -bit。

在 cluster array 內部，我們配置了各種組件以實現最佳性能。作為計算核心的 PE cluster 由一排 3x4 處理元件（PEs）組成，負責網絡中大部分算術運算。GLB cluster 作為內存的 SRAM Bank。其包含 7 個獨立的 SRAM ，分別用於存儲中 iacts(壓縮後)以及 psum，規格如表 I 所示。最後 Router cluster 則是對於架構內部的數據流動至關重要。它分為三個 iact router，三個 weight router 以及四個 psum router。此外，每個 router 內部的結構設計也做了一些修改，使相比原始的結構更加簡單。

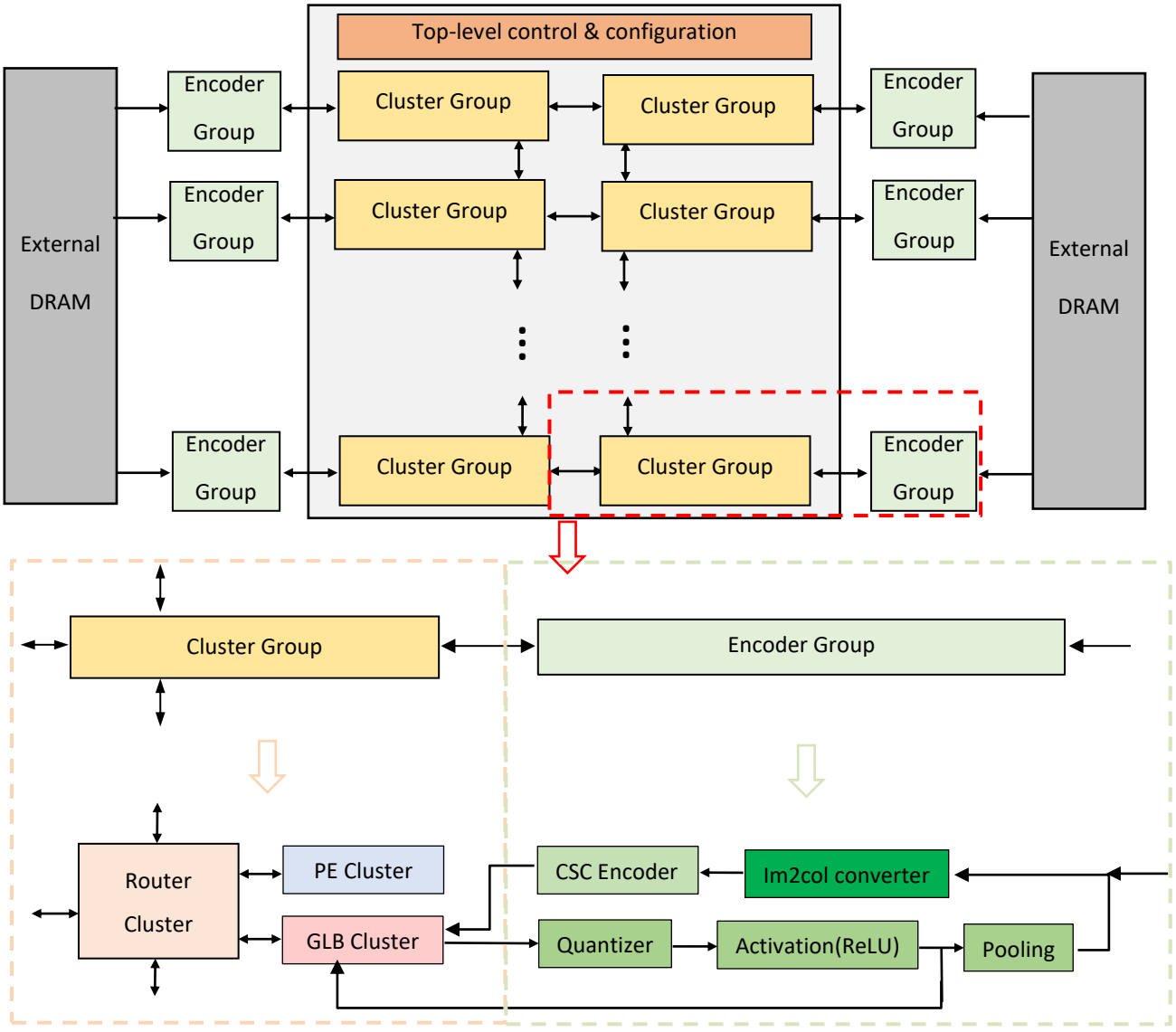


Fig. 2 Accelerator top-level architecture

B. GLB Cluster architecture

GLB（Global Buffer）cluster 作為核心加速器與外部 DRAM 之間的橋樑。GLB 包含 3 個 iact SRAM 和 4 個 psum SRAM，分別對應 PE Cluster 內 PE array 的分佈以提供足夠的頻寬。GLB cluster 內部也設有專門的 weights ports，但這些 weight ports 只用來連接到 weight routers 而已，與 iact 和 psum SRAM 不同，GLB 內部沒有專門為 weights 設置的 SRAM 儲存。這種設計是由於 weight 會直接儲存在 PE 內的 weight SRAM 中，類似於傳統的 weight stationary data flow。

加速器內的 SRAM 組件是利用 OpenRAM [6] 合成，此外，GLB cluster 內的所有 SRAM Banks 都能獨立運作，增加了系統的並行性和效率。由於 iact SRAM 是儲存已經經由 CSC 壓縮之後的資料，因此它分為兩個獨立的 SRAM，一個專門用於存儲 address，而另一個保存 data count 的資料。

由於 CSC 壓縮後的資料長度會不一樣，無法透過一般的 counter 來決定送到 PE 的資料何時結束，因此我們在 iact SRAM Banks 內部另外增加了一組 Register file(RF)。這些 RF 功能類似於 LUT，用來追蹤要分給 PE 時不同組的起始和結束地址，其中透過內部的 FSM Controller 來偵測 0 並記錄位置在 LUT 中，同時偵測連續兩個 0 表示整格個壓縮資料的結束。其中使用了「0」來標記一組資料的結束，提供了清晰的界限，並輔助於數據檢索。此外，為了保證數據完整性和 GLB Cluster 內的溝通，我們也有使用以「Valid」和「Ready」這組訊號實現 two-way hand shake 的資料傳送協議來確保資料的正確性。GLB Cluster 的 block diagram 如 Fig.3 所示。

