SystemC 與數位系統設計概論

Final Project 書面報告

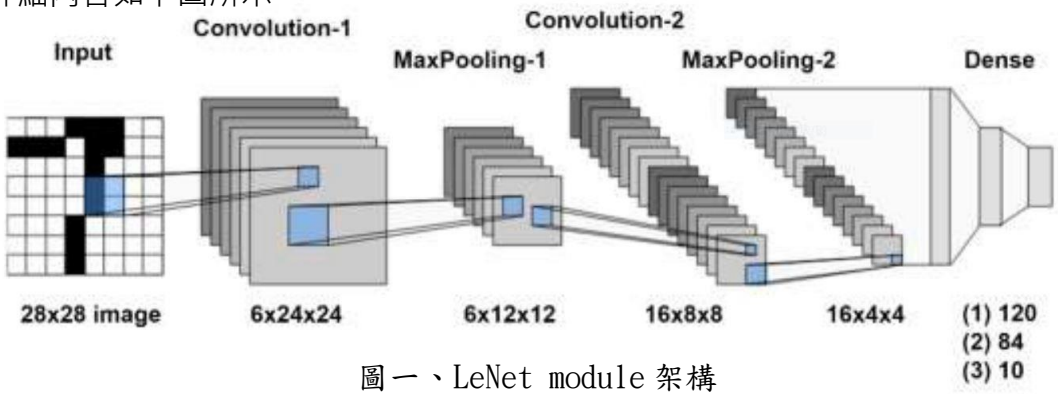
一、組別：第九組

二、組員(學號)：鄭璟翰(B093040003)、郭晏涵(B093040024)

三、繳交期限：2022/01/05

四、design ideas：

期末報告的要求是實作一個 CNN model – Lenet-5，能夠判斷一個輸入的圖片(手寫數字)的內容為何，而且需要作出兩種版本的輸入：fixed point、float。其中 CNN model 中所進行運算的 module 為 LeNet，也就是我們需要實作的部分，詳細內容如下圖所示。



圖一、LeNet module 架構

根據一開始所輸入的 input image 進行操作，最後可以得到十個數字，分別代表 0~9 的可能性。因為輸入圖片是一個手寫的 7，所以最後結果 7 應該最大。

除了上圖所見到的三個種類八層 module 需要自己實作之外，還有以下幾種 module 已經寫好了可以直接使用：Reset(初始化各 module)、Clock(供 module 判斷可否動作)、ROM(提供資料的記憶體，存有 input 和各個 module's Kernel)、RAM(暫存資料的記憶體，可以存入資料及讀取先前存入的資料)、Monitor(輸出結果)。

關於存取資料的方式，我們選擇將資料全部讀入每個 module 後再進行運算，並將結果輸出至 RAM 之中，下一個 module 則去 RAM 讀取上次運算的結果，除了最後一個 Dense 3 直接將結果輸出至 Monitor。

接下來將依序介紹三種 module 之間的內容。

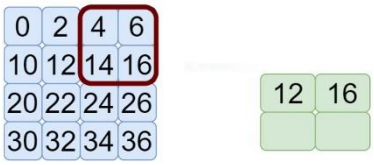
(一)、Convolution：

執行卷積的運算，根據讀入的 Channel 數量及 Kernel 數量進行乘累加的動作。假設有 A 組 input channel、B 組 output Channel(以 Conv-1 為例，A = 1、B = 6)，那總共會有 B 組 Kernel，每組 Kernel 會有 A 個 Kernel(一個 size * size

大小的 weight)和一個 bias，將每一個 input channel 乘上其中一組 Kernel 後累加並加上 bias，就可以得到一個其中一組 Output Channel。再將 Input Channel 乘上其他組 Kernel 就可以得到其他 Output Channel。

