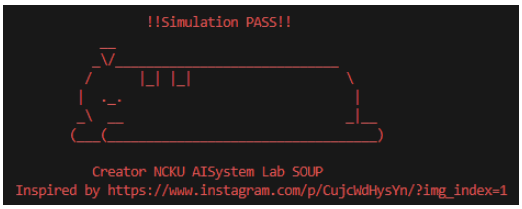
AOC 2024 Spring - Lab 3 Hybrid Multiplier

施宇庭 NN6124030

1 Implmenetation of Hybrid Multiplier

Testbench 0 Passed

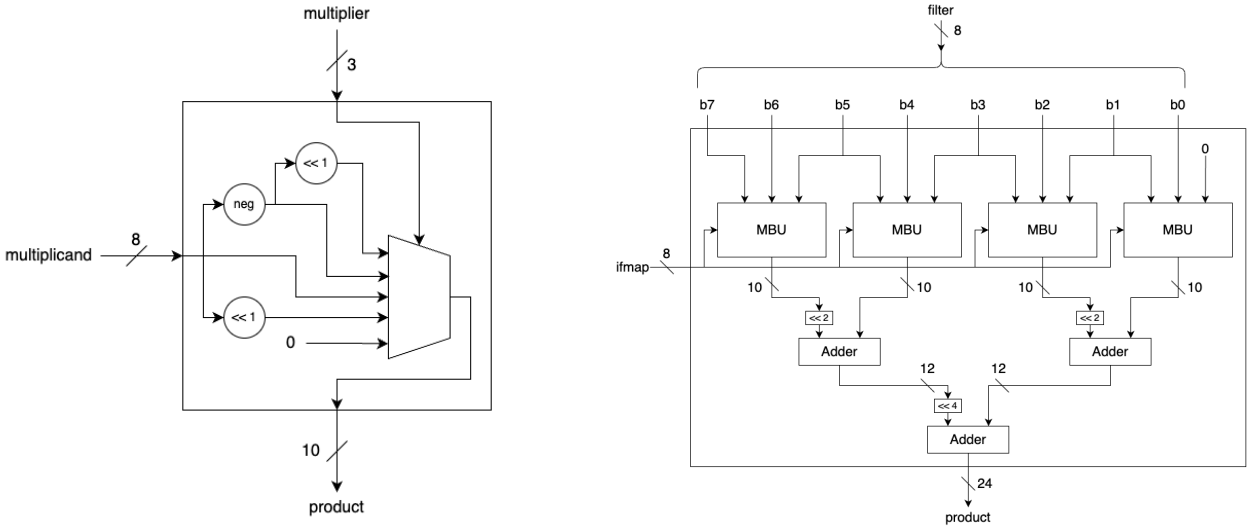


Testbench 1 Passed

Testbench 2 Passed

2 Architecture and Algorithm

這次的 hybrid multiplication 我選擇實作 Modified Booth's Algorithm，整個乘法器 (如 Figure 1(b) 所示) 可以做 8-bit 和 8-bit 的有號整數乘法，輸出為一個 24-bit 的有號整數，由四個 Modified Booth Unit (MBU) 和一個 adder tree 組成。每個 MBU (如 Figure 1(a) 所示) 由 3-bit 的 multiplier 來選擇輸出是 -2、-1、0、1 或 2 倍的 multiplicand，為了避免 overflow 輸出為 10 bits，並做 signed extension。



(a) Modified Booth Unit (MBU)

(b) Hybrid multiplier using MBUs

Figure 1: Hardware architecture of hybrid multiplier using Modified Booth's Algorithm.

