AI on Chip 2024

Final - MobilenetV3 Accelerator

REPORT

Student name: \_胡家豪、蔡明翰、王文楷、陳冠穎、黃雍翔

Student ID: N26122246、N26120579、N26121591、N26121622、N26122238

目錄

[一. Software 3](#_Toc169647256)

[1. Model introduction 3](#_Toc169647257)

[2. Model Training 4](#_Toc169647258)

[3. Quantization and Hardware mapping 5](#_Toc169647259)

[4. Hardware mapping 7](#_Toc169647260)

[5. Golden generation 7](#_Toc169647261)

[6. Software Result 8](#_Toc169647262)

[二. Hardware 9](#_Toc169647263)

[1. System Overview 9](#_Toc169647264)

[2. System Architecture 10](#_Toc169647265)

[3. Network on Chip 11](#_Toc169647266)

[4. Processing element 12](#_Toc169647267)

[5. Finite state machine diagram 14](#_Toc169647268)

[6. Programable Controller 15](#_Toc169647269)

[7. Horizontal buffer 16](#_Toc169647270)

[8. Vertical buffer 17](#_Toc169647271)

[9. Memory mapping 18](#_Toc169647272)

[10. Improvement: 19](#_Toc169647273)

[11. h-swish module 22](#_Toc169647274)

[三. Analysis 23](#_Toc169647275)

[1. Computation Utilization 23](#_Toc169647276)

[2. Performance 24](#_Toc169647277)

[3. Time vs Activity 25](#_Toc169647278)

[四. Share your thoughts 26](#_Toc169647279)

[五. Demo and Source code link 27](#_Toc169647280)

1. Software
2. Model introduction
3. MobileNetV3 overview

本次實作中，我們以mobilenetV3 為目標進行實作，其主要是由pointwise convolution進行運算，並且使用了SE layer與h swish增加精確度。

一張含有 電子藍, 行, 圖表, 貨櫃 的圖片

自動產生的描述

在原有的架構中，一個Mobilenet V3 block的流程為：pointwise conv -> depthwise conv -> SE layer -> pointwise conv。

原始的MobileNet V3中，總共含有11個這樣的block ，並且每一層這樣的block都可以自由選擇是否要用H-swish或是ReLU作為activation function以及是否需要SE layer。

1. Pointwise and Depthwsie

作者使用了Pointwise 加上 Depthwise來取代原有的Convolution，下方左圖為Depthwise conv之示意圖、右側則為Pointwise之示意圖

一張含有 鮮豔, 螢幕擷取畫面, 圖形, 正方形 的圖片

自動產生的描述 一張含有 螢幕擷取畫面, 鮮豔, 圖表, 像素 的圖片

自動產生的描述

作者透過使用Depthwise 作為提取特徵、使用pointwise 作為特徵混和，以此來取代原本的Convolution。

1. SE layer

一張含有 圖表, 行, 方案, 工程製圖 的圖片

自動產生的描述

SE layer可以看做是圖片的attention 機制，將原有的圖片藉由訓練乘上一個權重來得知哪個channel比較重要。

1. H-swish

一張含有 文字, 行, 繪圖, 字型 的圖片

自動產生的描述

Hswish是本篇用於取代原有ReLU的activation function，用於解決ReLU在負數部分消失的狀況，以此增加精確度。

一張含有 文字, 螢幕擷取畫面, 字型, 數字 的圖片

自動產生的描述

1. 一張含有 文字, 螢幕擷取畫面, 字型, 數字 的圖片

   自動產生的描述Model Training

由於原本的Mobilenet在模型建立上有較高的自由性，為了硬體的設計方便，我們將自己設計了一個Mobilenet V3 Like 的模型，與原有模型的差異在於我們的block每一層的內容都是固定的，在架構比較固定的狀況下比較好映射上硬體。