(二)、MaxPooling：

選取同一範圍內，最大的資料。根據 Input data 和 Size，每次選取一個 Size * Size 大小的矩陣，將最大的數字當作結果。

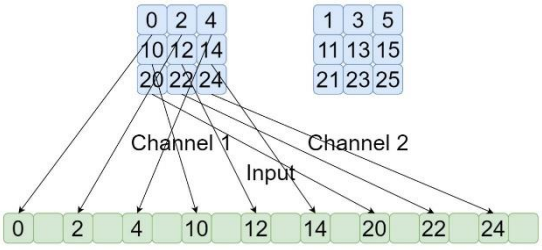


圖二、MaxPooling 範例

以上圖為例，Size = 2，Stride = 2(每次移動兩格)，選取 2 * 2 方陣中最大的數字，放入結果中。

(三)、Dense：

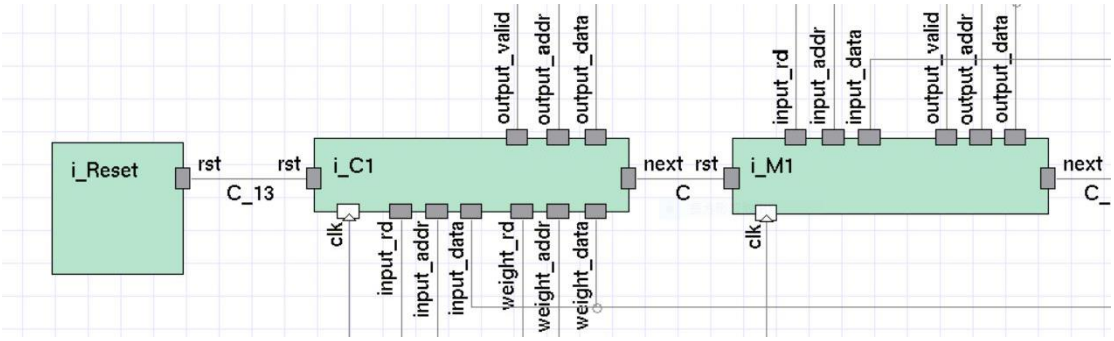
在進行第一次 Dense 步驟之前，需要先將二維的資料攤平成一維，攤平的順序是先將每個 Channel 的第一個位置好入陣列中，再回到第一個 Channel 擺第二個，直到全部排完。



圖三、Flatten

Dense 運算則是根據將 input list 乘上每一組 weight list 再加上 bias，可以得到一個數字，最後透過激活函數 ReLU(x if x > 0 else 0)將結果修正至正數。根據每一層有幾個 weight list，將會有幾個輸出。

完成每一個 module 後，接下來將實現每一個 module 之間的連線、資料如何儲存及傳輸。首先是 Reset 的部分。



圖四、Reset 連線(部分)

我們選擇只有將 Reset 接到第一個 module(conv-1)，其他 module 的 rest 則由上一個 module 運算完後進行呼叫。原因是如果將所有 module 都接到 Reset 後，那每個 module 將無法判斷什麼時候要開始運算，因此我們將每一個 module 在每一個 module 後都加上一個 reset 的程式碼，用於初始化下一個 module。

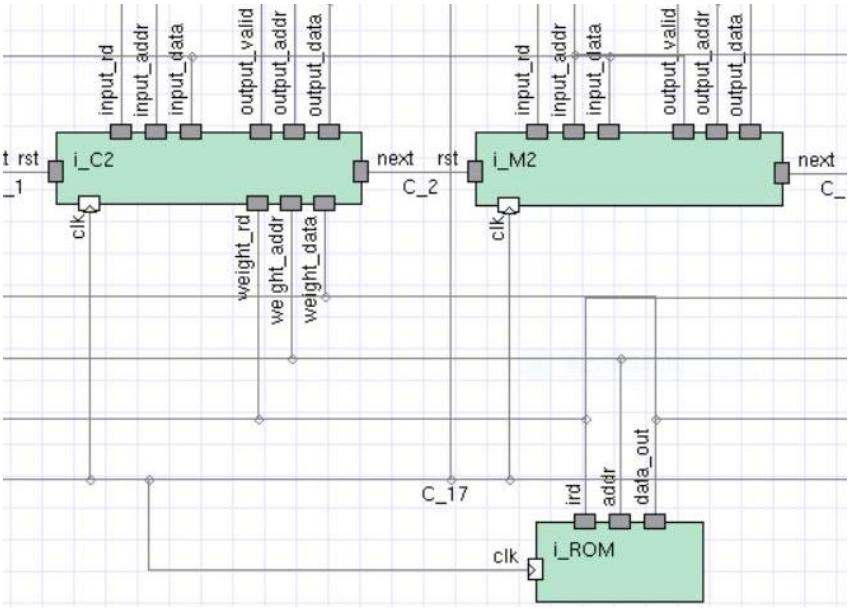
```
102     if(isNext){
103       for(int i = 0; i < inputChannel; i++){
104         for(int j = 0 ; j < inputSize; j++){
105           delete [] input[i][j];
106         }
107         delete [] input[i];
108       }
109       delete [] input;
110
111       for(int i = 0; i < filter; i++){
112         for(int j = 0; j < inputChannel; j++){
113           for(int k = 0 ; k < kernalSize; k++){
114             delete [] kernal[i][j][k];
115           }
116           delete [] kernal[i][j];
117         }
118         delete [] kernal[i];
119       }
120       delete [] kernal;
121
122       delete [] bias;
123
124       callRestNext.notify();
125       isNext=false;
126       start=false;
127     }
128   }
129 }
130
131 void Conv::resetNext(){
132   next.write(true);
133   wait( 3, SC_NS );
134   next.write(false);
135 }
```