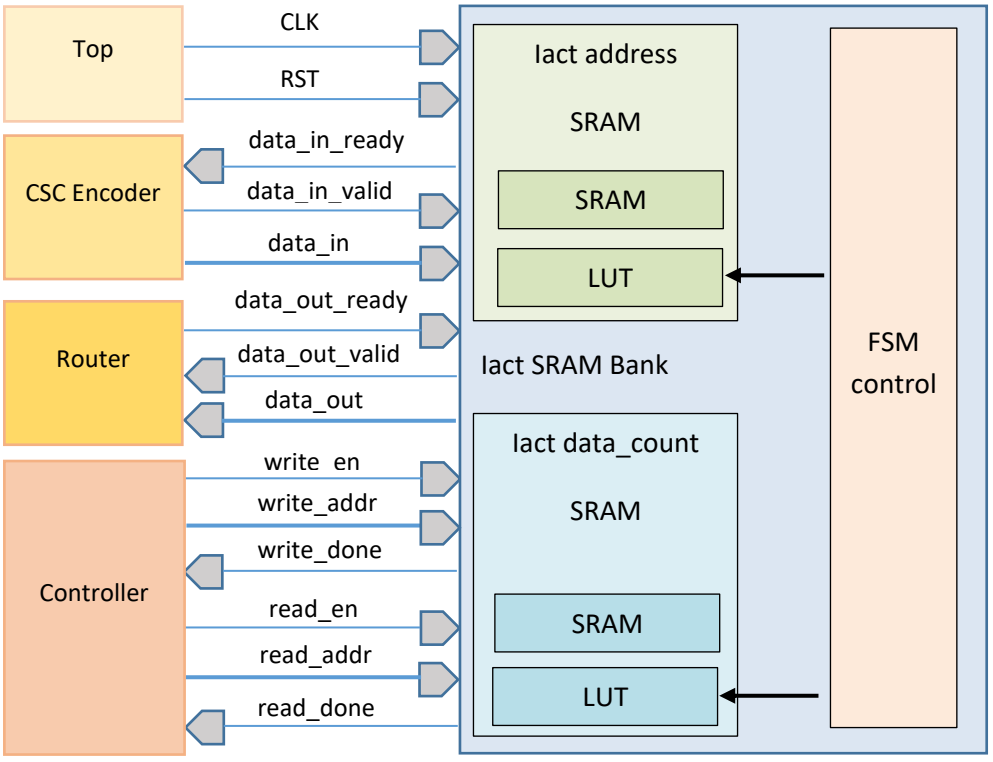


Fig. 3 Iact SRAM system block diagram (In the case of psum SRAM, only the data signal in the system block diagram.)

C. Router Cluster architecture

我們 router cluster 的設計主要參考 Eyeriss v2 中的 HM-NoC 的架構所設計，但我們在 router 的內部的設計做了些修改，原版中資料選擇是在末端進行的，每個輸出端口都有自己的 MUX，但我們的設計採取了更主動的方法。數據一到達路由器，就會遇到與 data_in_sel 信號協同工作的 MUX，立即決定最終將發送的數據。routing_mode (data_out_sel)簡單地指定了出口端口。這種設計通過前置決策過程，優化了數據流。通過減少冗餘操作和決策層次， 我們的設計相較於原版減少了 75%的 MUX 使用量。

這個 router cluster 包含了 3 個 iact routers，3 個 weight routers，以及 4 個 psum routers。每個 router 內部只使用 Circuit switching (MUX)來實現，systolic 使用的 register 則是放在 router 的連結之間。我們的設計除了標準的握手協議外，還增加了兩控制訊號 in_sel 和 routing_mode (out_sel)，這些對資料流的方向進行控制，其運作細節和邏輯如 Fig.4 所示。與 Eyeriss v2 原始的 router 相比，我們的設計引入 data_in_sel，以完成 3-way hand shake，進一步增強了 router 控制資料流的能力。

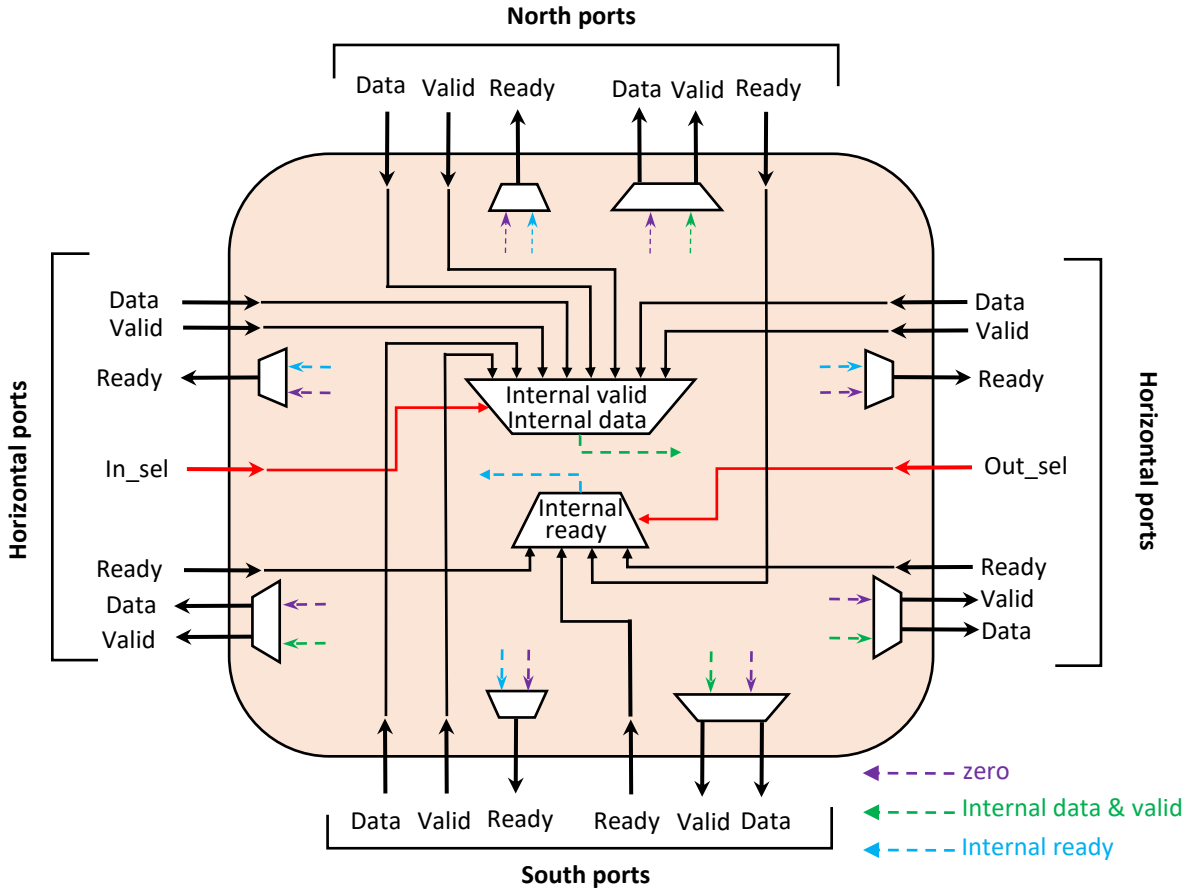


Fig. 4 Router Architecture

D. PE Cluster architecture

PE cluster 總共包含 12 顆 PE (3X4)，由於我們引入了 im2col 的資料重構模式，因此在 PE Cluster 要做修改，我們的設計將 3 列的 PE 分別與 3 顆 weight routers 連接 (weight 橫向 reuse)，將 3 列的 PE 分別與 3 顆 iact routers 連接。