右側是我們的block架構，與原本的mobilenet block接近，差異是我們直接固定這個架構在每一層，而不能在特定層決定activation與要不要SE

一張含有 文字, 螢幕擷取畫面, 字型 的圖片

自動產生的描述

上圖是我們整體架構，我們主要使用了7個block進行模型建立。

1. Quantization and Hardware mapping
2. BN Fold

在模型訓練完畢，已經在inference的階段，由於BN層的參數均已經固定，所以我們可以將其透過下列公式併入前一層的Conv layer

一張含有 圖表 的圖片

自動產生的描述一張含有 文字 的圖片

自動產生的描述

在實作上，我們建立了一個專門用來folding BN的function，並且遍歷我們的model來進行BN fold

一張含有 文字, 螢幕擷取畫面, 字型, 軟體 的圖片

自動產生的描述一張含有 文字, 螢幕擷取畫面, 軟體 的圖片

自動產生的描述

1. Normal Quantization vs Our Quantization

在Inference-only這篇的PTQ說道，其量化可以透過以下方式進行

透過上述公式，可以將其圖像如下：

一張含有 文字, 圖表, 方案, 工程製圖 的圖片

自動產生的描述一張含有 文字, 螢幕擷取畫面, 軟體 的圖片

自動產生的描述

他可以這樣做是因為一般模型都是使用ReLU進行，而ReLU在正的部分其實就是Linear的。

如果要將此方法應用到我們的模型上並不能直接應用，原因是因為hswish其實並不是一個線性函數，因此我們可以將PTQ的量化公式寫成如下形式：

由此可知，Inference-only的步驟我們必須先進行DeQuant，經過hswish後，再進行ReQuant

一張含有 螢幕擷取畫面, 圖表, 文字, 設計 的圖片

自動產生的描述流程如左圖表示。

1. SE layer Quantization

在SE layer中，由於有相乘與pooling的運算，所以也必須特別處理，下圖是我們的處理方式，在pool的後方與相乘的後方加入fake Quant(或稱Requant)，而中間的Linear則使用一般的QLinear

一張含有 文字, 圖表, 方案, 工程製圖 的圖片

自動產生的描述

一張含有 文字, 螢幕擷取畫面, 字型 的圖片

自動產生的描述

1. Hardware mapping

完成量化後，就可以開始針對量化flow將需要的運算映射到硬體上面，參考下面的圖，說明在圖片右側：

一張含有 文字, 圖表, 平行, 字型 的圖片

自動產生的描述

Software preprocess：由於我們的量化有zero point，所以會在軟體先把需要與zero point操作的部分先做好，例如Q\_wieght與Q\_input，input\_scale與weight\_scale的相乘，1/output\_scale等

PE array：在PE array中主要是執行Convolution與psum還有bias的相加

Postprocess：主要執行De\_quant與activation function與requant。

* 首先Convolution的運算會輸入量化為8 bits的權重與輸入，其結果為16 bits
* 用24 bits累加psum與bias
* 進行Dequant ，也就是乘上dequant scale，在這邊我們將scale表示乘一個8 bits的fix point，其小數點點在最左邊，並且記錄需要的位移。Dequant的輸出是一個24bit整數與8bit小數的fixed point，這個數往右上面的位移數就是結果
* 經過swish與requant，由於1/output\_scale通常是整數，所以直接乘就可以
* 最後的結果truncate掉小數部分即為解答。

對於dequant的部分圖像化為下圖：

一張含有 圖表, 行, 螢幕擷取畫面, 文字 的圖片

自動產生的描述

1. Golden generation

完成上述部分後，就可以將model的input以及weight轉換成硬體所需要的形式，我們的硬體input 與 weight存放在SRAM當中，由於使用的32 位元CPU支援LH等指令，所以需要一個Byte一個Byte的存。下面左圖是量化前的輸入、中圖是量化後的輸入、右圖則是將輸入轉換成硬體可以讀取的十六進位形式。