3 Hardware Reuse

Figure 1(b) 的輸出可作為 testbench 0 的答案，為了可以同時支援不同 bits 數的有號整數乘法，並盡可能重用既有的硬體，Figure 2 為修改後的 datapath，添加了 ifmap 和 filter 的 preprocessing units (圖中的 Prep 和 2-to-1 MUX)，針對不同的 testbench 做 input 的前處理，四個 MBUs 的輸出個自取 [5:0] bits 拼接起來 (圖中左邊的 Concat) 即可作為 testbench 2 的答案，第一層 adder 的輸出拼接起來可作為 testbench 1 的答案，第二層 adder 的輸出做 signed extension 後則為 testbench 0 的答案。如此便成功重用了原本 Figure 1(b) 架構中的 MBUs 和 adders。

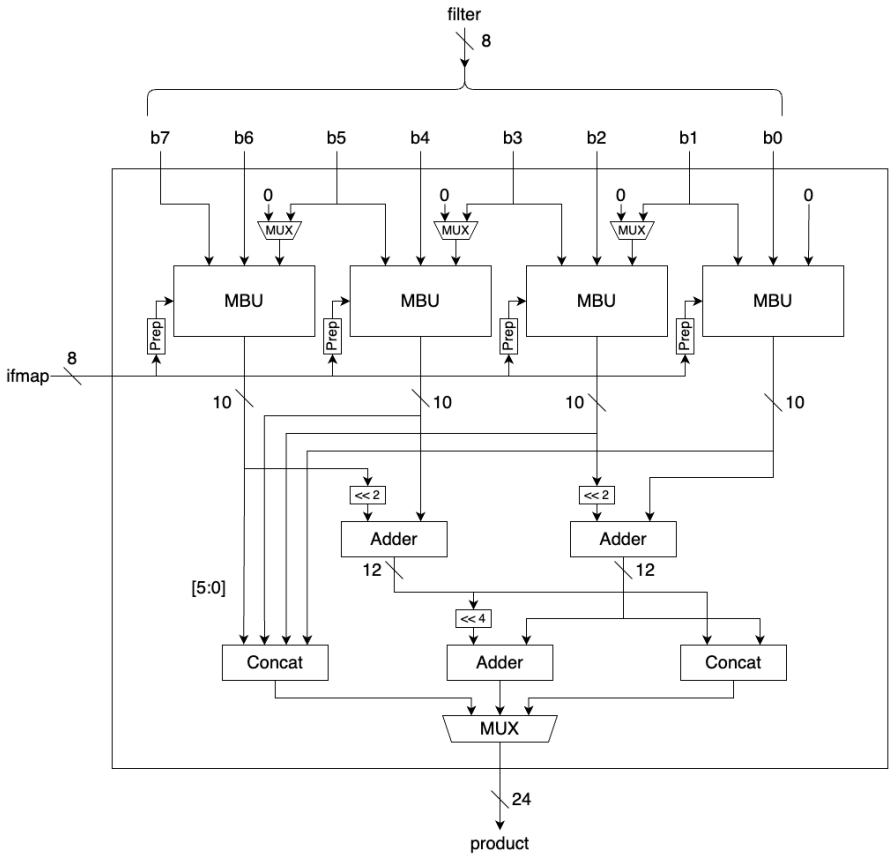


Figure 2: Hardware architecture of Modified Booth multiplier supporting 2-bit, 4-bit, and 8-bit signed integer multiplication.

4 FLOPs and MACs in the 1st Layer of VGG19-Cifar10

VGG-16 的第一層計算是 convolution，有 64 個 size 為 3x3 的 kernel，padding 為 1、padding 為 1，所以 input、output 和 kernel 的尺寸如 Table 1 所列。

Table 1: The size of input, output, and kernel of the first convolution layer of VGG16-CIFAR10.

	Batch size	Channel	Height	Width
Input	$N = 1$	$C = 3$	$X = 32$	$X = 32$
Kernel	$M = 64$	$C = 3$	$K = 3$	$K = 3$
Output	$N = 1$	$M = 64$	$Y = 32$	$Y = 32$

計算每個 output element 需要 $C \times K^2$ 個乘法，整個 output tensor 有 $N \times M \times Y^2$ 個 elements，因此總 MACs 數為：

$$\text{MACs} = (N \times M \times Y^2) \times (C \times K^2) = (1 \times 64 \times 32^2) \times (3 \times 3^2) = 1769472$$

計算每個 output element 需要 $C \times K^2$ 個乘法和 $C \times K^2 - 1$ 個加法，整個 output tensor 有 $N \times M \times Y^2$ 個 elements，因此總 FLOPs 數為：

$$\text{FLOPs} = (N \times M \times Y^2) \times (C \times K^2 + C \times K^2 - 1) = (1 \times 64 \times 32^2) \times (3 \times 3^2 + 3 \times 3^2 - 1) = 3473408$$

5 FLOPs and MACs in a 512x10 Fully Connected Layer

Table 2: The size of input, output, and kernel of a 512x10 fully connected layer.