在 Conv::Run function 中

```
124     SC_THREAD( resetNext );
125     sensitive << callRestNext;
126     dont_initialize();
```

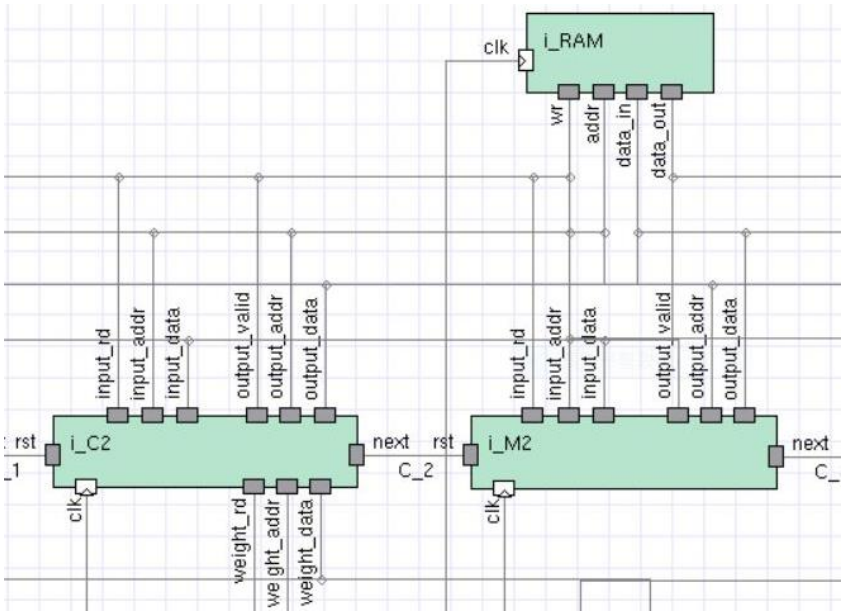
圖五、Reset 程式碼(部分)

接下來介紹如何資料如何儲存與讀取。



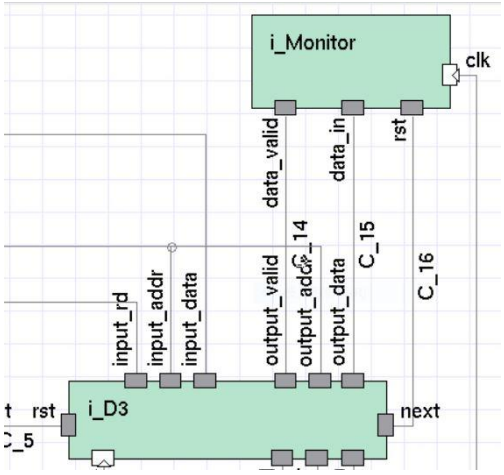
圖五、ROM 讀取資料連線(部分)

首先介紹從ROM讀取資料的部分。需要從ROM(LeNetAll.txt)的資料是input data以及所有的weight，因此要從ROM讀取資料的部分有Conv-1的input data、weight 和其他 module 的 weight。每一種資料要讀取需要三條線和兩邊各三個 port 進行連接：分別是 rd(判斷可不可以傳輸)、addr(address，要讀取資料的位置)、data(所要求的資料)，如上圖五所示，Conv-2 中的 weight 連了三條線道 ROM。



