AI on Chip 2024

LAB III Hybrid Multiplier

REPORT

Student name: \_\_胡家豪\_\_\_\_\_\_\_\_\_

Student ID: \_\_N26122246\_\_\_\_\_\_\_\_

目錄

[一. Implement Question List 3](#_Toc163638065)

[1. TB.0 3](#_Toc163638066)

[2. TB.1 3](#_Toc163638067)

[3. TB.2 3](#_Toc163638068)

[二. Implement Problem 3](#_Toc163638069)

[1. Explain how your multiplier works, its architecture and the algorithm you apply. 4](#_Toc163638070)

[2. Introduce each module and write down how do you reuse hardware resources to accomplish 3tbs 4](#_Toc163638071)

[三. No-needed implement questions 5](#_Toc163638072)

[1. How many (minimum) FLOPs and MACs does VGG16-Cifar10 1st layer require? Write down your calculation process. (without batch normalization and bias) 5](#_Toc163638073)

[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) 6](#_Toc163638074)

[3. Compare the difference among using Robertson Algorithm, Booth Algorithm and Modified Booth Algorithm with Hybrid Multiplier with a table. 6](#_Toc163638075)

[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). 6](#_Toc163638076)

[四. My opinion 7](#_Toc163638077)

1. Implement Question List
2. TB.0

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. TB.1

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. TB.2

一張含有 文字, 螢幕擷取畫面 的圖片

自動產生的描述

1. Implement Problem
2. 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

1. 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：

一張含有 文字, 圖表, 方案, 工程製圖 的圖片

自動產生的描述

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

1. No-needed implement questions
2. 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

1. 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

1. 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 |

1. (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).

一張含有 圖表, 工程製圖, 方案, 圖解 的圖片

自動產生的描述

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

1. 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。