	Rows	Columns
Input	$M = 512$	1
Kernel	$N = 10$	$M = 512$
Output	$N = 10$	1

計算每個 output element 需要 M 個乘法，整個 output tensor 有 N 個 elements，因此總 MACs 數為：

$$\text{MACs} = N \times M = 10 \times 512 = 5120$$

計算每個 output element 需要 M 個乘法和 $M - 1$ 個加法，整個 output tensor 有 N 個 elements，因此總 FLOPs 數為：

$$\text{FLOPs} = N \times (M + M - 1) = 10 \times (512 + 512 - 1) = 10230$$

6 Difference between Hybrid Multiplier Algorithms

三個乘法演算法大致上都可以分為兩個階段：

1. 從 multiplier 取出數個 bits (以下稱為 select bits)，用來決定輸入給下個階段的 partial sum 為幾倍的 multiplicand，若為有號數乘法，則產生 partial sum 的階段要做 signed extension
2. 將所有的 partial sum 加起來，輸出最後結果為 product

Robertson's Algorithm、Booth's Algorithm 和 Modified Booth's Algorithm 分別採用 1、2 和 3 個 select bits，select bits 數量越多代表第一階段會花費更多的硬體資源和計算時間，但會減少 partial sum 的數量，因此在第二階段可以節省硬體資源和計算時間。

三種演算法的差異主要體現在第一階段，與第二階段的作法的搭配是獨立的，可以使用傳統的 carry-propagate adders 或是 Wallace tree adder 等不同的作法，藉由平衡兩個階段的成本，在不同的場景下會有各自適合的演算法。以下 Table 3 為三種演算法的比較：

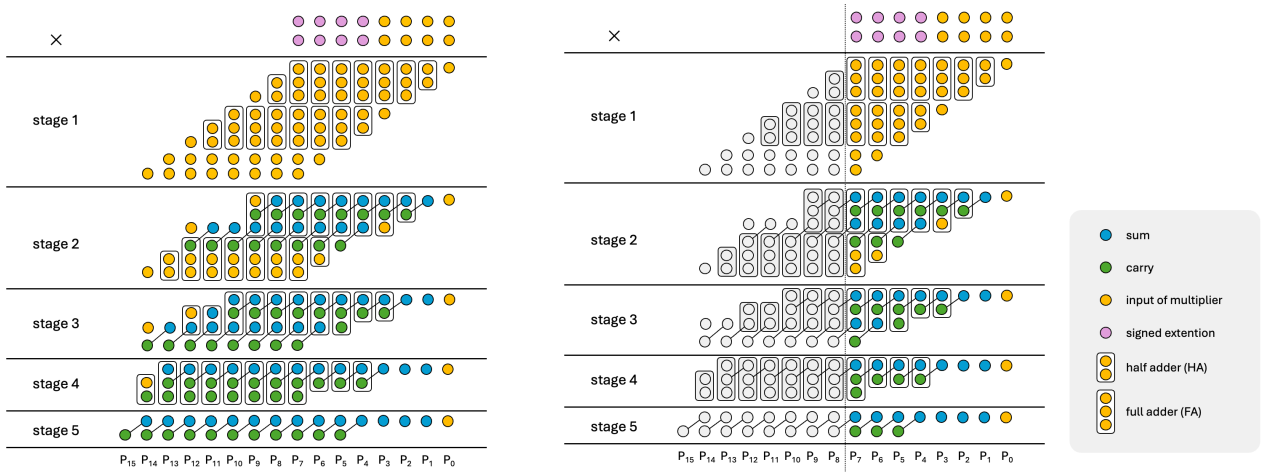
Table 3: Comparison among Robertson's, Booth's and Modified Booth's algorithms