首先，iact router 會依序傳入 PE Cluster 中，會分為三組，依序傳入不同行的 PE，此結構可以將每列的 PE 單獨處理卷積各別的 channel。當 PE cluster 全部都計算完後，會將 PE 裡的 MUX 設為外部 Psum_in 的累加模式，在 PE Cluster 中每顆 PE 裡面的 psum 進行縱向累加，最後會由第一列 4 顆 PE 送出 psum 經由 psum router 到 GLB 中跟其他 cluster 進行累加或合併。Fig. 6 顯示了整個 PE Cluster 和 router 的連接方式，藍色方框代表資料由 router 同時傳給方塊內所有的 PE，黃色方框代表資料由 router 依序特別傳給方塊內所有的 PE。

這種資料留會依照不同類型的運算而有改變，而 data resue 的類型也都會改變，例如在 convolution layer 時 PE 之間的資料流會是 weight stationary，PE 內部會是 input stationary，以此達到類似於 RS+ data flow，所有類型的 data 都有 reuse 的能力，而在 fully connected layer 時，iact 和 weight 輸入 PE 的位置會交換，PE 之間就會是 input stationary，而 psum 永遠都會是縱向累加。

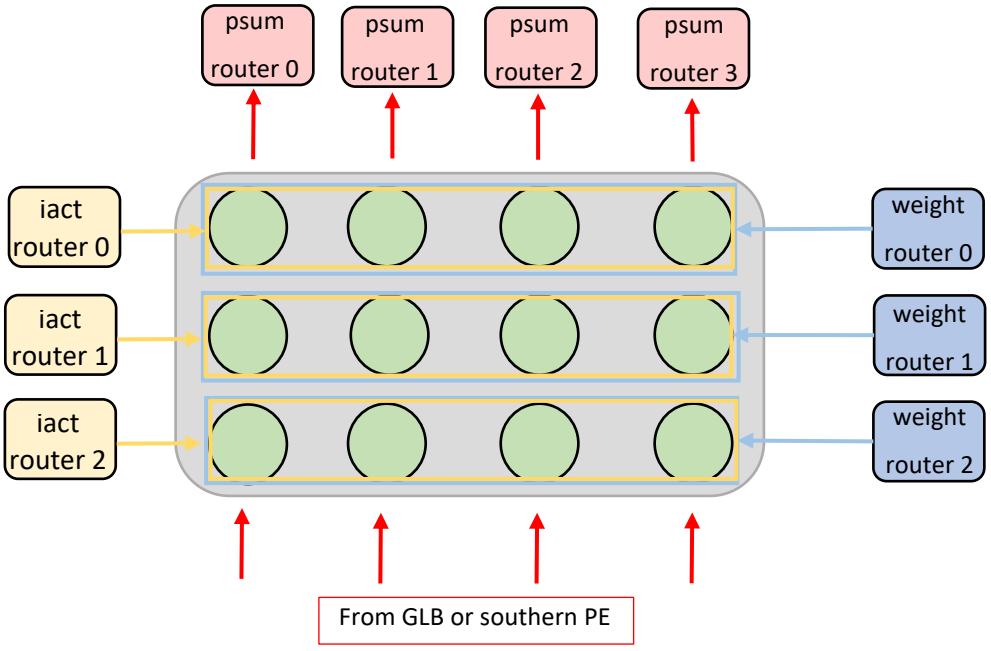


Fig. 5 The PE cluster-router connection

(Green: PEs, Yellow: Iact routers, Blue: weight routers, Red: psum routers.)

E. PE architecture

PE 裡面包含 5 個 SPad (Scratch Pad)來儲存資料，如 Fig.6 所示，其中 weights SPad 中是使用 SRAM 來儲存資料，其餘的 SPad 都是用 register file(RF)，為了要能夠運算 CSC 壓縮的資料格式，在解碼的過程中，要先由 Iact address 來確定要讀取 Iact data SPad 的哪個位址，接著 weight 透過 iact count 得到的位址來選擇要讀取哪個位址(column)，當 iact 和 weight 的 data 讀取後就可以透過他們的 count 來決定 psum 要寫入哪個位置，這樣的設計可以支援 GEMM 的運算，也可以讓同一個 iact 在對應 weight 中整行都一直復用當前的 iact (PE 內部復用 iacts，PE 之間復用 weights)。

我們加速器使用的 CSC 壓縮格式和 Eyeriss v2 一樣為三個 vector，但不同的是 address vector 在整行都為 0 的行數直接使用 address 的最大值來表示，如 Fig.7 所示，這樣不僅可以降低整體的複雜度，也可以不需要在多加個減法器來控制，同時在設計上層系統時就可以把 PE 內部都當成一個 black box，便於在 top level 時的控制也增加了靈活性，此外，count 格式也做了一些更改，我將 count 直接設為當前位置的列數，這樣可以在讀取時直接解碼而不需要再用額外的邏輯去計算與儲存 leading zero 的累加得到最終的列數。在每顆 PE 傳送資料的 I/O 也都配有 FIFO，使 PE 和外部(縱向 PE 或 GLB 傳過來的 psum)資料傳遞不同步時達到數據緩衝及確保資料的正確性。

每個 PE 都有獨立的狀態機在運作，PE 裡面的狀態機是整個系統中最複雜部分，它包含 7 個狀態來解碼 CSC 的資料格式並配對 iact 和 weight 相應的位置以完成 GEMM 的運算最後再將乘出來的結果和 SPad 裡面的 psum 相加(完成一次 MAC 運算)後存回 SPad 或把資料送到外部做後續的累加。

