AI on Chip 2024

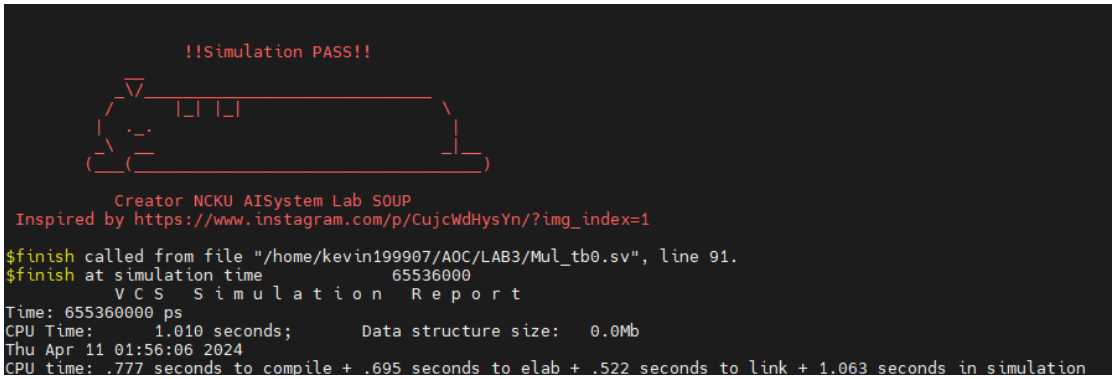
LAB III Hybrid Multiplier REPORT

Student name: __胡家豪_____
Student ID: __N26122246_____

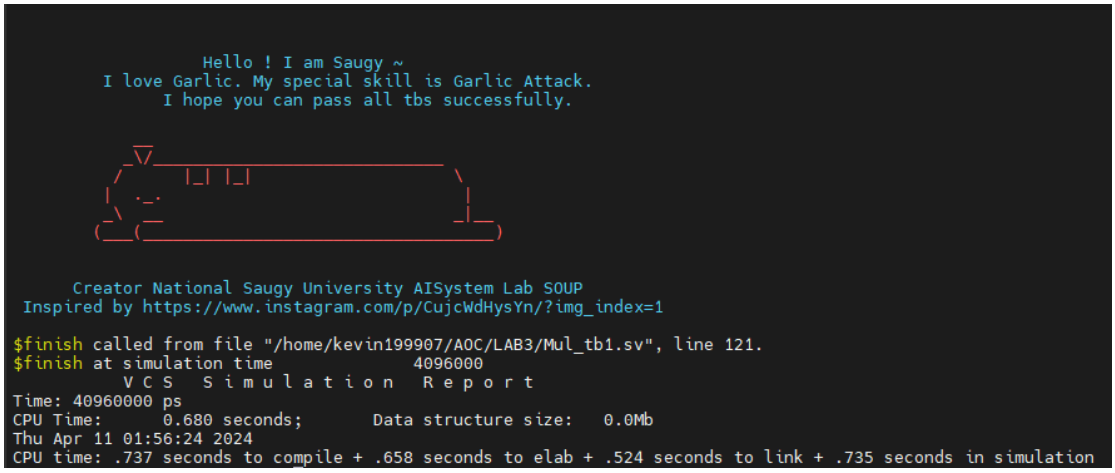
目錄

一.Implement Question List

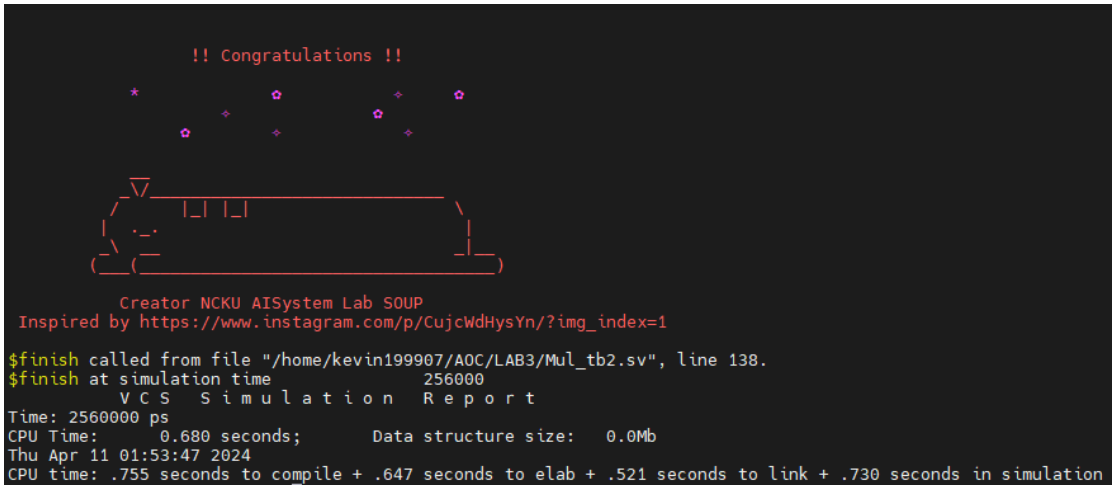
1. TB.0



2. TB.1



3. TB.2



二.Implement Problem

