國立清華大學 電機工程學系

實作專題研究成果摘要

Deep Neural Network FPGA Accelerator with GEMM-based Systolic Array Architecture

基於 GEMM 的脈動陣列架構之深度神經網路 FPGA 加速器

專題領域：系統領域

組　　別：B568

指導教授：鄭桂忠 教授

組員姓名：張敬皓、王聖翔

研究期間：114年 1月 1日至 114年 11月 1日止，共 11 個月

Abstract

This project designs and implements an efficient accelerator for the VGG-16 model on the Xilinx PYNQ-Z1 FPGA platform, targeting the CIFAR-10 image classification task. We adopt a hardware/software co-design methodology to strategically partition the inference workflow. The ARM processor on the Processing System (PS) is responsible for managing the overall control flow, data preprocessing, and the execution of the more flexible fully connected layers. The 13 computationally intensive convolutional (Conv) layers are offloaded to a dedicated hardware circuit implemented in the Programmable Logic (PL).

At the core of the hardware architecture, we transform convolutional operations into Generalized Matrix Multiplication (GEMM) using the im2col technique. A 9x2 systolic array is designed to execute the GEMM operations, enabling massive parallelism and maximizing data reuse to minimize data movement. To overcome the significant memory bottleneck—the mismatch between the large VGG-16 weights (~14 MB) and the limited on-chip BRAM of the PYNQ-Z1 (~630 KB)—we designed a two-level memory hierarchy that combines external DDR memory with on-chip BRAM. Furthermore, an input-channel-based tiling strategy is proposed. This strategy effectively reduces the size of the weight data block required for each computation to approximately 4.5 KB, allowing it to be efficiently loaded into BRAM. Additionally, optimizations such as Batch Normalization fusion and INT8 static quantization are integrated to significantly reduce memory footprint and computational complexity without a notable drop in accuracy. The complete design was implemented and validated on the PYNQ-Z1 board, yielding the following results:

- **Performance:** The hardware accelerator achieves an equivalent throughput of **1.417 GOPS** (Giga Operations Per Second) at a clock frequency of 100 MHz.
- **Latency and Throughput:** The end-to-end latency for processing a single CIFAR-10 image is **417 ms**, corresponding to a throughput of approximately **2.4 FPS**. The hardware (PL) execution of the 13 convolutional layers takes only **221 ms** (53% of the total time), successfully shifting the computational bottleneck from the hardware-accelerated layers to the software components.
- **Energy Efficiency:** The system exhibits an average power consumption of **1.708 W**, resulting in a high energy efficiency of **0.829 GOPS/W**, demonstrating the significant advantage of FPGAs in low-power applications.
- **Accuracy:** The final hardware model achieves an accuracy of **85.64%**, showing negligible degradation compared to the original 32-bit floating-point model's accuracy of 85.56%. This validates the correctness of our quantization scheme and hardware implementation.

摘要

本專題目標為在資源受限的 Xilinx PYNQ-Z1 FPGA 平台上，設計並實作一高效能、低功耗的深度神經網路硬體加速器，並以 CIFAR-10影像分類任務上的 VGG-16模型進行驗證。為此，我們採用了軟硬體協同設計(Hardware/Software Co-design)的策略，將推論流程中不同特性的任務進行劃分，由處理系統(PS)端的 ARM 處理器負責整體流程控制、資料預處理、以及彈性較高的全連接層運算；而計算最密集的13層卷積運算卸載至可程式化邏輯(PL)端的專用硬體電路中執行。

在硬體架構的核心，我們將卷積運算透過 im2col 轉換為通用矩陣乘法(GEMM)，並利用一個9x2的脈動陣列(Systolic Array)架構來實現大規模的平行運算，最大化資料的重複使用性並減少資料搬運。為解決 PYNQ-Z1有限的 BRAM 容量(約630 KB)與 VGG-16龐大權重(約14 MB)之間的矛盾，我們設計了結合外部 DDR 記憶體與片上 BRAM 的二級記憶體階層，並提出基於輸入通道的 Tiling 策略。此策略將每次運算所需的權重資料塊縮減至約4.5 KB，使其能高效地載入 BRAM，成功克服了記憶體瓶頸。此外，我們整合了批次正規化(Batch Normalization)融合與 INT8靜態量化技術，在不顯著影響模型精度的前提下，大幅減少了記憶體佔用與運算複雜度。在經過完整的設計、實現與上板驗證，我們達到以下成果：

- **效能：**在100 MHz 時脈下，硬體加速器實現了**1.417 GOPS** (每秒十億次操作)的等效運算通量。
- **延遲與吞吐率：**處理單張 CIFAR-10影像端到端(End-to-End)延遲為**417 ms**，吞吐率約為**2.4 FPS**。其中硬體(PL)執行13層卷積運算僅需**221 ms**，佔總時間的53%，成功將運算瓶頸從卷積層轉移至軟體處理部分。
- **能源效率：**系統平均功耗為**1.708 W**，能源效率高達**0.829 GOPS/W**，充分展現了 FPGA 在低功耗應用場景下的顯著優勢。
- **準確率：**最終硬體模型的準確率達到**85.64%**，與原始32位元浮點數模型(85.56%)相比幾乎沒有損失，驗證了我們所採用的量化與硬體實現方案的正確性。