一張含有 文字, 字型, 螢幕擷取畫面, 功能表 的圖片

自動產生的描述 一張含有 文字, 字型, 螢幕擷取畫面, 功能表 的圖片

自動產生的描述 一張含有 文字, 字型, 螢幕擷取畫面, 設計 的圖片

自動產生的描述

1. Software Result

下面是我們的軟體結果，可以看到我們自己搭建的模型得到了88.1%的精確度；BN\_fold的模型也獲得同樣的精確度，以此驗證我們的BN 方法是正確的、最後是我們的模型經過PTQ的精確度，由於模型較深且沒有用QAT的方式train，所以最後量化誤差會不斷累積導致精確度下降。因此量化後的模型精確度為70.1%，還在堪用的範圍內。

|  |  |
| --- | --- |
| **Model** | **accuracy** |
| Our-mobilenet | 88.1% |
| BN\_fold\_our\_mobilenet | 88.1% |
| Q\_BN\_fold\_our\_mobilenet | 70.1% |

1. Hardware
2. System Overview

一張含有 文字, 螢幕擷取畫面, 便利貼, Rectangle 的圖片

自動產生的描述

整個系統的流程，CPU會透過DMA將input data和weight data從DRAM分別搬到DLA的ping-pong SRAM和weight SRAM。搬完資料後，會寫入特定address給DLA\_wrapper，DLA則會開始進行運算，CPU則會閒置，不斷去讀另一個特定address。DLA運算結束後，會在該特定位址寫值，CPU則會在該位址讀到數值後，開始將ping-pong SRAM的資料搬回DRAM。另外，power management unit，會判斷哪些component處於閒置狀態，降低其操作電壓，以減少功率消耗。

1. System Architecture

一張含有 文字, 螢幕擷取畫面, Rectangle, 正方形 的圖片

自動產生的描述

整個DLA的架構包含用來儲存weight的weight SRAM，還有Ping-Pong SRAM 用來儲存 input 跟 output feature map，我們會透過加速器的main controller控制vertical buffer和horizontal buffer分別讀取SRAM中的ifmap和weight資料，並且將資料排列成加速器支援計算的格式。

1. Network on Chip

一張含有 螢幕擷取畫面, 文字, 正方形, 設計 的圖片

自動產生的描述

Vertical buffer中的input feature map資料會以systolic的方式由上往下傳遞；horizontal buffer中的weight資料則會直接broadcast到PE array的每個PE上。每個row的psum結果會透過row adder tree進行相加，若是depthwise covolution的模式下，會將每三個row adder tree的output在經過一個adder tree進行相加。這些psum data接著會送入後面的quantizer、h-swish、re-quantizer，對資料做進一步的處理後，將psum data存入psum-buffer。

DLA Storage size

|  |  |
| --- | --- |
| Register | Size |
| Vertical Buffer | 64 Byte |
| Horizonal Buffer | 1024 Byte |
| PE Array Registers | 728 Byte |
| Psum Buffer | 8192 Byte |
| Pipeline Registers | 96 Byte |
| Total | 9.8671875 KB |

|  |  |
| --- | --- |
| SRAM | Size |
| Ping-Pong SRAM | 64 Byte |
| Horizonal Buffer | 1024 Byte |

1. Processing element

一張含有 螢幕擷取畫面, 圖表, 文字, 行 的圖片

自動產生的描述

每個PE包含1個8 bits \* 8 bits 的有號數乘法器、1個8 bits registe用來傳送ifmap資料到下一個PE和1個16bits的register暫存乘法器的output以減少critical path。並且在weight input有一個filter active訊號，這個遮蔽設計讓我們的加速器能兼容更多種kernal size大小的convolution以及避免row adder tree有錯誤的非0訊號輸入。