1. Explain how your multiplier works, its architecture and the algorithm you apply.

   我使用的是一般的 Booth's Algorithm 進行實作，並且使用 ifmap 當作乘數、filter 當作被乘數。

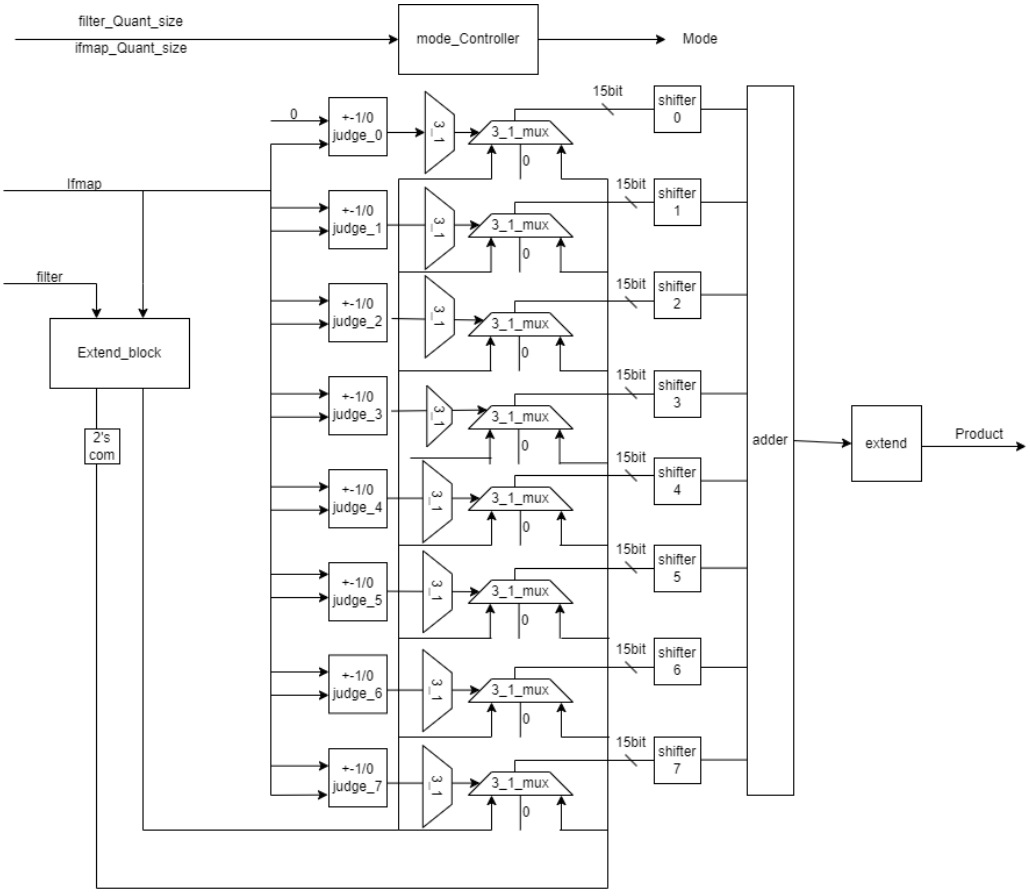
   將乘數最左邊先填充一個 0，接著將每兩個乘數進行分組，以 8*8 為例會分為八組，這八組都拉一條線出來並且擴充到 7+7-1 個 bits(15)，並利用下列判斷決定這組線要給甚麼值：00、11 給全 0；01 給被乘數；10 給被乘數的 2 補數。