本專題的實測結果證明，我們成功地在 PYNQ-Z1平台上實現了一個完整的 VGG-16推論加速系統。不僅驗證用脈動陣列做 GEMM 在 FPGA 上進行 CNN 加速的可行性與高效性，也為未來在邊緣裝置上部署更複雜的神經網路提供了堅實的基礎與可行的參考方案。

目錄

1. Introduction

1.1 Motivation and Background

隨著人工智慧 (AI) 技術的飛速發展，卷積神經網路 (Convolutional Neural Networks, CNNs)的深度也逐年增加，其龐大的模型參數與密集的矩陣運算，對運算平台的性能造成挑戰。而在功耗、體積和成本受限的嵌入式裝置上高效部署 AI 推論 (Inference) 的需求也隨之上升，所以為 CNN 開發專用的硬體加速器成為我們的研究方向。

FPGA 由大量可程式化的邏輯單元、記憶體區塊和 DSP 運算單元組成，其核心優勢在於硬體可重構性與高度客製化的平行架構。且 FPGA 沒有複雜的通用控制單元，因此能以更低的功耗完成相同的運算量，其能源效率 (GOPS/Watt) 往往優於 GPU，非常適合對功耗敏感的邊緣裝置。憑藉其在效能、功耗、延遲和開發靈活性之間的絕佳平衡，成為在邊緣端實現高效能、低功耗 CNN 推論加速器的理想平台。

1.2 Purpose & Method

本專題參考[1]的加速器架構並搭配我們自行設計的資料流，目標為設計、實作並驗證一個基於 PYNQ FPGA 平台的 VGG-16 硬體加速器。為了達成此目標，我們將專注於以下幾個具體的執行項目：

1. 設計高效率的 CNN 硬體加速架構：

分析 VGG-16 模型中的運算瓶頸（主要是卷積層），設計一個能夠高度平行化處理卷積運算的硬體架構。此架構將著重於優化資料流 (Dataflow) 與利用晶片內部記憶體 (On-chip Memory)，以最大化運算效率並減少外部記憶體存取的延遲。

2. 在 PYNQ 平台上實現軟硬體整合系統：

使用硬體描述語言 (HDL) 在 FPGA 的可程式化邏輯 (PL) 實現設計的加速器。接著使用 Vivado 將其封裝為一個 PYNQ Overlay，並在處理系統 (PS) 端利用 Python 語言開發對應的驅動程式，建立一個完整的軟硬體協同工作系統。

2. Related Technologies and Works

2.1 Convolutional Neural Network (CNN)

表 1 卷積神經網路架構

Batch Layer	Layer	Input Channel	Output Channel	Input Fmap Size
1	Conv, BN, ReLU	3	64	32
2	Conv, BN, ReLU	64	64	32
	Max Pooling			
3	Conv, BN, ReLU	64	128	16
4	Conv, BN, ReLU	128	128	16
	Max Pooling			
5	Conv, BN, ReLU	128	256	8
6	Conv, BN, ReLU	256	256	8
7	Conv, BN, ReLU	256	256	8
	Max Pooling			
8	Conv, BN, ReLU	256	512	4
9	Conv, BN, ReLU	512	512	4
10	Conv, BN, ReLU	512	512	4
	Max Pooling			
11	Conv, BN, ReLU	512	512	2
12	Conv, BN, ReLU	512	512	2
13	Conv, BN, ReLU	512	512	2
	Max Pooling			
Fully Connected Layer		512	10	1

我們的模型架構基於著名的 VGG16模型並針對訓練需求進行微調，模型主要包含13個卷積層及1個全連階層(Fully Connected Layer)。我們的硬體加速器會依序執行每一個批次層(batch layer)的計算，並把輸出結果傳回軟體端執行全連階層運算，完整網路架構如表 1。

2.2 INT8 Quantization

參考[1]， 我們採用後訓練靜態量化（Post-Training Static Quantization），將模型參數從浮點數(fp32)量化為8位元整數(int8)， 我們原始使用 fp32的參數訓練的模型在 CIFAR-10測試集上的準確率為85.56%，而將模型量化為 int8後在測試集上的準確率為85.47%，顯示將模型量化並不會損失很多的精度，並且可以加速硬體運算和減少所需記憶體空間。

2.3 GEMM and Tiling Strategy

參考[1]，我們使用 GEMM(Generalized Matrix Multiplication)取代滑動視窗(sliding windows)進行卷積運算。把特徵圖中被權重核心覆蓋的資料，以及權重核心本身從原本的二維轉成一維的行與列，並拼接成特徵圖矩陣和權重矩陣，然後進行矩陣乘法得到卷積運算的結果。