圖五、RAM 讀取資料、輸出資料連線(部分)

接著說明 RAM 讀資料與寫資料的方式。因為 RAM 比 ROM 多了一個寫資料的功能，所以 RAM 多了一個 data in 的 port，作為其他 module 寫資料的管道。讀資料的部分，除了 conv-1 的 input data 要從 ROM 讀之外，其他 module 的 input 都要從 RAM 讀(前一個 module 做完輸出到 RAM)，範例如上圖中 conv-2 和 maxpool-2 的 input rd、input addr、input data 分別接到 RAM 的 wr、addr、data out(輸出)；寫資料的部分，除了最後一個 Dense-3 輸出到 Monitor 之外，其他 module 的結果都要輸出至 RAM 之中，如上圖中 conv-2 和 maxpool-2 的 output vaild、output addr、output data 分別接到 RAM 的 wr、addr、data in(寫入)。



圖六、Monitor 連線

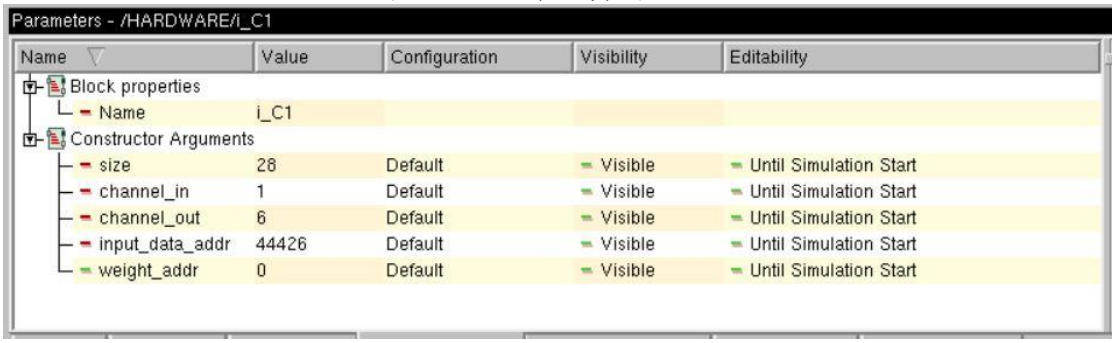
最後則是輸出 Monitor 的接線，最後一個 Dense-3 將輸出的信號 output valid 以及要輸出的數字 output data 接到 Monitor 的 data valid、data in，即可將結果輸出到 PA 的 console 上。

完成 module 之間的連線後，還需要設定每一個 module 之中 size 大小，如 input channel size、kernel size、output channel size 等、channel 的數量、以及讀寫資料的起始位置等數值。每一個 module 的 size 和 channel 數量可以從圖一(題目所示)中得到，從 ROM 讀取的資料同樣可以從題目中所附的表格得知(如下圖六)。從 RAM 讀資料的話則都是從 0 開始，因為每次計算時，都先將全部的資料讀入，再將開始計算，最後才輸出，因此可以將資料從頭開始放即可，不會有覆蓋資料的問題，因為裡面的資料都已經使用過了。

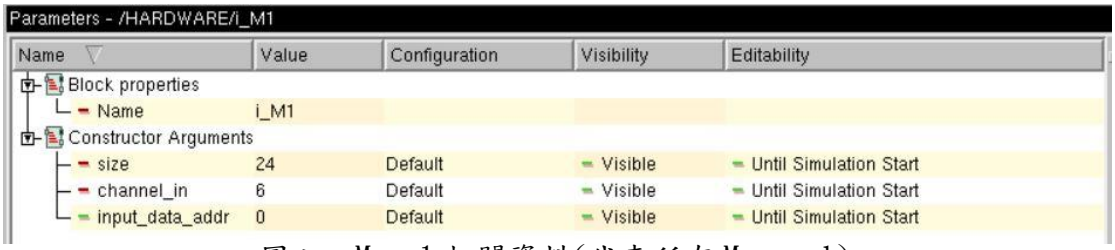
ROM

Index	Data
0-155	Conv_1_weight
156-2571	Conv_2_weight
2572-33411	Dense_1_weight
33412-43575	Dense_2_weight
43576-44425	Dense_3_weight
44426-45209	Input_data