   接著由右至左依序將每一個組別的線往左 1bit：第一組往左 0 bit、第二組往左 1 bit….。結果利用一個加法器加起來，最後用 sign 拓展到需要的 bit 數。

   值得一提的是，如果結果是負最大乘上負最大需要額外進行處理

2. Introduce each module and write down how do you reuse hardware resources to accomplish 3tbs

   下列是我的架構，由於把所有的線畫出來會很雜亂，所以沒有把 judge block 進入 mux 的線畫出。我透過選擇 Quant_size 決定 mode，用 mode 進行控制直的 mux 的輸出與 shift bit 的位數以及輸入輸出的被乘數與乘數的數量：



tb0、tb1 與 tb2 都使用同一個架構，雖然 mux 輸出都是 15bits，但是根據不同的 tb，可以調整只賦值 15bit 的其中某幾個 bit。

Tb0：mux 輸出取 15 bits，並且上面的八個 judge block 是判斷同一個乘數，由上至下分別判斷 {ifmap[0],1'b0}….. { ifmap[6], ifmap[7]}，並且 shifter0 位移 0 bit、shifter1 位移 1bit、… shifter7 位移 7bits。

Tb1：mux 輸出取 7bits，並且只需要上面的四個 judge block 判斷一個乘數(ifmap)。上面四個從{ifmap[0],1'b0}…{ ifmap[3], ifmap[2]}；下面四個 judge block 則是 don't care。
被乘數(filter)則用兩個 7bits 的線拉進去橫的 mux，並且第一個 filter 使用上半四個；第二個 filter 使用下半四個。
橫的 0、4 mux 用第 1 個 judge block 進行判斷、橫的 1、5 mux 用第 2 個 judge block 進行判斷、橫的 2、6 mux 用第 3 個 judge block 進行判斷、橫的 3、7 mux 用第 4 個 judge block 進行判斷。

Tb2：mux 輸出取 3bit。並且利用第 0、第 1 個 judge block 是判斷第一個乘數(ifmap)；利用的第 4 個、第 5 個 judge block 是判斷第二個乘數。上面兩個判斷{ifmap[0],1'b0}、{ ifmap[1], ifmap[0]}；下面兩個判斷 {ifmap[4],1'b0}、{ ifmap[5], ifmap[4]}。
被乘數(filter)則用兩個 3bits 的線拉進去橫的 mux，由上數下來：第一個 filter 使用 mux0、mux1；第二個 filter 用 mux2、mux3；第三個 filter 用 mux4、mux5；第四個 filter 用 mux6、mux7。
Mux0、mux2 利用第 0 個 judge block 判斷；Mux1、mux3 利用第 1 個 judge block 判斷；Mux4、mux6 利用第 4 個 judge block 判斷；Mux5、mux7 利用第 5 個 judge block 判斷

三. No-needed implement questions

1. How many (minimum) FLOPs and MACs does VGG16-Cifar10 1st layer require? Write down your calculation process. (without batch normalization and bias)

在 VGG-16 第一層的 layer 輸入為 224×224×3，kernal size 為 3×3，output feature map 為 224×224×64。
其參數量為：input channel × output channel × kernal size，也就是 3×64×3×3。
總共 MACs 數為：3×64×3×3×224×224 = 86,704,128
而 FLOPs 約為 MACs 的兩倍，為 173,408,256

