HW5_Domain specific accelerator

黃柏竣, 110550123

***Abstract*—此作業目標為，對用 C code 寫的手寫字辨識程式進行加速，透過此學到 domain specific accelerator 的概念、如何從 C 來跟 Aquila 的該元件做溝通，以及 Xilinx IP 的使用。了解後，嘗試做分析及加速。**

I. INTRODUCTION

Domain specific accelerator (DSA) 是專門設計來用於執行特定任務的硬體加速器。相對於 CPU 可以執行各個任務，特地設計的 DSA，會在執行特定任務時提供更好的效能效率。

而此作業有神經網路模型相關應用(做的是浮點數的乘加)，並使用 Xilinx IP，也用了 SD card 讀取資料，除了 Verilog 以外，也要對 C 的部分做一些修改，才能讓軟體執行時可以對硬體做操作，然後獲得加速的效果。

II. IMPLEMENTATION

A. *Memory-mapped I/O*

Memory-mapped I/O (MMIO) 是透過讀寫特定內存位址，來讓 CPU 和其他設備/裝置溝通，而在此次作業中也是使用這樣的方法，讓 CPU 執行 C code 時，可以觸發 DSA 的操作。實作就是在 C code 就是指定好位置後，把 neuron 或 weight 的值 assign 到該位置，Aquila 執行程式時，透過 soc_top.v 的 dev_addr 判斷，就會知道要去使用 DSA 這個裝置。

B. *DSA detail*

Verilog 部分，先在 soc_top.v 把 HW5 會用到的東西取消註解，用 C4 開頭的那串位址(dev_addr)，來得知現在是否有要使用此裝置。再來是重要的裝置，DSA 本身，多寫一個 DSA module，而此 module 是參考 clint module 來實作的，裡面有 register dsa_mem，來存取特定位址的資料，像是 input A 是 weight 的部分，就分給他 0xc400_0000 這個位址，因為 input 有 dev_addr，可以來判斷是給哪個 register，其他 registers 像是 input B、lock、result，都和 input A 類似，各有一個特定位址。

因為這個 module 每次觸發到時都須回傳值回去，但是 floating point IP 又沒辦法馬上就算好結果，所以這邊才用了 lock 這個方式，C code 那邊就是用 while 迴圈一直讀 lock 的值，讓它 busy wait，直到結果被算出來，valid 訊號會將 lock 改為 1，讓程式繼續下去。

此次有做到的計算修改僅有 neuronet.c 迴圈裡的小部分，也就是沒有一次將 neuron、weight 的 vector 取出來，而是一次取一個值，先做計算，算完了才再去取下一個值，且一開始只有先讓 Xilinx IP 做乘法的部分，然後讓 CPU 做加法，之後找到了自己寫 code 的 bug，才讓加法實際派上用場。(E3 上用加法之前的檔案是沒有 new 的壓縮檔)

C. *Xilinx IP for floating point*

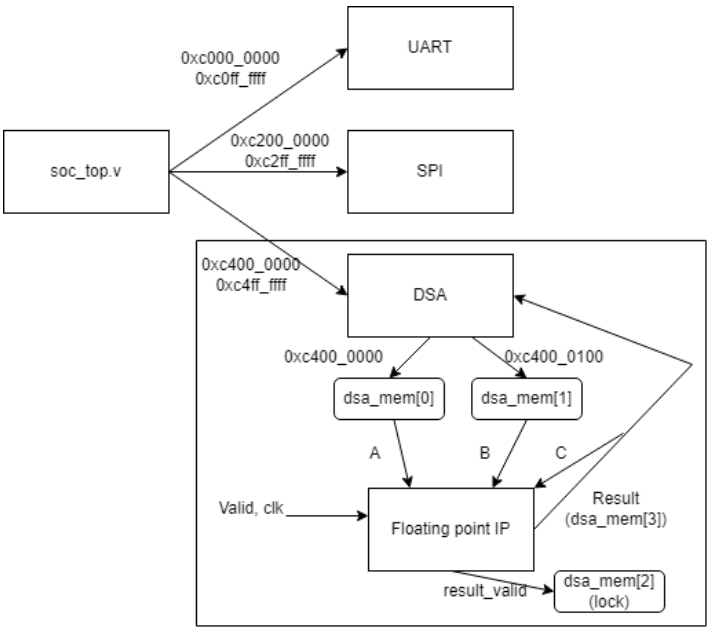
Floating point IP 的部分，就是照著簡報的資訊來做設定，因為是要做神經網路的各神經元權重間的乘法加法，所以是選 multiply add，以及 non-blocking mode。

接下來就是把 input A B 提供給它，valid 的部分就是對應 register 有進來值時就是 1，在此輪算完後再變回 0，input C 的部分如同上面所說先只有做乘法部分就是讓 C 保持是 0，在 C code 中再把乘起來的值用 CPU 加進 inner product 裡。

後來改成把值存在 register 內，每次放入 input C，讓加法也有做事，每跑完 784 個 neurons，先加進 inner product，再在 C code 部分把值設回 0，答案才會是正確的，不會一直無限累加上去。

實作的部分整體設計大約如下圖 Fig. 1. 所示。

Fig. 1. Architecture diagram for a part of homework5 Verilog.



III. RESULTS

A. Original and DSA runtime

在寫出 DSA 架構及修改 C code 後，進行測試，執行 ocr.elf，執行時間的單位為 millisecond，從 gtkterm 中得到的結果資訊如下表 TABLE I.。

TABLE I. RUNTIME OF OCR.ELF BETWEEN ORIGINAL AND DSA

Type	***Runtime***	***Correct rate***
Original	21848	85%
DSA without adding	13144	85%
DSA with adding	6626	85%

IV. DISCUSSION

A. No high performance improvement

此次的作業沒達到很好的效果，可能的原因如下: 沒有處理整個向量，而是每次僅處理一個值，還有運算的部分也沒有跟把資料餵進位址裡做同時進行，看起來都是一輪一輪慢慢做。應該是因為這些種種因素疊加起來，使得表現沒有辦法加速到老師上課說的那麼快。

B. Computing data feed and computation time

原本有打算仿照 ocr.c 的方式用 tick()來算時間，不過我是一組一組 neuron weight 算的，不是一次把資料都餵進去，所以毫秒的精度不太夠，結果都顯示 0，又想說如果把他們加起來，或是加這些在 C code，可能會造成一些意料之外的 overhead，所以最後就沒有做。

C. Using adding or not

原本因為有點便宜行事，就先沒去管 adding 的部分，結果意外的看到了有沒有使用加法之間的差距，在把加法視為虛設的情況下，加速程度連兩倍都不到，不過把 C code 的加法，也就是 CPU 做的部分加法給 DSA 做後，就有超過三倍的加速。

D. Reflection

此次作業花了蠻多時間在 debug，結果錯誤就重跑 bitstream，也是不少時間，加上期末各項事情，可惜結果沒到很好看，不過仍學到了很多，像是之前雖然上課會聽到 Xilinx IP，但都不太確定要怎麼用，此次作業就有需要實際操作，去看規格書之類的情況；還有雖然第一次作業就有提到 clint 了，但我一直沒有很搞懂，是到了此次作業才了解在做甚麼，以及怎麼讓它做自己想做的事；最後，有實際見識到硬體和軟體間計算的速度能耐，僅是最裡面的迴圈的乘加改成直接讓 DSA 做，就能讓秒數變化如此大，這些都令我受益蠻多，對軟硬體整合的概念也更加清晰。