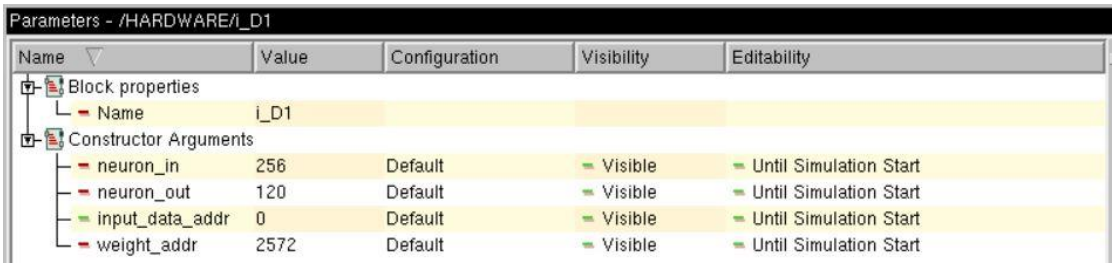
圖七、ROM 存放資料的位置



圖七、Conv-1 相關資料(代表所有 Convolution)

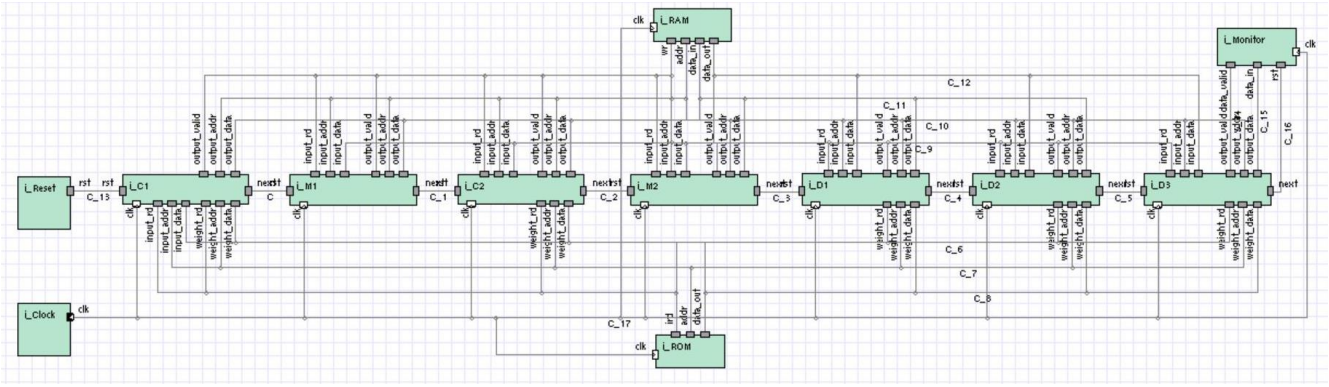


圖八、Max-1 相關資料(代表所有 Maxpool)



圖九、Dense-1 相關資料(代表所有 Dense)

五、block diagram, and simulation result of PA：



圖十、block diagram

最上方的module是RAM，最下方的RAM，右上角則是Monitor，中間module的順序從左到右分別是 conv-1、maxpool-1、conv-2、maxpool-2、dense-1、dense-2、dense-3。

```
                Copyright 1996-2017 Synopsys, Inc.
        This Synopsys product and all associated documentation are
  proprietary to Synopsys, Inc. and may only be used pursuant to the terms
     and conditions of a written license agreement with Synopsys, Inc.
     All other use, reproduction, modification, or distribution of the
  Synopsys product or the associated documentation is strictly prohibited.

  SystemC 2.3.1 --- May 12 2017 20:20:38
  Copyright 1996-2017 by all Contributors,
  ALL RIGHTS RESERVED

Loading weights and input data...
done!
0: 0!
1: 0!
2: 0!
3: 0!
4: 0!
5: 0!
6: 0!
7: 31.999!
8: 0!
9: 0!
SystemC: simulation stopped by user.
```