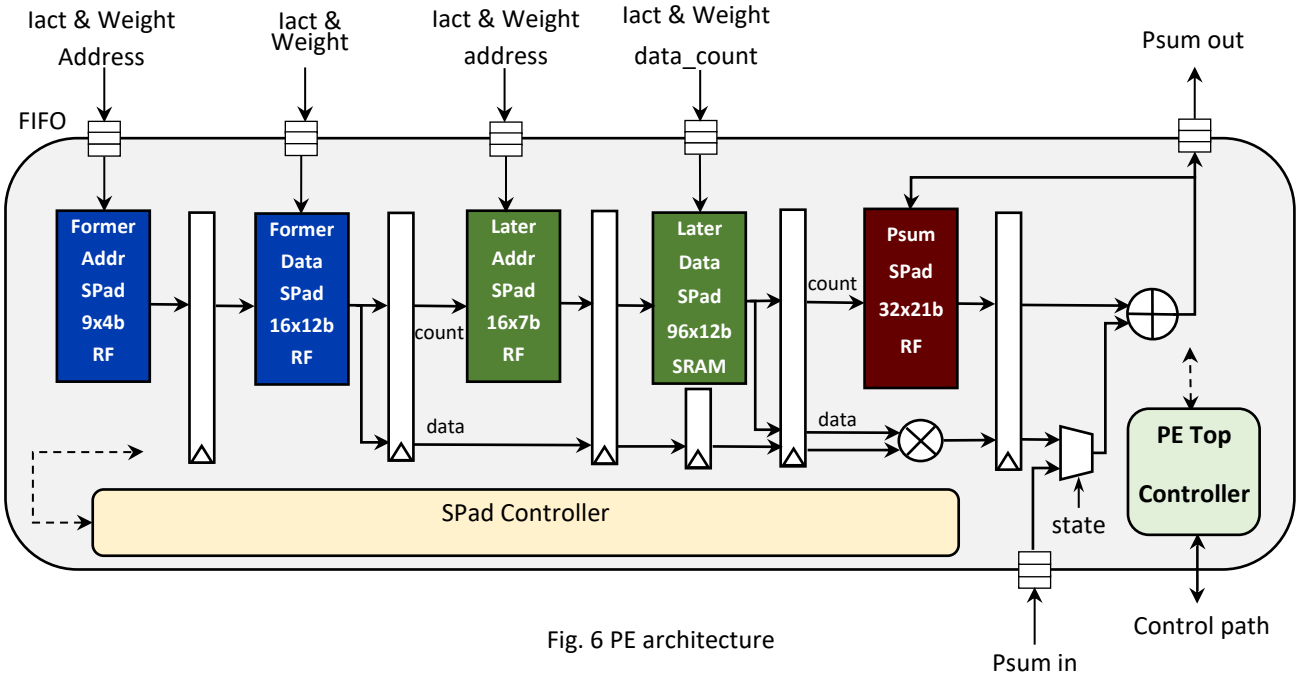


Fig. 6 PE architecture

	3		5	7
	4			8
1			6	9
				a
2				b

CSC Compress data:

Address vector :{2, 31, 4, 6, 11, 0}

count vector : {2, 4, 0, 1, 0, 2, 0, 1, 2, 3, 4, 0}

dara vector : {1, 2, 3, 4, 5, 6, 7, 8, 9, a, b, 0}

Fig. 7 The example of CSC encoding

F. Data flow & physical mapping

4.1 Im2Col + GEMM

當資料讀取進加速器後，必須實現 im2col 的轉換，我們設計一個轉換器把原本 ifmaps 的形狀根據 filter 的形狀和 stride 重新排列成新的矩陣，雖然這樣的做法會需要大量的儲存空間以及頻寬，但可以靈活地支援不同形狀的 filters。此外，在將 ifmap 儲存到 GLB 之前，會先把資料做 CSC 壓縮，以此減少儲存的空間，由此可以有效地解決 im2col 最大的缺點，此外，我們也可以利用轉換後的矩陣在對角線上有大量共用的特性，直接把原始 ifmaps 中的元素根據位置解碼輸出成矩陣形狀的所有位置再儲存到 GLB 中，最後再透過 NoC 分送資料進行後續的 GEMM 運算。

4.2 Data flow design and NN shape partitioning

在實際運算時，通常會將 input feature map(iact)和 filter(weight)分割，這樣可以充分地利用硬體的特性進行平行運算，並且可以有效地復用資料，由於我們引入了 im2col 的資料重構模式，引此原本的 Row stationary plus data flow 必修要做修改。

在我們的加速器中，資料留會根據當前執行的運算而有改變，如 Table 1 所示，由於 im2col 轉換之後的資料形狀已經是普通的矩陣乘法的運算了，因此我們平行化要割分的資料就會相對簡單。在 PE Cluster 中 PE array 每一列只處理一個 channel 的運算，在 PE array 的橫向會共用同一組 weight (weight stationary)，而在 PE 內部由於事先讀取 iact，因此同一個 iact 可以在 weight matrix 的整行都 reuse (input stationary)，由於每一列的 PE 都只處理一個 channel 的資料，因此 psum 可以縱向累加(橫跨多個 PE Cluster)。

在 fully connected layer(FC)和 depth-wise convolution layer(DW-CONV)時，資料進入 PE Cluster 的位置會交換，跟 PE array 的同一列使用同一組 iact (input stationary)，而每一列同樣只處理一個 channel ，在 FC 運算時由於 iact 做 fully connection 運算時會先 flatten，因此將 flatten 後的資料分割成 n 組，代表 n 個 channels，而 psum 同樣也可以縱向累加，但由於 fully connection 的運算，weight 都只會使用一次而已，因此在 PE 中的 weight 只會有一行，而 DW-CONV 則可以有多行。[7]提出不同層級的讀取資料的能耗(DRAM > Global Buffer > PE > RF(register files))，這種階層式的資料存儲模式再加上資料復用可以大大減少資料讀取帶來的能耗。

	Data flow		SPad Stored data type	
Layer type	Inter-PE	Psum SPad	Intra-PE Former SPad	Intra-PE Later SPad
CONV	Weight stationary	Vertical accumulate	Weight	Iact
DW-CONV	Input stationary	Vertical accumulate	Iact	Weight
FC	Input stationary	Vertical accumulate	Iact	Weight

TABLE 1 data flow and data store scheme

G. Data path design (Data scheduling)

在我們設計的硬體架構中，增加了 on-chip 的 CSC Encoder 以及 im2col converter，因此在實際運行時，會需要額外的 cycles 數來壓縮資料以及重構資料，因此，我們在設計的 data path 中，在不同的 modules 之間，需要做 pipeline 化的設計，以最大化地減少 loading data 以及壓縮資料的時間。Fig.9 為系統 data scheduling 以及簡化後的系統方塊圖。