一張含有 螢幕擷取畫面 的圖片

自動產生的描述

因為PE array大小為16\*16，一個row adder tree需要將16筆16bits的資料進行累加，故adder tree架構需要4層。

一張含有 螢幕擷取畫面, 文字, 正方形, 設計 的圖片

自動產生的描述Critical Path

根據合成時timing analysis的report結果，我們發現電路的critical path是在adder tree到h-swish，為了解決這個問題，我們決定將這段路徑切成2 stage pipeline的架構，將delay time壓到10ns以下，並成功合成。

1. Finite state machine diagram

一張含有 螢幕擷取畫面, 圓形 的圖片

自動產生的描述

當DLA接收到dla\_start訊號，會進入build模式，並將該層layer運算的資訊給各個controller，包括ifmap、weight、ofmap的起始addr、ifmap, ofmap & kernal的size、進行運算的模式。之後根據當下是要進行depthwise convolution還是pointwise convolution進入對應的state。

Depthwise convolution: 在state CHECK\_DW，檢查和設定該次運算的資訊到control registers。設定好資訊後，進入LOAD state，depthwise convlotuiun不僅需要等待weight data ready，同時需要等待vertical buffer是否準備好一個kernal size的資料，這邊我們是直接使用fifo的full訊號，以簡化設計。只有當vertical buffer和horizontal buffer都發出data\_ready訊號後，才會進入運算的state RUN\_DW，將資料送進PE array內，並將計算完的ofmap存入psum buffer。RUN\_DW運算時，會檢查計算結果的次數和ofmap size，以拉起結束訊號dw\_all\_done，進入state STORE\_MAIN。

Pointwise convolution: 因為我們有做pipeline PE array的設計，可以容許vertical buffer沒有準備好的情況，在LOAD\_PW僅需要確認horizontal buffer有準備好，就會進入運算state RUN\_PW，邊載入資料和運算，並將結果存入sum buffer。若有超過16 channels需要計算，則會回到LOAD\_PW，準備好weight，在進入RUN\_PW。RUN\_PW運算時，會檢查計算結果的次數和ofmap size，以拉起結束訊號pw\_all\_done，進入state STORE\_MAIN。STOE\_MAIN: 將之前存在psum buffer的資料寫入ping-pong sram。

1. Programable Controller

一張含有 文字, 螢幕擷取畫面, 鮮豔, Rectangle 的圖片

自動產生的描述

因為DLA內部的module是多人進行開發的緣故，所以有4個controller分別去控制PE array、vertical buffer、horizontal buffer和psum buffer，但最主要是控制PE array的DLA controller會先從config register讀取transfer\_confog即該層layer的資訊，並將對應的資訊分派給各個controller，以generalize不同大小的和模式的convolution，達到programable的效果。

1. Horizontal buffer
2. FIFO

一張含有 螢幕擷取畫面, 圖表, Rectangle, 行 的圖片

自動產生的描述

1. horizontal buffer

一張含有 文字, 螢幕擷取畫面, 圖表, 行 的圖片

自動產生的描述

horizontal buffer 架構如圖所示，a圖的FIFO總共有16個,b圖的FIFOs的內部架構則是a圖，b圖總共會有16個FIFOs，所以我們的Horizontal buffer總共有256個FIFOs，以對應256個PE array，horizontal buffer的功能是將filter的值從memory讀出以後暫存在buffer內，當filter所需的值都從memory搬到buffer後，Horizontal buffer會向PE Array發出ready信號，當PE Array接收到ready信號以後，再根據目前所要執行的convoltion方式，向horizontal buffer發出valid信號，此時PE Array就可以讀取所需要的值。

1. Vertical buffer

一張含有 行, 圖表, 平行, 設計 的圖片

自動產生的描述 一張含有 圖表, 行, 平行, 工程製圖 的圖片

自動產生的描述

Fig1. Pointwise data access order

Fig2. depthwise data access order

Vertical controller 主要的功能是將ifmap從ping-pong SRAM拿出來，並且將資料排列成PE array 支援計算的格式，存入vertical buffer 中。