	Robertson	Booth	Modified Booth
select bits	1 bit	2 bits	3 bits
partial sum	和 multiplier bits 數量一樣	和 multiplier bits 數量一樣	multiplier bits 數量的一半
計算時間	較長	multiplier 由連續 1/0 組成時有加速效果	減少 partial sum 數量，速度最快
硬體實作	簡單	需要 LUT，中等	需要更多 LUT，較複雜
適用場景	硬體資源受限的環境	需要快速計算的應用	高性能計算和大規模數據處理

7 4-bit Signed Integer Multiplier using Wallace Tree

Algorithm

講義中提供的 wallace tree multiplier 做的是 4-bit 無號整數的乘法，輸入為兩個 4-bit integer，輸出則為一個 8-bit integer，而 4-bit 的有號整數乘法，需要先將兩個 4-bit 的輸入擴展成 8-bit，再使用 Wallace Tree 進行乘法運算，如 Figure 3(a) 所示，其中每個小圓點是一個 bit，黃色是乘法器的輸入，粉色是 signed extension bits，藍色是加法器的 sum，綠色是加法器的 carry，帶有兩個原點的圓角方格代表半加器，三個原點的圓角方格則為全加器。



(a) Wallace Tree (b) Optimized Wallace Tree

Figure 3: Algorithm of 4-bit signed integer multiplication using Wallace Tree.

Figure 3(a) 的演算法可以完成 4-bit 對 4-bit 的有號整數乘法，但仔細觀察可以發現，輸出

的 16 bits 中其實只需要低位的 8 bits，因此如 Figure 3(b) 所呈現，左半邊的運算都不是必要的可以省略。

Architecture

根據 Figure 3(b) 可以設計出 Figure 4 的硬體，critical path 則發生在計算 bit P_7 的路徑上，如 Figure 6 所示，以紅色的線和紅色邊框的方塊來表示，其中一個全加器的 latency 等於兩個半加器的 latency 加上一個 OR gate 的 latency (如 Figure 5 所示)，vector merging 的部分需要由三個全加器構成的 carry-propagate adder (CPA)，因此在 critical path 上需要完整經過 6 層全加器和 1 層半加器的 latency。

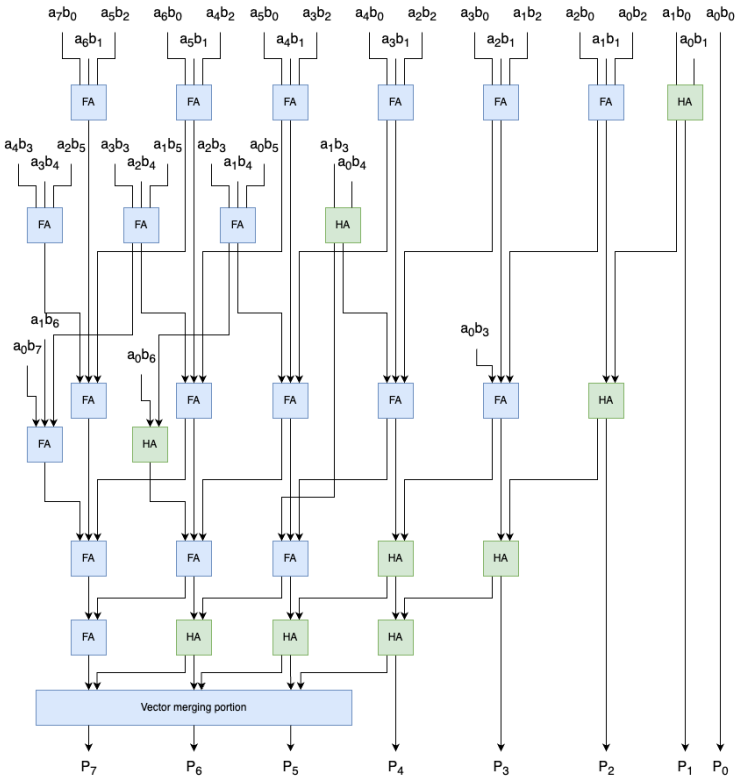


Figure 4: Hardware architecture of 4-bit signed integer multiplier using Optimized Wallace Tree.