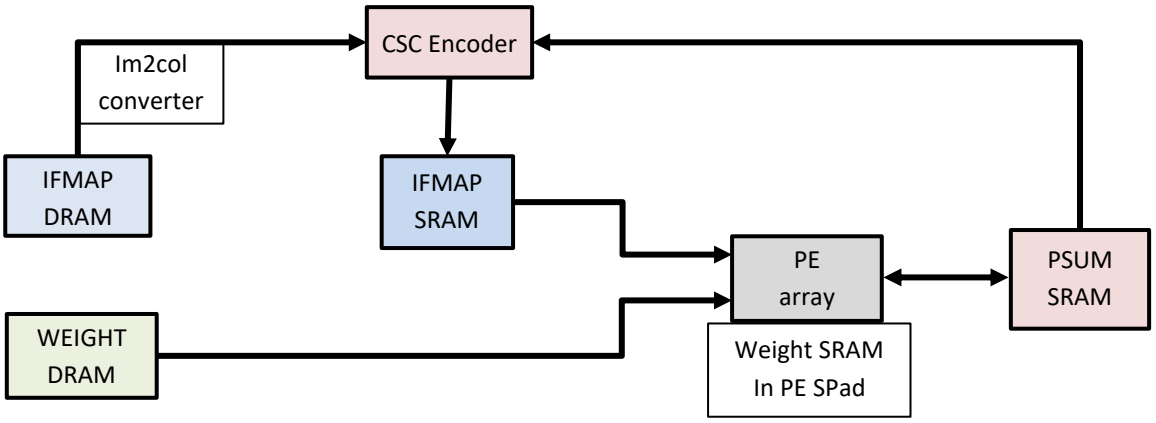


Fig. 8 (a) Simplified system block diagram

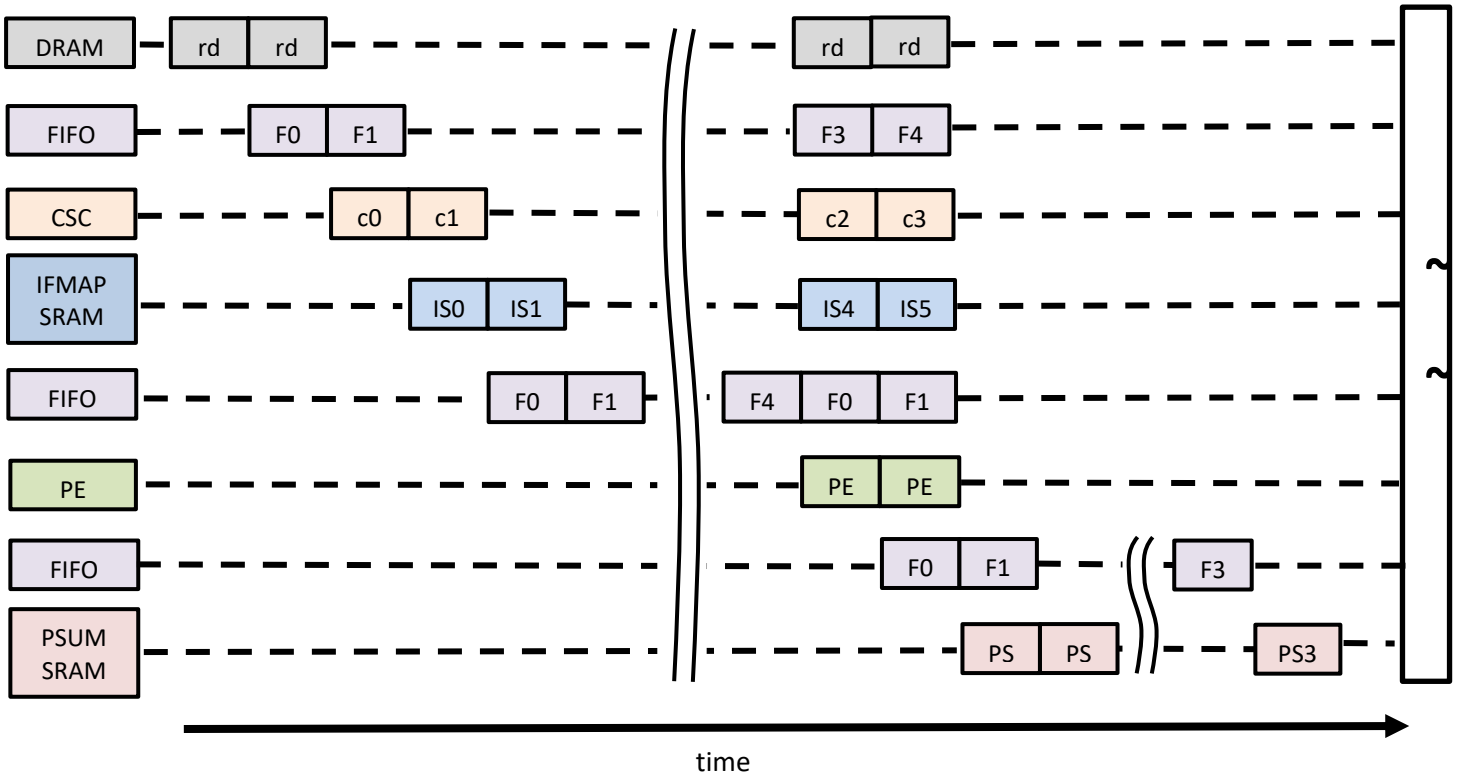


Fig. 8 (b) Data scheduling

H. Sparse model optimization (SCNN)

現今深度學習模型有需多優化的方法，其中最主流的方法為剪枝(Pruning)與量化(Quantization)[8]，一般通常都會先 Pruning 後 Quantization，但也可以依需要 Quantization 後重新 Pruning 來微調(Fine-tuning)模型，如 Fig.9 所示，這些技巧都可以讓整體模型變得更加輕量化且在加速器中也都扮演極為重要的腳色。

在我們的加速器中，相對於 RLC 格式只能用 clock-gating 來減少能耗，我們採用的 CSC 格式，還可以跳過所有 weight 和 iact 為 0 的 cycles 數。模型輕量化的 Pruning 在這方面有極大的貢獻，當我們 Pruning 的次數越多，剩下非 0 的值就會越少，[3]中提到深度學習模型中平均有 80%的 weight 都是不需要的，因此就算我們 Pruning 多次後仍然可以再次把模型訓練到之前相近的準確率，此外大部分的模型都有使用 ReLu 當作 activation function，這會使所有小於 0 的數值都設為 0，因此在訓練後往往會有大量的 0 存在於 weight 與 iact 中，CSC 的資料壓縮格式可以很好的利用這個特性來進行加速與節能，當模型的 sparse ratio 越高，我們加速器的速度就越快、儲存空間更小，並且能耗也會越少。

另一個優化技巧則是量化，由於現今模型在訓練時都是，32-bit floating-point 的精度，但當我們想要用於邊緣計算或嵌入式系統時，這麼高的精度準確率並不會比量化後高很多，因此我們的加速器也如一般主流加速器一樣設計為 8-bit fix-point 的精度，此外根據我們的實驗發現有時候量化後的準確率甚至會高於原本的精度，我們的推測是原本高精度下的模型，可能有類似過擬合(overfitting)的情況，然而在量化降低精度後反而讓模型有更好的泛用性。