以下說明vertical controller至SRAM拿資料的順序，因為要計算pointwise 或是 depthwise 取資料與排列資料的方式都不一樣 ，因此一開始vertical controller需要接收到這層要計算的layer資訊。

首先是pointwise，ifmap 會以NHWC 的格式存放在SRAM當中，而controller 會先從channel 方向開始拿資料，如Fig1.的紫色方塊所示，因為我們的 PE array大小為16\*16，一次可以對ifmap的16個channel做計算，因此一次會先拿16個channel 的資料出來，並且以列方向排列，存入vertical buffer 中，接著會往ifmap 的寬方向移動，一樣取16個channel 的資料排列好存入vertical buffer，最後才往寬方向移動，重複上述步驟直到移動完整個 ifmap 平面，會再往後16個channel 取資料，直到拿完所有的input feature map。

再來介紹depthwise，ifmap 存放在SRAM 的格式與 pointwise 不同，是以一個convolutional block 為單位來存放，一個convolutional block 又會以NCHW的方式儲存。因為我們PE array支援一次對5個 convolution kernel 做計算，因此Controller 拿資料也是以convolutional block 為單位，差別在於每個convolutional block 的 channel 為 5，如Fig2. 的藍色方塊所示。針對convolutional block 的資料存取順序則會先以寬度方向再往 channel 方向，最後則是高度方向，這樣的順序，與 PE array 如何計算depthwise 有關，當計算完一次depthwise，會再往寬度方向移動stride的大小，存取下一塊 convolutional block，寬度方向都計算完畢，再往高度方向移動，重複上述步驟直到計算完整個ifmap 的平面後，會再往後5 個 channel 做存取，直到拿完所有的input feature map。

1. Memory mapping
2. Memory mapping-pointwise convolution

一張含有 螢幕擷取畫面, 文字, 平行, 行 的圖片

自動產生的描述

我們硬體實作的部分為pointwise convolution和depthwise convolution,在pointwise convolution的部分，我們採用的是channel first，即把filter1的第一個channel(a1)放在memory第一個位置，filter1的第二個channel(a2)放在memory第二個位置，假設第一個filter有8個channel，那memory的第8個位置就會是filter1的第8個channel的值，接者再放filter2 第一個channel的值，以此類推，範例如下圖所示，featue map也是按照channel first的方法排列。

1. Memory Mapping–Depthwise Convolution一張含有 文字, 螢幕擷取畫面, 正方形, 日曆 的圖片

   自動產生的描述

而depthwise convolution部分，我們使用的方法為row major，我們排放的順序為a1~a9,b1~b9,c1~c9…..以此類推,如下圖所示，feature map的排放方式也是 row major。

1. Improvement:

相對於參考論文的架構，我們在架構上添加了一些新的設計:

1. Ifmap data bubble tolerate design

一張含有 文字, 螢幕擷取畫面, 數字, 字型 的圖片

自動產生的描述

首先是bubble tolerate design，無論是Pointwise或Depthwise Convolution，在vertical buffer準備資料時，我們的PE是能夠pipeline將ifmap往下傳遞做計算的 (上圖紅色箭頭為傳遞方向)，但在ifmap往下傳遞的同時會有bubble出現，也就是在傳遞真正需要的data (上圖中間3個藍色矩形框起來處) 時會有不必要的data也一起往下傳遞進行運算並輸出，因此我們在電路上添加了output valid這條訊號線。

我們將output valid與資料一起往下作傳遞，這樣做除了能夠準確地將真正所需的data做輸出，還能避免輸出bubble算出來的data，並且達到pipeline的效果。