GEMM 的優點是可以搭配 FPGA 上的 DSP 資源用脈動陣列(systolic array)進行大量的平行運算達到極高的數據吞吐量，是此專題主要用來加速卷積運算的核心方法。但缺點就是把特徵圖透過 im2col(image to column)方式轉為矩陣會面臨數據膨脹的問題，在 FPGA 有限的記憶體空間上造成挑戰。

im2col 會將三維的輸入特徵圖張量 $I[C][H][W]$轉為一個二維的矩陣 $I[CRS][PQ]$，並把四維的權重 $W[K][C][R][S]$轉為二維的矩陣 $W[K][CRS]$。在我們的卷積層運算中，特徵圖輸入和輸出的尺寸相同($PQ = HW$)，而權重尺寸(RS)皆為3×3，所以將輸入特徵圖透過 im2col 展開，資料量會膨脹成 RS=9倍，導致無法將所有的輸入資料存在有限的 BRAM 中，若將資料存在外部的 DDR 記憶體，又會頻繁讀寫造成額外的功耗和延遲，所以我們參考[1]採用分塊策略(tiling strategy)，並來處理這個問題。

首先我們直接把所有輸入特徵圖 $I[C][H][W]$的數據存入 BRAM，然後一次從 DDR 記憶體讀取同個輸入通道$[C]$的所有權重資料 $W[K][R][S]$到 BRAM。接下來透過 im2col 把輸入特徵圖同個通道中的 $I[H][W]$展開成 $I[RS][HW]$，而權重會轉換成 $W[K][RS]$，在 $PQ = HW$ 的情況下，運算所需資料量從 $W[K][CRS]\cdot I[CRS][HW]$變成 $W[K][RS]\cdot I[RS][HW]$，輸出都是 $O[K][HW]$，但是需要存取在 BRAM 的權重從 $W[K][CRS]$變成 $W[K][RS]$，輸入特徵圖從 $I[CRS][HW]$變成 $I[C][HW]$，分別省 C、RS 倍，使我們可以將運算所需的數據存進 BRAM 中，解決 BRAM 空間不足以及頻繁從 DDR 記憶體搬運數據的問題。

3. System Architecture and Design

3.1 PS / PL Collaboration

3.1.1 任務劃分

我們將推論流程劃分為以下兩部分：PS 端運行嵌入式 Linux 作業系統以及 Jupyter Notebook Python 環境，主要負責控制密集與需要高度靈活性的任務：

- **系統控制與配置:** 透過 PYNQ Python API 載入硬體 Overlay (bitstream)，並負責管理與配置整個推論流程。
- **資料管理:** 從儲存裝置讀取影像資料與預先訓練好的 VGG-16 模型權重，並將其載入至共享的 DDR 記憶體中。
- **預處理與後處理:** 影像預處理並在 PL 完成運算後，對輸出特徵圖進行後續的全連接層 (Fully Connected Layer) 運算，並輸出最終分類結果。
- **指令下達:** 透過 AXI-Lite 介面向 PL 傳遞控制指令與參數，並啟動硬體運算。

PL 端執行整個 VGG-16 模型計算最密集且結構固定的部分。我們使用硬體描述語言 Verilog 設計了一個專用的 VGG-16 加速器，其核心職責為：

- **高速卷積運算:** 以高度平行化的方式，高效執行 VGG-16 模型中的所有卷積層 (Convolutional Layer) 運算。這是模型中最耗時的部分。
- **池化層加速:** 根據設計，池化層 (Pooling Layer) 也在 PL 中一併實現，以減少資料來回搬運的開銷。

3.1.2 硬體架構與通訊界面

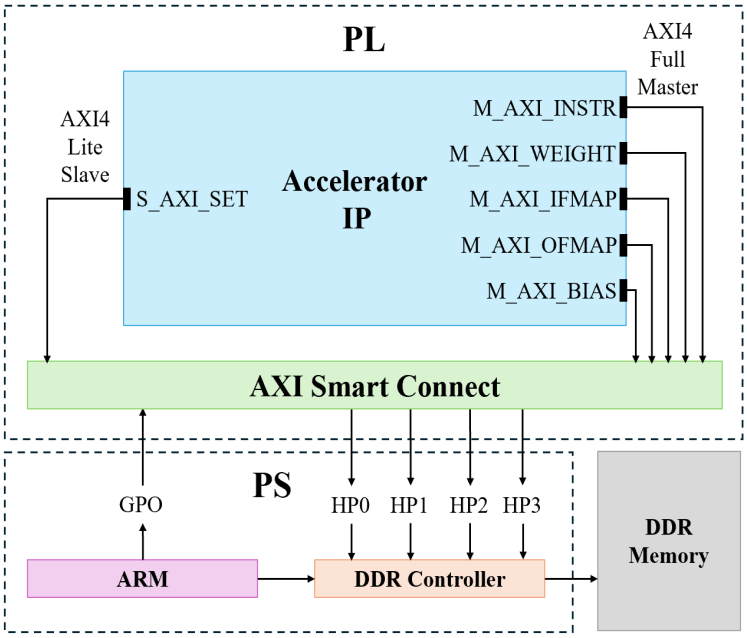

圖 1 PS / PL 系統架構圖

圖 1 詳細展示了本專題中 PS 與 PL 協同工作的硬體實現架構。整個系統的溝通是圍繞著 AXI (Advanced eXtensible Interface)匯流排協定[2]建立，它定義 PS 與 PL 之間以及 PL 內部各模組間的通訊標準。為了兼顧效能與資源利用率，我們設計了兩種不同類型的通訊路徑：低頻寬的控制路徑與高頻寬的資料路徑。