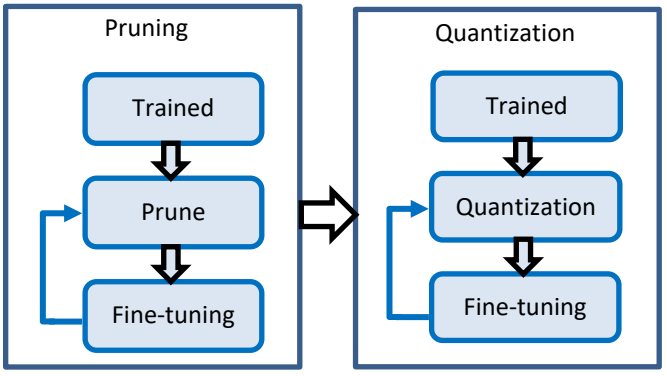


Fig. 9 Model Optimization Flowchart

五、實現與實驗結果

I. ASIC

我的們硬體實現使用 45nm CMOS 製成的工藝庫於 OpenROAD 平台上合成分析，規格如 Table V 所示。根據我們得實驗發現在矩陣分割後，在較高的 Sparse ratio 下，有很大部分的矩陣中一行只有一 1 或 0 個元素而已，在這樣的情況下 SIMD 的優勢也就消失了，因此我們的 PE 裡面沒有支援 SIMD，所以 GOPS 少了一半，但整體控制邏輯複雜度較低、減少面積及功耗，且實際運作的平均 GOPS 並不會差太多。

我們設計的加速器利用 OpenROAD 這款開源的 EDA tools 合成出 1297.3k 個 NAND-2(logic only)，而 SRAM 則使用 OpenRAM 生成，整個加速器在 45nm 的製程工藝下合成出的面積達到 2.34mm2 (Fig.15)，其中幾乎所有面積都由 PE 和 SRAM 所貢獻，而 NoC 和 Encoder 各別只佔了總體面積的 1.58 及 3.42%而已(Fig.14)，因此這個加速器的 NoC 架構的成本非常低，並且具有高度的可擴展性，也因為 NoC 不是 All-to-all 的架構，因此擴展的成本成線性增加而不是指

Tcchnology	NanGate 45nm
Area	1.53mm*1.53mm
Gate count(logic only)	1512k (NAND-2)
Total power	241.2mW
On-Chip SRAM	225KB
Number of PE	192
Scratch Pads (per PE)	weight addr: 14B(RF) weight data: 144B(SRAM) iact addr: 4.5B(RF) iact data: 30B(RF) psum: 84B(RF)
Clock Rate	200 MHz
Peak Throughput	76.8GOPS
Arithmetic Precision	weights & iacts: 8b fix-point psum: 21b fix-point

TABLE 2.
Accelerator Specifications

	Eyeriss v2	**This work**
Technology	TSMC 65nm	NanGate 45nm
Area (logic only)	2695k (NAND-2 gates)	1512k (NAND-2 gates)
On-Chip SRAM	246kB	225kB
Nums. of MACs	384	192
PE Architecture	Sparse	Sparse GEMM
Clock Rate	200MHz	200MHz
Batch size	1	
Peak throughput	153.6GOPS	76.8GOPS
Power	419.4mW (Best conditions)	241.2mW (average)

TABLE 3.
Compare Specifications with Eyeriss v2

DNN	**Performance**	**Eyeriss v2**	**This work**
Sparse AlexNet	Nominal Num. of MACs	724.4M	
	Inference/sec	278.7	335.6
	DRAM acc.	22.3 MB	13.0 MB
	PE Utilization[2]	100%	100%
	Sparse Ratio	Not mention	0.9
Sparse MobileNet	Nominal Num. of MACs	49.2M	
	Inference/sec	1470.6	1559.7
	DRAM acc.	3.9 MB	2.2 MB
	PE Utilization	91.5%	100%
	Sparse Ratio	Not mention	0.4

TABLE 4.
Compare Performance with Eyeriss v2

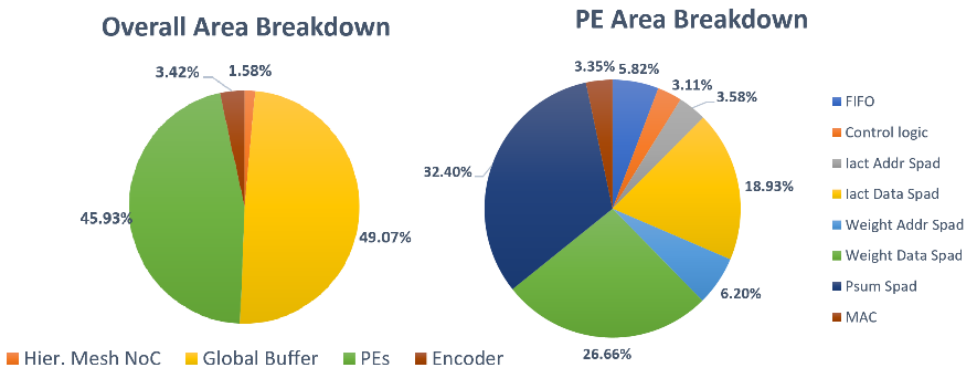


Fig.9 Accelerator area breakdown.