1. Optimize Depthwise Convolution - Shortcut Strategy

一張含有 文字, 螢幕擷取畫面, 圖表, 平行 的圖片

自動產生的描述

在原始論文中，PE之間傳遞data的方式是由上往下作傳遞，這在做Depthwise Convolution時，會花費較多的clock cycle數準備資料，因為此架構在Depthwise Convolution時須等到每個所需的PE都準備好資料才會做運算並輸出。比如上圖PE array裡藍色3\*3大小矩形的最下方列需要共15個cycles才能將資料從最上方的input傳遞到該位置做運算並輸出。

因此我們在架構上使用了Shortcut Strategy (PE array裡紅色箭頭)，因為添加了short cut的電路，讓原本花15 cycles的傳輸時間減少為3個cycles，大幅減少在Depthwise Convolution搬運資料所需的時間。

1. Weight active design

* Pointwise Convolution:

一張含有 螢幕擷取畫面, 文字, 圖表, Rectangle 的圖片

自動產生的描述

* Depthwise Convolution:

一張含有 文字, 螢幕擷取畫面, 圖表, 方案 的圖片

自動產生的描述

在我們的架構中，會使用Row adder tree將PE array每列16個data做加總，但若該列裡有未使用的PE，會導致Row adder tree連同不需要的data也一起加總。因此我們添加了Weight active的設計，這個設計能夠將沒有使用的PE的Weight設為0，因為weight為0與任何ifmap做乘法都為0，這麼做能夠避免 Row adder tree 將不必要的data也一起加總，上方兩張分別為Pointwise Convolution在ifmap channel = 8、ofmap channel = 8與Depthwise Convolution的Weight active design (灰色區域為Weight active)。

1. h-swish module
2. Approximation h-swish Architecture

一張含有 螢幕擷取畫面, 黑色, 設計 的圖片

自動產生的描述

1. h-swish approximation formula

一張含有 筆跡, 文字, 字型, 書法 的圖片

自動產生的描述

1. h-swish vs approximation h-swish

一張含有 文字, 行, 繪圖, 圖表 的圖片

自動產生的描述一張含有 圖表, 行, 繪圖, 文字 的圖片

自動產生的描述

我們實作的h-swish架構是參考K. Choi, S. Kim, J. Kim and I. -C. Park, "Hardware-Friendly Approximation for Swish Activation and Its Implementation" 這篇論文的作法，但我們沒有注意到本篇論文使用的數學近似式和原本的h-swish相差非常大，導致我們做出來的結果無法和軟體的結果相匹配，我們太晚才發現這個問題，來不及更正，最直接的改善方法就是如同老師給的建議:使用look up table 就能解決了!

1. Analysis
   1. Computation Utilization

Input data size: 8\*8\*1, kernal大小均是3\*3

|  |  |  |  |
| --- | --- | --- | --- |
| Layer | Computation type | Ofmap size | Utilization |
| 1 | Standard convolution | 8\*8\*8 | 14.0625 % |
| 2 | Pointwise convolution | 16\*16\*8 | 25 % |
| 2 | Depthwise convolution | 16\*16\*8 | 14.0625 % |
| 3 | Pointwise convolution | 32\*32\*48 | 50 % |
| 3 | Depthwise convolution | 32\*32\*48 | 16.875 % |
| 4 | Pointwise convolution | 32\*32\*64 | 100 % |
| 4 | Depthwise convolution | 32\*32\*64 | 17.307 % |
| 5 | Pointwise convolution | 48\*48\*64 | 100 % |
| 5 | Depthwise convolution | 48\*48\*64 | 17.307 % |
| 6 | Pointwise convolution | 64\*64\*96 | 100 % |
| 6 | Depthwise convolution | 64\*64\*96 | 16.875 % |
| 7 | Pointwise convolution | 32\*32\*48 | 100 % |
| 7 | Depthwise convolution | 32\*32\*48 | * 1. % |