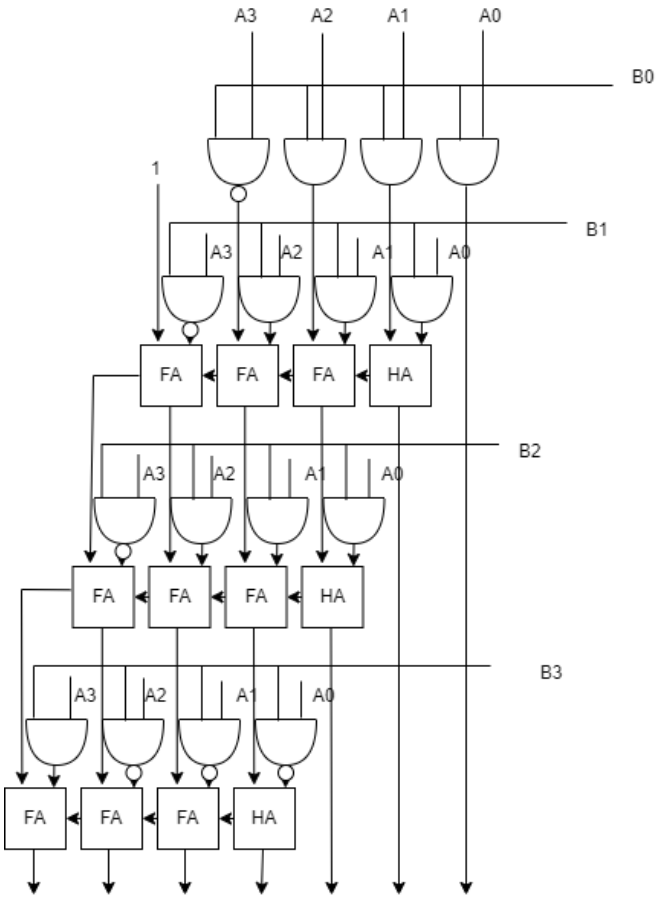
2. How many (minimum) FLOPs and MACs does a fully connected (512-10) layer require? Write down your calculation process. (without batch normalization and bias)

   在 fully-connceted layer 中，MACs 是 input node × output node
   也就是 5120； FLOPs 約為兩倍，也就是 10240

3. Compare the difference among using Robertson Algorithm, Booth Algorithm and Modified Booth Algorithm with Hybrid Multiplier with a table.

Algorithm	**描述**	**優勢**	**劣勢**
Robertson Algorithm	判斷每個 bit 並且在最高位判斷正負	行為簡單	因為要判斷每一個 bit，所以很慢
Booth Algorithm	每兩位判斷一次正負並且進行位移	較 Robertson Algorithm 快速	實作難度中等、遇到特殊的 case 還是很慢
Modified Booth Algorithm	每三位判斷一次正負與要相加的值並且進行位移	比原本的 Booth 還要快，需要判斷的次數也比較少	實作較 Booth 複雜

4. (Architecture 2) Design 4-bit & 4-bit multiplier (signed-number multiplication) with full and half adders follow architecture and point figure format in page 14. Introduce your architecture and method. Count and mark the latency/critical path of 4-bit & 4-bit multiplier. (you can use Adder as time unit).



用經典的串接三層 adder 架構完成 4*4 的乘法器，total 的 latency 為三層 adder

四.My opinion

這次的作業花了我不少心力，最花時間的應該是思考硬體到底怎麼實現 booth 演算法。然後使用 combinational 電路對不同的 input 進行 reuse。不過我覺得只要搞懂了第一個與第二個 tb，後面第三個 tb 的概念就呼之欲出

過去在計組學習這種快速乘法都只有在紙上練習，實際實作的時候才發現沒有那麼簡單。不過雖然過程很辛苦，但是透過這次課程也讓我初步了解到如何透過硬體實現這種支援不同輸入的情況(例如過去學到有的 AI 加速器可以支援不同的 kernal 大小)。

最後看著三個腸太郎都變成粉紅色，成就感也蠻大的。