(a)Overall area breakdown (b) PE area breakdown

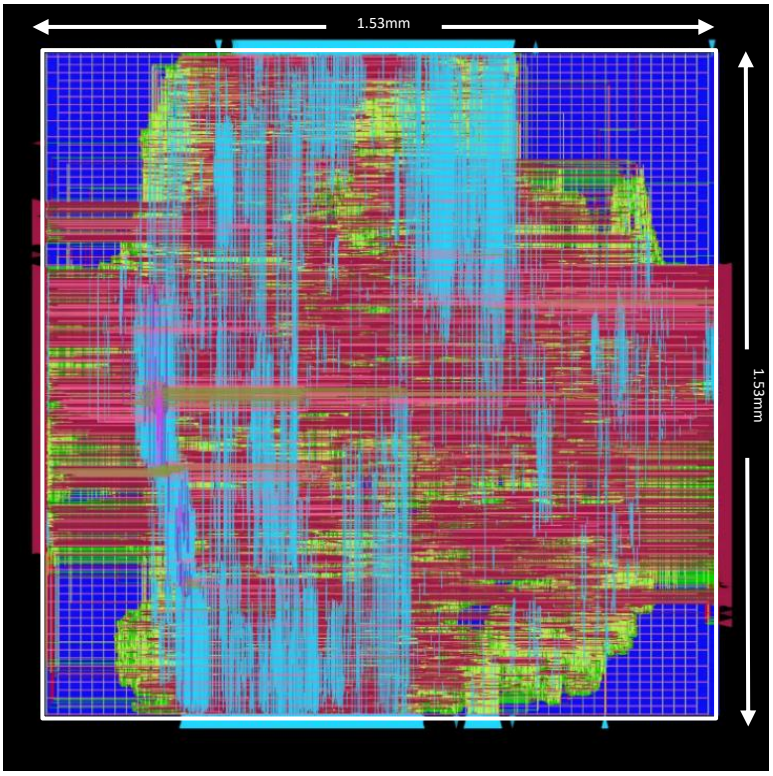


Fig. 10 Accelerator Layout

II. FPGA

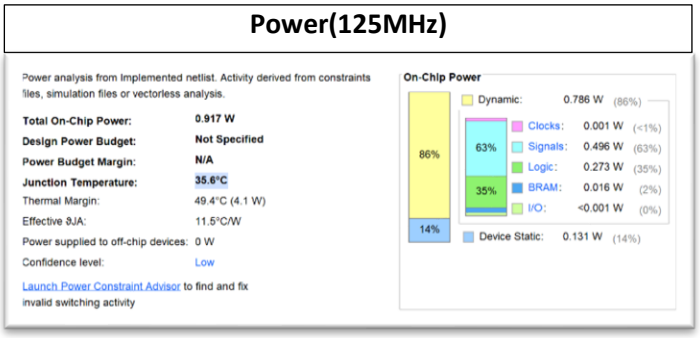
我們也有將加速器時坐在 FPGA 平台上驗證及展示。我們使用 PYNQ-Z2，透過 Vivado 將 verilog code 合成至 bitstream 並燒錄到 PYNQ-Z2 上，並且透過平台上 PL 端直接連接的外部 GPIO，將 PC 端的資料傳至 FPGA 平台上，在平台上我們也設計了一個 UART 協議的介面，讓 PL 端可以直接接收來自 PC 的資料。

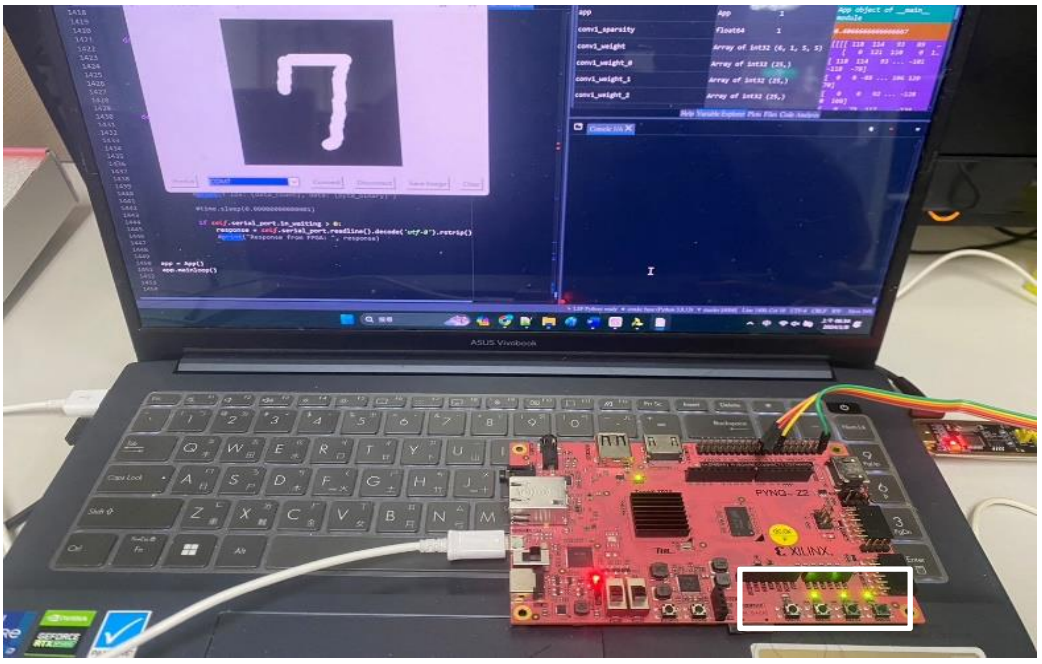
我們在 FPGA 上展示的 Model 為 LeNet-5，用來進行手寫數字辨識，在 PC 上則是用 python 設計了一個可以手寫數字 GUI 面板，透過實際手寫的數字，並將寫完的資料直接傳進 FPGA 上進行 Inference，以提升整體互動感及實際運行的真實性。

Resource Utilization

Resource	Utilization	Available	Utilization %
LUT	39577	53200	74.39
LUTRAM	339	17400	1.95
FF	51819	106400	48.70
BRAM	134.50	140	96.07
IO	7	125	5.60

Power(125MHz)





4 顆 LED 代表 2 進位的輸出結果

圖片視為 $0111_2 = 7_{10}$

Demo 影片：https://www.youtube.com/watch?v=wLz8Di9vdas&ab_channel=BOCHUNChen

六、效能評估與成果

A. Performance Analysis

我們的加速器採用和 Eyeriss v2 論文中類似的配置，主要不同在於我們引入了 im2col+GEMM 的優化，此外隨著 sparse ratio 增加，一行超過一個非 0 的元素會呈現指數下降，因此，我們的加速器沒有支援 SIMD，但在 im2col+GEMM 的優化下並且將 MAC 數量擴增到與 Eyeriss v2 一樣時，在 Sparse AlexNet 和 Sparse MobileNet 的推理速度相較 Eyeriss v2 論文中更快，同時也擁有更低邏輯複雜度及面積(Table VI、Table VII)。在推理 MobileNet 的過程中，為了讓我們的計算量和 Eyeriss v2 相近，我把 width multiplier 設為 0.5、input size 設為 128x128 [9]以便比較結果。此外根據我們的實驗結果發現原始 Eyeriss v2 的 NoC 架構由於 router 只由 circuit switching 構成，因此在實際設計上存在 combination loop 的問題，雖然這些 loop 在實際運行時可以透過設計讓它不會有任何導通的 case，但這仍然是電路的隱患，且 8x2cluster group 的結構下會極大增加延遲導致時序無法滿足，並使整個電路無法正常工作，因此在我們的設計中將 NoC 改良，別於傳統用在 PE 上 systolic array，我們將 systolic array mapping 到整個 cluster group 上，以解決 combination loop 和時序不滿足的問題，此外我們也提出將同一套硬體用於處理不同類型的資料增加不同類型的 reuse，大大地增加了 Energy-efficient 的效果。