1. Performance

一張含有 文字, 螢幕擷取畫面, 字型, 數字 的圖片

自動產生的描述

pointwise convolution執行cycles，實際和理想比較，以一層運算為例。

Test : ifmap size 28\*28\*8，ofmap size 26\*26\*8

Max utilization: ifmap size 28\*28\*16，ofmap size 26\*26\*16

一張含有 文字, 螢幕擷取畫面, 字型, 數字 的圖片

自動產生的描述

depthwise convolution執行cycles，實際和理想比較，以一層運算為例。

Test: ifmap size 30\*30\*8，stride 3，ofmap size 10\*10\*8

Max utilization: ifmap size 30\*30\*10，stride 3，ofmap size 10\*10\*10

一張含有 文字, 螢幕擷取畫面, 字型, 數字 的圖片

自動產生的描述

BUS transfer: 75.2MB/s

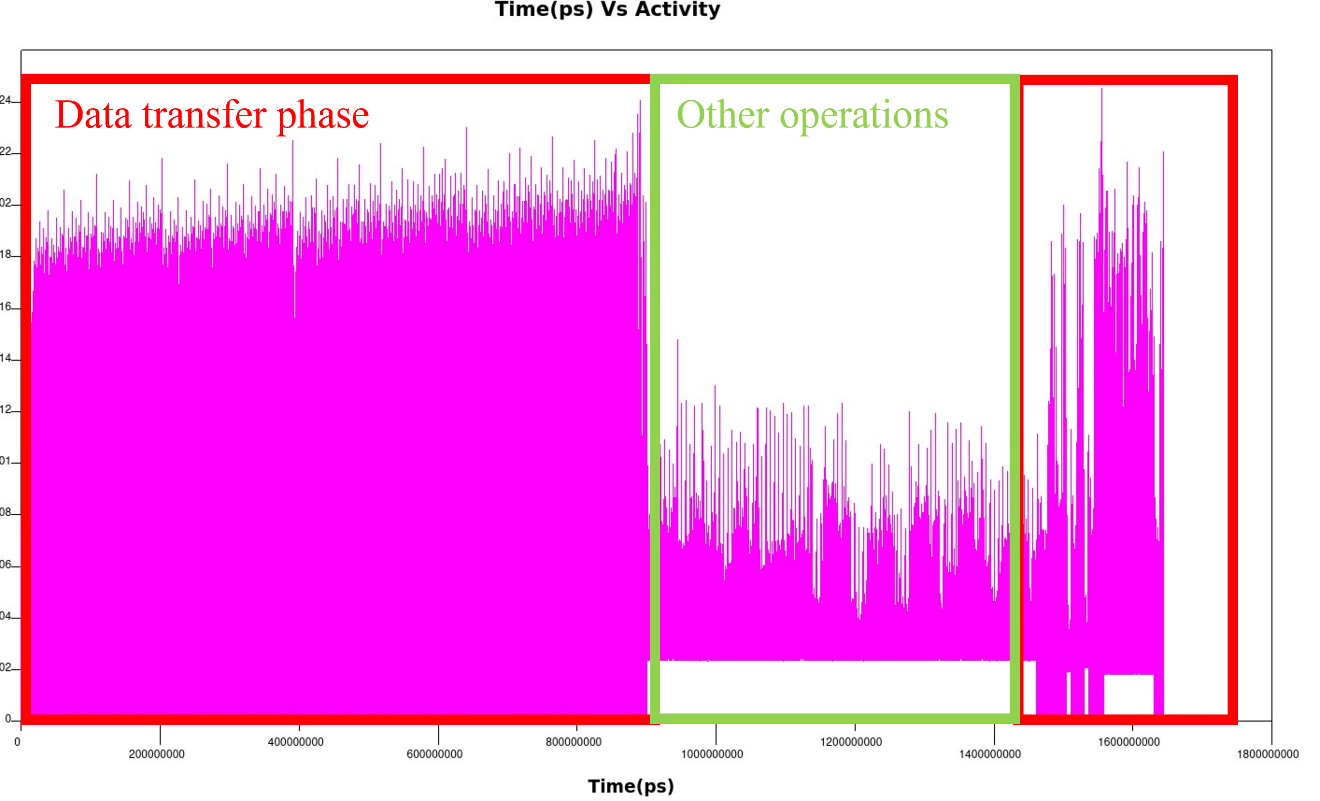
MAC：800M FLOP (28\*28\*8\*8 MAC in 62875 ns)