控制路徑 (Control Path)：

系統的控制流程由 PS 端的 ARM 處理器主導。當 Python 程式需要配置或啟動 PL 加速器時，會透過 AXI4-Lite 介面進行通訊。我們的 Accelerator IP 上實作了一個 AXI4-Lite Slave 介面 S_AXI_SET。這個介面主要用於傳輸少量、非密集的控制訊號與狀態參數。PS 端的 ARM 處理器可以像讀寫記憶體一樣，透過這個介面來訪問加速器內部的控制暫存器。具體操作包含：

- 傳遞儲存在 DDR Memory 中控制指令和輸入與輸出特徵圖的 Base Address。
- 傳遞 Batch Layer 的數目，PL 會根據此數字進行參數讀取以及運算。
- 發送啟動 (start) 或重置 (reset) 等控制指令。
- 讀取加速器的狀態暫存器，以確認其是否完成運算 (done signal)。

資料路徑 (Data Path)：

系統的資料路徑透過 AXI4-Full 匯流排和 PS 端的 HP (High-Performance) 連接埠實現。Accelerator IP 被設計為一個強大的 AXI Master 單元，擁有多個獨立的 AXI4 主控介面，能主動向 DDR Memory 發起讀寫請求。每個介面都有專門的用途，這種多通道設計降低匯流排的存取延遲與競爭，最大化記憶體頻寬利用率：

- M_AXI_INSTR: 讀取儲存在 DDR 中的指令，用以指導加速器執行特定層的運算，實現了指令驅動的靈活架構。
- M_AXI_WEIGHT: 專門用於從 DDR 讀取卷積層的權重 (Weights)。
- M_AXI_IFMAP: 讀取輸入特徵圖 (Input Feature Map)。
- M_AXI_BIAS: 讀取偏置值 (Bias)。
- M_AXI_OFMAP: 將運算完成的輸出特徵圖 (Output Feature Map) 寫回至 DDR Memory 的指定位置。

AXI Smart Connect & HP Ports:：

所有來自 Accelerator IP 的 AXI 介面都連接到一 AXI Smart Connect 模組。這是一個 AXI 互聯核心，將多個 AXI 主控的請求，有效地路由到 PS 端的 DDR Controller。這些連接透過 Zynq 晶片提供的四個從屬連接埠 HP0-HP3 建立，確保了 PL 能夠以極高的頻寬直接存取系統共享的 DDR Memory。

3.2 Software Overlay-Supported Design (PS)

3.2.1 自訂指令集設計

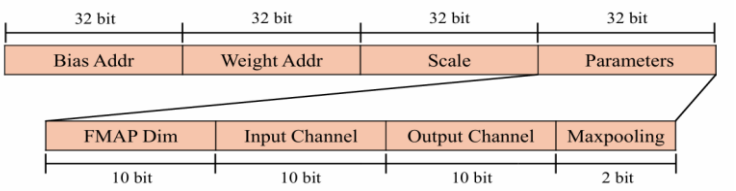

圖 2 自訂指令集架構圖

為了使加速器硬體具有間容性，我們設計了一個可自訂的指令集用於描述每一層卷積層的參數，由於 AXI-Full 協議的限制[1]，單層指令集使用128bits 分為4個 instruction field 描述，可自訂的項目如下：

- FMAP Dim Field: 有10 bits，本模型輸入與輸出特徵圖大小相等，FMAP Dim 描述長寬的 Pixel 大小，本設計最大值為32，最小值為2。
- Input Channel Field: 有10 bits，本設計最大值為512，最小值為3。
- Output Channel Field: 有10 bits，本設計最大值為512，最小值為64。
- Maxpooling Field: 有2 bits，若該 Batch Layer 需要進行 Maxpooling 運算則設為1，反之則設為0。

以上四個 Field 共同組成32bits 的 Parameter Filed。除了模型參數外，還有三個每層各異的 Field 需要設置：

- Bias Addr Field: 描述本層 Bias 在 DRAM 內的32 bits physical address，M_AXI_BIAS 會根據此值 Parameter Filed 內的值發起資料交換。
- Weight Addr Field: 描述本層 Weight 在 DRAM 內的32 bits physical address，M_AXI_WEIGHT 會根據此值與 Parameter Filed 內的值發起資料交換。
- Scale Field: 描述本層的量化參數，在 MAC 運算後再量化回8-bit 的關鍵參數。

[1] AXI4 Full 協議的 ARSIZE 參數為3bits，其十進位值 n 定義當前傳輸每個 beats 的 burst size 為2^n Bytes，故 instruction 設計為128 bits 對應 n 為3'd4