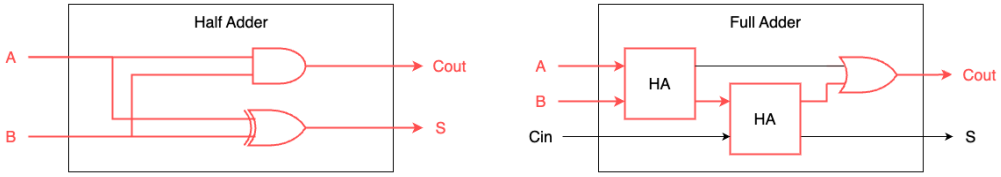


Figure 5: Critical paths of half adder and full adder (shown as red wires and blocks). The latency of a full adder is 3x of the latency of a half adder.

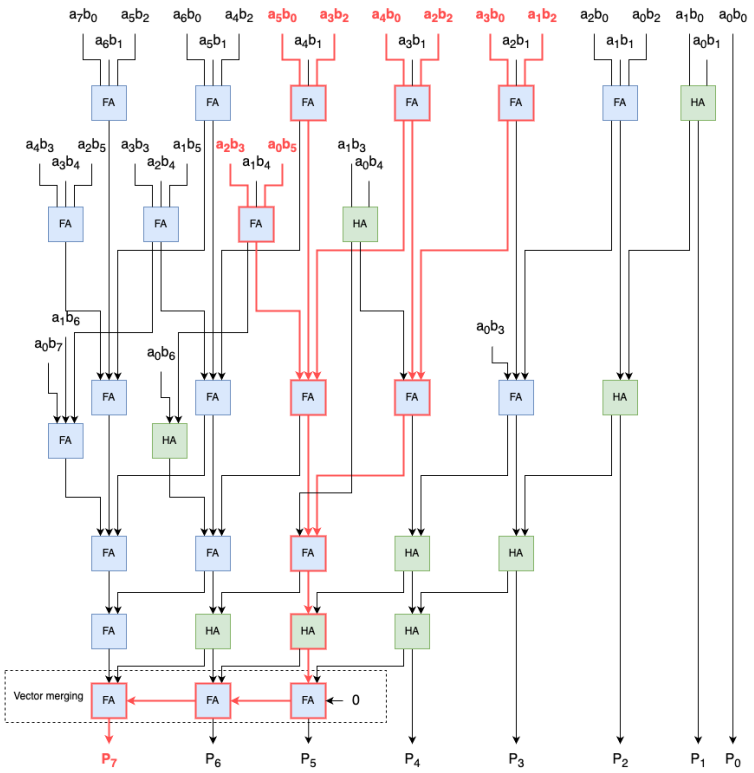


Figure 6: Critical path of 4-bit signed integer multiplier using Optimized Wallace Tree (shown as red wires and blocks).

8 Thoughts and Advices

1. 這次的 lab 自己設計乘法器滿有趣的，以往計組最多只學到 Robertson algorithm 和 Booth algorithm，但這次的 lab 還介紹了其他的作法和優化技巧，包括 Modified Booth's algorithm 和 Wallace Tree，收穫很多。

2. 這次 lab 但在環境上花了不少時間，助教推薦用的 MobaXTerm 只支援 Windows 系統，但 macOS 和 Linux 都無法安裝，如果可以的話希望未來助教在準備與測試開發環境時能多考量到使用不同作業系統的同學。

3. Makefile 有 redundant 的部分，可以寫得更精簡，不只乘法器 hardware 要 reuse，Makefile 指令也應該要 reuse。