圖十一、fixed-type result

```
                Copyright 1996-2017 Synopsys, Inc.
        This Synopsys product and all associated documentation are
  proprietary to Synopsys, Inc. and may only be used pursuant to the terms
     and conditions of a written license agreement with Synopsys, Inc.
     All other use, reproduction, modification, or distribution of the
  Synopsys product or the associated documentation is strictly prohibited.

  SystemC 2.3.1 --- May 12 2017 20:20:38
  Copyright 1996-2017 by all Contributors,
  ALL RIGHTS RESERVED

Loading weights and input data...
done!
0: 0!
1: 1.51689!
2: 0!
3: 1.89325!
4: 0!
5: 0!
6: 0!
7: 15.3189!
8: 0!
9: 0!
SystemC: simulation stopped by user.
```

圖十二、float-type result

六、difference of floating-point-based version and fixed-point-based version

```
83      #ifdef fixed_DATA_TYPE
84      if(sum.range(31, 16)){
85        sum[15]=0;
86        sum.range(14, 0)=32767;
87      }
88      #endif
```

```
296     #ifdef fixed_DATA_TYPE
297     if(sum.range(31, 16)){
298       sum[15]=0;
299       sum.range(14, 0)=32767;
300     }
301     #endif
```

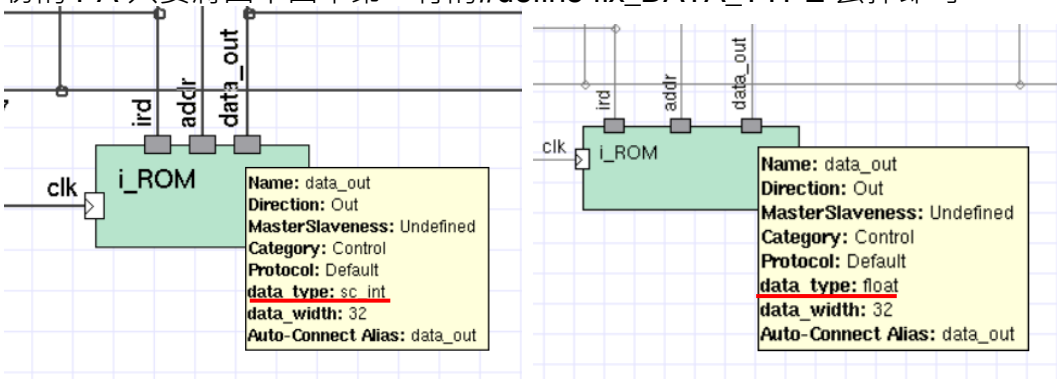
圖十三、fixed 與 float 計算差別

左圖是來自 LeNet.cpp 中的 conv::run()、右圖是來自同檔案中的 Dense::run()，這兩個地方是在程式碼中唯二針對 fixed-type 和 float-type 處理的地方，是在處理 fixed-type 乘法後進位的問題。可以看到這兩個地方都是如果有宣告 fixed_DATA_TYPE 的話就執行下面 84~87(296~300)這段程式碼，程式碼的內容是判斷 sum 的第 16~31 個位元是否有數值，如果有的話，則將 sum 的第十五個位元設定成 0，其他位元設定成 1(2^15 -1 = 32767)，原因是在下圖十四中，fix_data 的 MUL_DATA_TYPE 設定成 32 個 byte，但在 RAM 的讀寫中，只能允許 DATA_TYPE 或 READ_DATA_TYPE(16byte)的 size，因此如果計算完後的結果超過十六位數的話，則將結果設定為 16byte 能表示的最大數(第一個為 sign bit)。

```
1  #define fixed_DATA_TYPE
2
3  #ifdef fixed_DATA_TYPE
4    #define READ_DATA_TYPE sc_lv < 16 >
5    #define DATA_TYPE sc_int < 16 >
6    #define MUL_DATA_TYPE sc_int < 32 >
7  #else
8    #define DATA_TYPE float
9    #define MUL_DATA_TYPE float
10 #endif
```

圖十四、define.h

最後製作 PA 的部分需要製作兩份，因為發現如果在 PA import module 的當下如果 port 是 sc_int<16>的 type 話，即使更改程式碼仍無法改變狀態，第二份的 PA 只要將圖十四中第一行的#define fix_DATA_TYPE 去掉即可。



圖十五、data type 差異