本模型 Instruction 共13層，由 PS 進行配置後為13x128 bits 的 DRAM 空間，在初始化階段 PS 須將 Instruction DRAM 的 Base Address 和 Instruction 的數量由 AXI-Lite 匯流排寫入 PL 的控制單元，M_AXI_INSTR 會在加速開始後連續讀入13條 instruction 到 BRAM 內保存。

3.2.2 PS 工作與交互流程

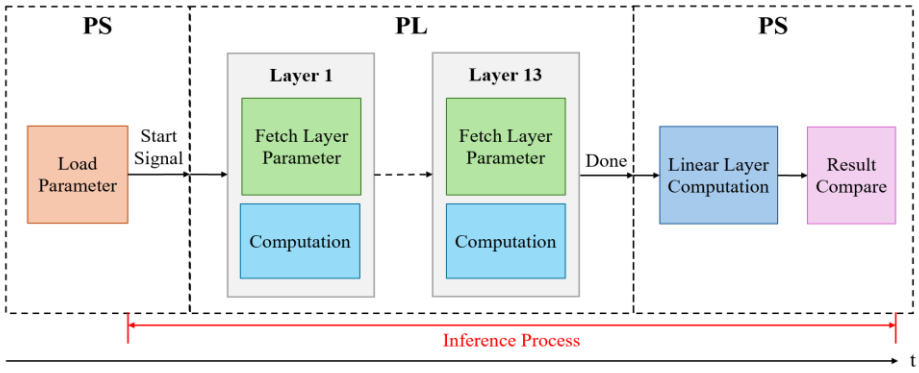

圖 3 模型推論時序圖

為了將軟硬體無縫整合，我們將整個推論過程封裝成一個對使用者友善的 Python inference 函式。當使用者呼叫此函式時，其內部詳細的工作流程如下：

- Instruction 設置： Python 程式首先將影像資料和所需的權重從 DRAM 中準備好。於 Instruction Field 記錄每層參數並配置 Instruction DRAM 空間。
- 配置控制參數：透過 AXI-Lite 匯流排寫入加速器的 S_AXI_SET 內的參數暫存器，需配置 Instruction DRAM address 及 Instruction number。以及第一層輸入特徵圖和最後一層輸出特徵圖的 DRAM address。
- 啟動加速器： 設定好該層的運算參數後，對 S_AXI_SET 內的控制暫存器發出開始訊號。此時 PS 進入會以100ms 的間隔讀取 S_AXI_SET 內的狀態暫存器判斷 PL 加速是否完成。
- 硬體平行運算：PL 加速器開始逐層進行運算，如圖一所示，讀取參數和計算的過程是平行的，逐一完成共13層 Batch Layer 的運算。在把輸出特徵圖寫回 DRAM 後，PL 控制單元會把 S_AXI_SET 內的狀態暫存器標記完成。
- 結果回傳： PS 會從狀態暫存器得知 PL 完成計算。PS 此時可以讀取 DRAM 內的輸出特徵圖空間。
- 軟體接續處理：PS 接收到硬體加速完成的資料後，繼續在軟體層面執行剩餘的全連接層運算，並最終輸出分類結果。

透過此種軟硬體協同設計 (Hardware/Software Co-design) 的模式，我們成功將需要大量平行運算的卷積層交由 FPGA 硬體處理，而將複雜的邏輯控制與彈性較高的全連接層保留在 ARM 處理器上，從而實現了效能與靈活性的最佳平衡。

3.3 Hardware Accelerator Design (PL)

整個 Top module 可分為7個主要的 module，module 的輸入輸出為了緩衝都是使用 BRAM-based FIFO generator，module 間資料交互透過 FIFO 的空、滿判斷。

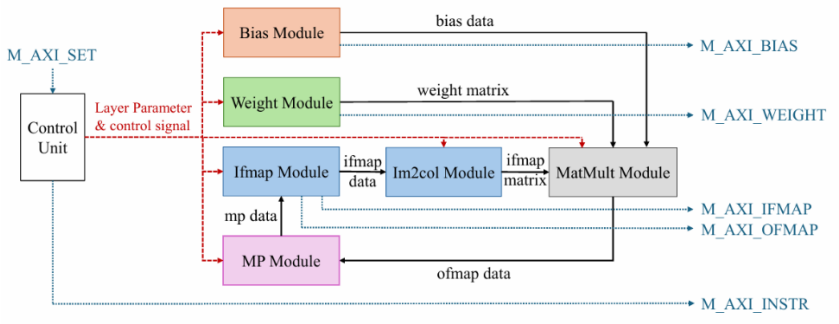

圖 4 PL 模組架構圖

Control unit 為中央控制單元，其配備 S_AXI_SET 接收來自 PS 的參數與控制訊號，並用 M_AXI_INSTR 至 DRAM 讀取 instruction，在 Decode 完後提供 Batch Layer 的參數並下達 Batch Layer 開始的訊號給其他 module。