BUS transfer: 400MB/s(Best case)

MAC：1600M FLOP (28\*28\*8\*16 MAC )

I=75.2×106800×106=75.2800≈10.64 FLOP/byte

1. Time vs Activity



我們針對系統搬運資料的dataflow利用sypglass進行power的分析，可以在spyglass產出的report中看到，當系統在存取資料時，power的activity會比較高，而進行其他操作時，power activity 則會比較低，我們可以分析系統在哪裡有比較高的功率消耗，進而對該部分進行優化，以減少power consumption。

1. Share your thoughts
2. 胡家豪

經過了本次專題我學習到了很多，主要我是負責軟體的部分，包括模型搭建與量化。除了對量化的flaw更加認識以外，我也理解到其實在做量化有一部份就是在設計model。必須不斷的與硬體端溝通到底要做那些運算，還必須考慮到位寬到底夠不夠，會不會溢位等。雖然這次專題很累，但是收獲也很大！

1. 蔡明翰

這次的project我負責的是DLA controller和PE array的部分，在刻硬體的過程中，會發現paper跟實際在實作會有很大的出入，並且可能會因為誤解paper的意思，而不斷的進行修正。所以一開始在制定硬體規格時，可能就需要想好整個data flow如何運作，甚至可能需要透過軟體先進行模擬。另外就是軟硬體溝通的問題，在這次的project中，我們遇到很多次軟體的結果迥異於硬體，原本以為是量化或其他原因的損失，後來在用testbench確認硬體功能應該是正常的後，才發現是軟體和硬體的資料排序不同。另外就是量化的問題，原本以為量化僅僅只是將數字縮放range而已，但其實不然，量化需要可慮到硬體可以表示的範圍，以及硬體運算後的結果，其實是非常複雜的。經過這次project，算是跑過個簡單的AI加速器的flow，要programable，甚至需要compiler的輔助，以避免都人工計算，也深刻體會到，AI加速器最難的其實是整個data flow並如何想出好的控制訊號，收益良多。

1. 王文楷

在這次的final project中，我們組解決了需多困難，也收穫的非常多。首先在技術層面，我們克服了許多難題，包括硬體架構設計、以及優化我們的dataflow。每一項克服的困難都表示我們的硬體設計的實力又邁進了一步，此外，這次final project需要互相溝通才能有效的解決遇到的難題，這提升了我與他人共同合作的能力。

設計這次final project硬體的過程也讓我們體會到團隊合作的重要性，只有在組員的共同努力下，我們才能克服遇到的困難，這次經歷讓我們學到了以後在面對新的挑戰時，只要保持這種合作精神和不懈追求的態度，一定能解決問題。

1. 陳冠穎

這次的final project讓我深入了解了加速器的開發過程，從硬體架構的構想、資料運算的dataflow討論，到硬體設計的實現。實作過程中，軟硬體的整合以及硬體單元之間的溝通與合併，是我們投入最多時間的部分，透過大家合作，最後才能夠順利的完成這次的project。

1. 黃雍翔

透過這次的final project 我學習到了許多關於硬體設計相關的知識，還有設計AI加速器的基本流程，也了解到在設計不同的硬體、軟體和硬體之間需要多溝通才會更有效率的解決問題，透過這次的經驗，我了解到自身還有許多的不足，需要多努力才能增進自身的實力。

1. Demo and Source code link

Demo：<https://youtu.be/nx5mJhMWBNw>

Source code：<https://github.com/kevin199907/AOC>