B. Performance Comparison

由於 Eyeriss v2 的一個 PE 中放了 2 個 MAC 所以使用率無法達到 100%，但我們的加速器每個 PE 只放了一個 MAC，雖然一個 cycle 能處理的運算少了一倍，但在高稀疏率的模型下，Eyeriss v2 的 PE 使用率就會下降，而我們加速器的 PE 可以一直維持在 100%的使用率，並且有較少的能耗，相較於推理速度，我們更專注於低功耗的表現。在 DRAM 中除了儲存模型本身的 weights 外，原始的架構還需要額外多出儲存 output activation 的空間以及在 DRAM 的讀寫，但在我們的設計中，資料在從 psum SRAM Bank 讀取出來後會直接壓縮並直接進行後續 ReLU 以及 Pooling 的運算並存回 Iact SRAM 中，這樣就務需要在讀寫 DRAM 造成大量的能耗了。

七、結論

我們改良了 Eyeriss v2 的架構，並且引入 im2col+GEMM 的數據重構方式，同時也在外部擴增 Encoder 讓整體系統更加靈活且泛用，同時也提出了在 PE 中交換資料類型的儲存模式。我們透過 OpenROAD 平台從 Synthesis、Floorplan、Placement、CTS、Routing 等一系列 cell-based design flow 到最後的 GDSII，在 NanGate 45nm 的製程下，最後 layout 的面積為 2.34 mm^2，並且只使用 1512k 個 NAND-2 來實現，相對 Eyeriss v2 少了 0.57X，在相同 MAC 數下 AlexNet 和 MobileNet 的推理速度分別可以達到 335.6 以及 1559.7Inference/sec 這相對於 Eyeriss v2 分別快了 1.2X 和 1.06X，且 power 只有 241.2mW，這比 Eyeriss v2 在最低功耗的神經網路(Sparse AlexNet)上運算還要低了 0.42X。

A. Pros and Cons

模型的稀疏率會很大程度地影響推理的速度與能耗，我們可以透果各種技巧提高模型的稀疏率，在 AlexNet、VGG 這類大模型很容易在維持準確率的前提下將稀疏率提高到 90%以上，這在我們的加速器中可以得到非常有效的加速，但在 MobileNet 這種結構精巧的小模型下，相對較困難將稀疏率提高，在 Sparse ratio 較小的情況下，加速的效果也就相對比較有限，但在我們加速器這種靈活的 HM-NoC 架構下，仍可以使用在不同架構的 PE 上，因此我們也可以只重新設計 PE 的架構。我們在 GLB 中使用 0 來當作 data stream 的結尾偵測，且配有 LUT 來記錄 data stream 的起始位址，這樣的作法雖然增加了一些儲存空間，但就不需要額外的控制邏輯(counter)與 Top-level 的配置來控制資料讀取的數量，當有需要更改參數配置時也不需要調整上層控制的邏輯，也讓整體的擴展更加靈活。此外我們也將整個 NoC 以 systolic array 的架構改良，以解決 logic loop 的問題以及實現真正線性擴展的特性。

B. Future work

我們的加速器在特定情況下仍會有一些瓶頸，例如當用來加速稀疏率較低的模型時會因為使用 CSC 資料格式的 PE 架構已經固定無法調整而導致壓縮的效果不佳。

我們之後會持續參考一些得到傑出成果的加速器架構[10][11]，並且繼續優化與改良加速器的架構，此外也會將我們的加速器用來運算其他種類的神經網路，以進一步分析與改進，同時也會嘗試利用開源的 Rocket chip(RISC-V)把整個加速器整合到 SoC 中，我們也預計會將加速器應用在 SSD-MobileNet (Single Shot Detector)中[12]，來實際應用在物件偵測上，以此來驗證加速器實際的應用價值。

八、 參考文獻

[1] Y. -H. Chen, T. -J. Yang, J. Emer and V. Sze, "Eyeriss v2: A Flexible Accelerator for Emerging Deep Neural Networks on Mobile Devices," in IEEE Journal on Emerging and Selected Topics in Circuits and Systems, vol. 9, no. 2, pp. 292-308, June 2019

[2] Chen, Yu-Hsin & Emer, Joel & Sze, Vivienne. (2018). Eyeriss v2: A Flexible and High-Performance Accelerator for Emerging Deep Neural Networks.

[3] V. Sze, Y. -H. Chen, T. -J. Yang and J. S. Emer, "Efficient Processing of Deep Neural Networks: A Tutorial and Survey," in Proceedings of the IEEE, vol. 105, no. 12, pp. 2295-2329, Dec. 2017

[4] H. Wang and C. Ma, "An optimization of im2col, an important method of CNNs, based on continuous address access," 2021 IEEE International Conference on Consumer Electronics and Computer Engineering (ICCECE), Guangzhou, China, 2021, pp. 314-320.

[5] S. Lym, D. Lee, M. O'Connor, N. Chatterjee and M. Erez, "DeLTA: GPU Performance Model for Deep Learning Applications with In-Depth Memory System Traffic Analysis," 2019 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), Madison, WI, USA, 2019, pp. 293-303

[6] M. Guthaus, H. Nichols, J. Cirimelli-Low, J. Kunzler and B. Wu, "Enabling Design Technology Co-Optimization of SRAMs through Open-Source Software," 2020 IEEE International Electron Devices Meeting (IEDM), San Francisco, CA, USA, 2020, pp. 41.7.1-41.7.4

[7] T. -J. Yang, Y. -H. Chen, J. Emer and V. Sze, "A method to estimate the energy consumption of deep neural networks," 2017 51st Asilomar Conference on Signals, Systems, and Computers, Pacific Grove, CA, USA, 2017, pp. 1916-1920

[8] Hoefler, T., Alistarh, D., Ben-Nun, T., Dryden, N., and Peste, A., “Sparsity in Deep Learning: Pruning and growth for efficient inference and training in neural networks”, arXiv e-prints, 2021.

[9] Howard, A. G., “MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications”, arXiv e-prints, 2017.

[10] M. A. Qureshi and A. Munir, "Sparse-PE: A Performance-Efficient Processing Engine Core for Sparse Convolutional Neural Networks," in IEEE Access, vol. 9, pp. 151458-151475, 2021

[11] Y. Ma, Y. Cao, S. Vrudhula and J. -s. Seo, "Optimizing the Convolution Operation to Accelerate Deep Neural Networks on FPGA," in IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 26, no. 7, pp. 1354-1367, July 2018

[12] H. Ali, M. Khursheed, S. K. Fatima, S. M. Shuja and S. Noor, "Object Recognition for Dental Instruments Using SSD-MobileNet," 2019 International Conference on Information Science and Communication Technology (ICISCT), Karachi, Pakistan, 2019, pp. 1-6