Bias module 和 **Weight module** 功能類似，接收 Batch Layer 開始訊號後，分別透過 M_AXI_BIAS 和 M_AXI_WEIGHT 到 DRAM 讀取參數，先保存至內部 BRAM 暫存，依 MatMult module 資料需求將資料填入 Bias FIFO 和 Weight FIFO。

IFmap module 較為複雜，由於只從 DRAM 讀取第一層的 IFmap，而中間所有層的 OFmap 都透過同一塊 BRAM 保存，直到最後一層運算完才把 OFmap 寫回 DRAM。因此模組不但需要用 M_AXI_IFMAP 和 M_AXI_OFMAP 做 DRAM 的讀寫，還需要保存當前運算的 Batch Layer 從 Maxpooling module 產生的 OFmap 到 BRAM 內，做為下一層運算的 IFmap，在下一層運算時輸出到 IFmap FIFO 內。

Im2col module 將 IFmap 展開為 matrix form，使其可以放入 Systolic array 進行乘法運算。此模組會讀取 IFmap FIFO 資料進行運算後輸出至 Matrix form FIFO。

MatMult module 為矩陣乘法的核心運算單元，其內部包含一組9x2的 Systolic array 採取 IFmap-stationary-Weight-Fluding 的運算形式，將 Weight FIFO、Bias FIFO、Matrix form FIFO 的資料做 MAC(multiplication accumulation)運算，且由於 Tiling strategy 的因素，必須保存 Partial sum 於模組內部的 BRAM，運算完成經過再量化後輸出到 OFmap FIFO。

3.4 Core Computation Module Architecture

圖5展示了本加速器的核心運算模組，此模組是所有密集運算的核心，其周邊的 FIFO 與資料通路 (Datapath) 元件皆為其高效運作而服務。本設計的核心思想是將完整的卷積運算，拆解為逐一輸入通道 (Input Channel) 的小型矩陣乘法，並在 BRAM 中累加其中間結果 (Partial Sum)。

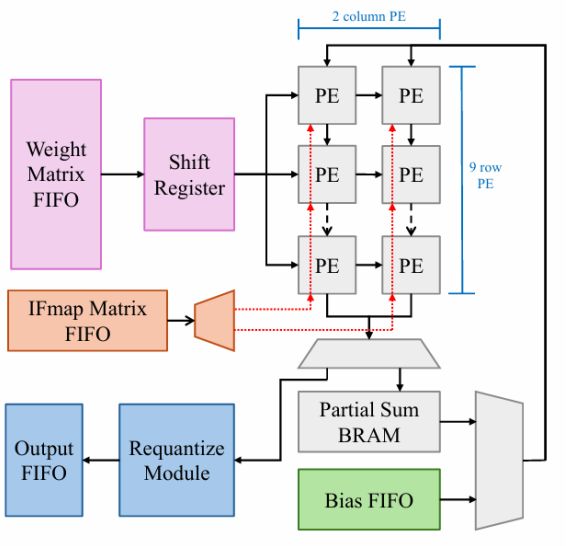

圖 5 MatMult. Module 架構圖

3.4.1 Systolic Array Design

(1) Row (列) 數 = 9: SA 的列數被設計為 9，這直接對應了 3x3 卷積核 (Kernel) 在 Im2col 轉換後被拉直的向量維度。

(2) Column (行) 數 = 2: SA 的行數被設定為 2。這個數字是根據 VGG-16 所有卷積層特徵圖寬度 (32, 16, 8, 4, 2) 的最大公因數所決定。此設計確保了無論在哪一層，整個特徵圖的運算都可以被完整地切分為以 2 個像素為單位的計算任務，避免了複雜的邊界處理，最大化了硬體的運算效率。

3.4.2 Dataflow and Computation Process

完整的 OFMAP (Output Feature Map) 產出過程，是透過 C 次(C 為輸入通道數)迭代累加完成的。每一次迭代處理一個輸入通道下的權重 (Weight) 與特徵圖 (IFmap) 資料。

(1) 任務切分：一個完整的 OFMAP 計算，被精細地切分為 C * (H*W)/2 次更小的矩陣乘法分塊。每一個分塊的計算本質為 W[K][9] * I[9][2] = O[K][2]，其中 K 是輸出通道數。

(2) 管線化運算：Weight 與 IFmap 數據從各自的 FIFO 被送入 SA。經過 9 個週期的初始延遲後，SA 進入高效的管線化運作狀態，能在接下來的 K 個週期內，連續地產出 K 個 O[K][2] 的部分和 (Partial Sum) 結果。

3.4.3 Efficient Partial Sum Accumulation with Bias Injection

本設計的一個關鍵優化在於部分和 (Partial Sum) 的處理方式。首次運算 (第一個 Channel): 在處理第一個輸入通道時，Partial Sum BRAM 是空的。此時，Bias FIFO 中的偏置值會被直接作為初始值送入累加單元。這個設計將偏置加法無縫地整合進了主運算迴圈中，省去了在所有通道運算結束後，還需要一個額外的加法階段，節省了硬體資源與運算週期。

後續運算(其餘 Channel)從第二個輸入通道開始，先前儲存在 Partial Sum BRAM 對應位置的部分會被讀出，送回累加單元與當前通道的計算結果累加。

3.4.4 Requantization and ReLU Activation

當最後一個輸入通道的運算完成後，儲存在 Partial Sum BRAM 的便是 32-bit 的最終累加結果。這些結果會被送入後處理流水線，最終的 OFmap 結果以 2 個 32-bit 像素為一個單位，按照 k→w→h (通道優先) 的順序被讀出。

Requantize Module 內含一個 5 級管線化的 32-bit 乘法器，負責將 32-bit 的累加值乘以一個量化縮放因子 (scaling factor)實現再量化。最終取其結果的高 8-bit (MSB) 作為下一層所需的 8-bit 特徵圖輸出。

4. Implementation Results and Performance Evaluation

4.1 Resource Utilization

本專題用的 FPGA 版為 PYNQ-Z1，並以 Xilinx Vivado 2025.1作為設計平台。表2的資訊為經過合成(Synthesis)及佈局繞線(Implementation)，得到的硬體資源使用報告。

表 2 Implementation Result

Resource	Utilization	Available	Utilization
LUT	19463	53200	36.58 %
LUTRAM	4094	17400	23.53 %
FF	24909	106400	23,41 %
BRAM	89.50	140	63.93 %
DSP	27	220	12.27 %

BRAM 的使用率達到了 **63.93%**，較高的 BRAM 使用率證明我們有效地利用晶片上的記憶體資源，以最大化地減少對外部 DDR 的存取延遲。

DSP Slice 的使用率目前為 12.27%，它們被用於建構 Systolic Array 的核心處理單元(PE)。目前的 DSP 使用率較低，主要因為現階段的實作僅包含一個 Systolic 運算核心。

4.2 Performance and Accuracy Evaluation

4.2.1 Latency and Throughput Analysis

我們使用 CIFAR-10 的一萬張標準測試影像對此系統進行測試，最終的效能與準確率結果統計如表 3 與表 4 所示。

表 3 CIFAR-10 Hardware Model Test Result

Total Images Processed	10000
Correct Predictions	8564
Accuracy	85.64 %
Total Execution Time	4173.72 s
Avg. Time per Images	417 ms
Total Hardware Time	2213.84 s
Avg. Hardware Time	221 ms
Hardware Time Portion	53 %

從表 3 的數據中，我們可以得出以下結論：

1. **整體效能：** 系統處理單張影像的平均端到端 (End-to-End) 延遲為 **417 ms**，對應的推論吞吐量約為 **2.4 FPS** (Frames Per Second)。

2. **軟硬體時間佔比分析：**軟體執行的部分 (PS 端) 佔據了總耗時的 47%。而硬體加速器 (PL 端) 在 53% 的時間內，完成計算量最為龐大的 13 層卷積運算。在加速後系統的瓶頸已從卷積運算轉移至軟體處理與資料通訊部分。

4.2.2 Accuracy Analysis

我們比較三種不同模型的表現：原始的 32-bit 浮點數模型、在 PyTorch 中模擬的 8-bit 量化模型，以及我們最終在 PYNQ 板上實測的硬體模型。

表 4 Accuracy Comparison

	Original Model	Quantized Model	Hardware Model
Accuracy	85.56%	85.47	85.64
Accuracy Difference	-	-0.09%	+0.08%

一萬張影像的測試基數足以讓我們得出結論，**我們的硬體加速器在功能上是完全正確的，其準確率表現與軟體模擬的量化模型幾乎沒有差異**，成功地在硬體上復現了模型的預測能力。

4.3 Computation Performance and Power Efficiency Analysis

4.3.1 Computation Analysis

對於卷積神經網路而言，核心運算為乘加運算 (Multiply-Accumulate, MAC)。對於一個核心 (Kernel) 大小為 3x3 的標準卷積層，總 MAC 數可由以下公式計算：

#MAC = 9×Input_Channel(C)×Output_Channel(K)×Fmap_Dim(H)×Fmap_Dim(W)

根據此公式統計模型中 13 個卷積層，共需執行約 3.13 億次的 MAC 運算。數據如表 5 所示。

表 5 各層 MAC 數量計算

Batch Layer	Output channel (K)	Input Channel (C)	Fmap Dim (H, W)	#MAC
1	64	3	32	1769472
2	64	64	32	37748736
3	128	64	16	18874368
4	128	128	16	37748736
5	256	128	8	18874368
6	256	256	8	37748736
7	256	256	8	37748736
8	512	256	4	18874368
9	512	512	4	37748736
10	512	512	4	37748736
11	512	512	2	9437184
12	512	512	2	9437184
13	512	512	2	9437184
			Total MAC	313196544

4.3.2 性能與功耗總結

表 6 Summary Table

Clk Freqency	100 MHz
Platform	PYNQ-Z1
Latency	221 ms
GOPS	1.417
Power	1.708 W
GOPS/W	0.829 (1/W)

將 3.13 億次的 MAC 運算總量除以 212 ms 的硬體執行時間，計算出系統的等效運算效能為 1.417 GOPS。代表在 100 MHz 的時脈下，每秒能夠穩定地執行超過 14 億次乘加運算，展現了其高效的平行處理能力。

在功耗方面，系統運行時的平均功率為 1.708 瓦。我們得到了 0.829 GOPS/W 的能源效率指標。凸顯 FPGA 平台在提供高效能運算的同時，依然能維持低功耗的顯著優勢。

4.4 Comparison with Other Works

表 7 Comparison with Other Works

Work	[3]	[4]	This work
Model	VGG16		
FPGA	Zynq XC7Z045	Intel Stratix 10 GX2800	PYNQ Z1
Precision	FP 16	FP 8/16	INT 8
Clock(MHz)	150	300	100
#BRAM	486(89%)	2421(21%)	89.5(64%)
#DSP	780(87%)	8216(71%)	27(12%)
Throughput(GOPS)	137	1604.57	1.417
Power(W)	9.63	100	1.71
GOPS/BRAM/DSP/W/freq(10^{-6})	0.25	0.0027	**3.43**

表 7中展示我們與其他同樣在 FPGA 平台加速器的比較，雖然我們的絕對吞吐量(throughput)比起[3][4]小上百倍甚至上千倍，但針對所使用的運行頻率、BRAM、DSP 數量以及整體功耗量化評估 GOPS/BRAM/DSP/W/freq，我們的表現就明顯優於[3][4]的加速器，顯示我們讓加速器用有限的資源以極低的功耗運行，並且達到很好的相對吞吐量，充分發揮我們用 tiling strategy 分配 BRAM 資源，還有 systolic array 的大吞吐量以及 FPGA 的低功耗這三項優勢，達到此專題的預期成效。

5. Conclusion

本專題在資源受限的邊緣計算平台上部署複雜深度學習模型。完整地設計、實作並驗證了一個基於 PYNQ-Z1 FPGA 平台的 VGG-16 硬體加速系統。

在硬體架構上，我們採用將卷積運算轉換為通用矩陣乘法 (GEMM) 的策略，並利用 Systolic Array 架構進行高效能平行運算。針對 FPGA 有限的 BRAM 資源，我們提出了基於輸入通道的 Tiling 策略與記憶體階層設計，成功地在僅約 70.5 KB 的 BRAM 緩衝區內，完成了對超過 14 MB 權重模型的加速。

- **效能方面：** 在 100 MHz 的時脈下，系統實現 **1.417 GOPS** 的運算吞吐量，單張影像的端到端推論延遲為 417 ms。
- **能效方面：** 整體功耗僅為 **1.708 W**，達成了 **0.829 GOPS/W** 的能源效率，凸顯了 FPGA 在邊緣 AI 應用中的巨大潛力。
- **準確率方面：** 硬體實現模型在 CIFAR-10 測試集達到 **85.64%** 的準確率，比起原始浮點數模型幾乎沒有損失，證明量化策略與硬體設計的正確性。

6. Reference

[1] Adiono, T. a. Putra, A. a. Sutisna, N. a. Syafalni, I. a. Mulyawan và Rahmat, “Low Latency YOLOv3-Tiny Accelerator for Low-Cost FPGA Using General Matrix Multiplication Principle,” *IEEE Access*, vol. 9, pp. 141890–141913, 2021.

[2] Arm Limited, AMBA AXI Protocol Specification, ARM IHI 0022J, 2023.

[3] K. Guo et al., “Angel-eye: A complete design flow for mapping CNN onto embedded FPGA,” IEEE Trans. Comput.-Aided Design Integr. Circuits Syst., vol. 37, no. 1, pp. 35–47, Jan. 2018.

[4] Y. Ma, Y. Cao, S. Vrudhula, and J.-S. Seo, “Automatic compilation of diverse CNNs onto high-performance FPGA accelerators,” IEEE Trans. Comput.-Aided Design Integr. Circuits Syst., vol. 39, no. 2, pp. 424–437, Feb. 2020.

7. Reflection

在「專題實作(一)」的修課期間，我們廣泛閱讀 FPGA 加速卷積神經網路的相關學術論文與研究。初步確立採用 Im2col 結合 Systolic Array 的技術路線。我們定期與實驗室學長進行會議，報告我們的研究文獻並提出初步架構構想。學長們主要扮演引導與諮詢的角色，幫助我們確立一個清晰且可行的基礎目標。

利用暑假期間進行練習。從零開始熟悉 PYNQ-Z1 開發板環境，並透過數個 AXI DMA 的練習專案，掌握 PS 與 PL 之間的關鍵數據通訊方式。這段前期的基礎能力建構，為後續正式專題的順利推進奠定了堅實的基礎。

我們將專題劃分為模型量化、硬體架構設計、RTL 實作、軟硬體整合等，在設計過程中，我們獨立完成架構的提出到實作。遇到技術瓶頸時，我們會主動與實驗室學長尋求解決方向，他們提供第三方視角，並尊重我們的設計決策，這種開放的討論氛圍激勵了我們獨立思考並最終克服了技術難關。