**National Cheng Kung University**

**Department of Electrical Engineering**

***Introduction to VLSI CAD (Spring 2020)***

**Lab Session 7**

**Design of CNN Processing System**

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Student ID | | |
| 黃子青  劉育誠 | E24076938  E24076069 | | |
| Practical | | Points | Marks |
| Part 1 | | 10 |  |
| Part 2 | | 10 |  |
| Part 3 | | 15 |  |
| Part 4 | | 15 |  |
| Part 5  (Word整體報告內容詳細程度) | | 10 |  |
| Notes | |  |  |
|  | | | |

**Due: 23:50 May 3, 2020@ moodle**

**Deliverables**

1. All Verilog codes including testbenche should be uploaded.

NOTE: Please **DO NOT** include source code in the paper report!

1. All homework requirements should be uploaded in this file hierarchy.
2. Please **DO NOT** upload waveforms (.fsdb or .vcd)!
3. **If you upload a dead body which we can’t even compile, you will get NO credit!**
4. **All Verilog file should get at least 90% SuperLint Coverage.**
5. All homework requirements should be uploaded in this file hierarchy or you will not get full credit.

**File Hierarchy**

* Lab7\_E240XXXXX\_E240XXXXX.tar(Lab7\_E240XXXXX.tar) (Don’t add other text in file name)
  + Lab7\_E240XXXXX\_E240XXXXX (Main folder of this project)
    - Lab7\_E240XXXXX\_E240XXXXX.docx (Your homework report)
    - Makefile (You shouldn’t modify it)
* data (Images data in .txt format)
* test\_im\*.txt
* golden (Golden hexadecimal data)
* \*.txt
* images (Hand-written digit images)
* \*.png
* Include (RTL parameters definition file)
* def.v
* parameters (The weights and bias of the neural network)
  + layer\*\_w.txt
  + layer\*\_b.txt
* script (Any scripts of verification and synthesis)
  + - * Script files (DC.sdc, \*.tcl)
* sim
* top\_tb.v
* tsmc13\_neg.v
  + - * ROM (ROM behavior model)
        + ROM.v
        + ROM\_tb.v
      * TwoPortSRAM (Two-Port SRAM behavior model)
        + TwoPortSRAM.v
        + TwoPortSRAM\_tb.v
* src (RTL code)
* Adder.v
* Controller.v
* Decoder.v
* MUX2to1\_16b.v
* MUX2to1\_32b.v
* MUX4to1\_32b.v
* PE.v
* Pooling.v
* Relu.v
* Truncate.v
* top.v
* syn (Synthesized code and timing file)
* top\_syn.v
* top\_syn.sdf

*Part1*

* Objective:

Build a CNN Processing System as the architecture of Fig.1. The system can do inference of CNN as Fig.2 and classify hand-written digit images.

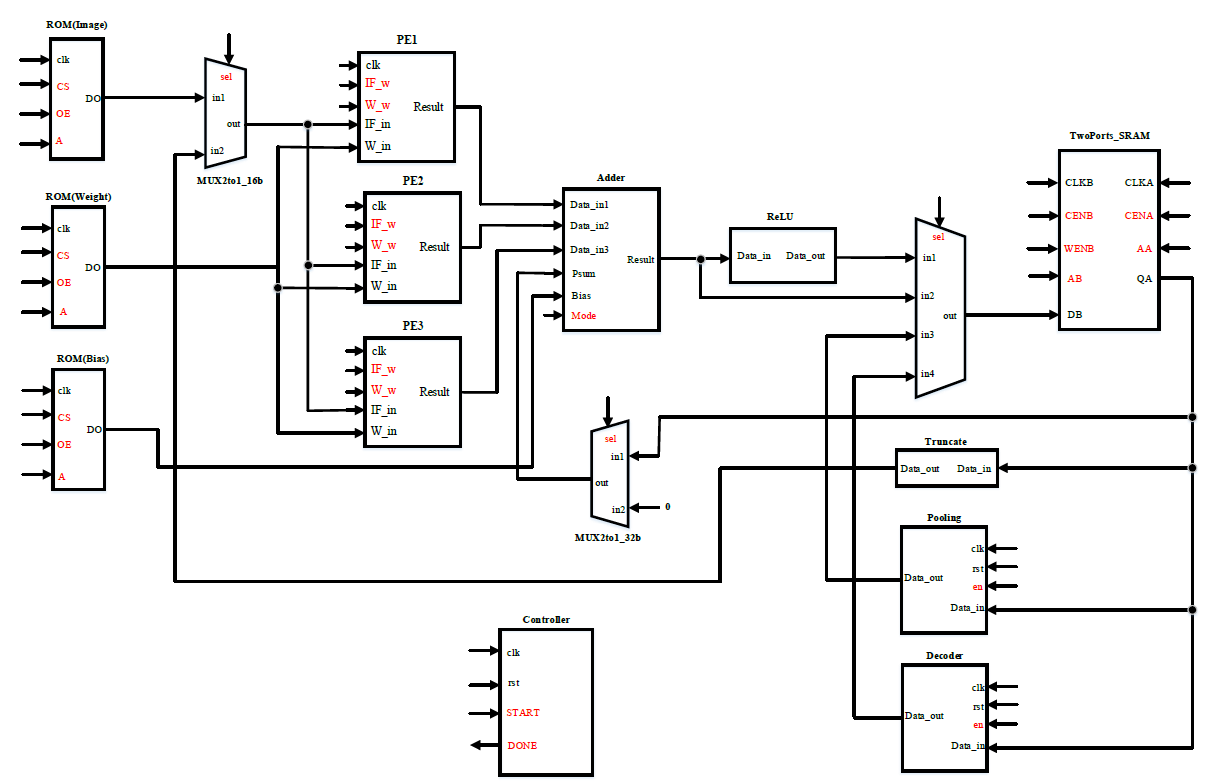


Fig.1 The architecture of CNN Processing System

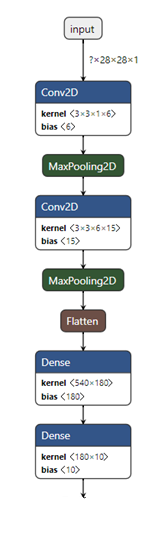


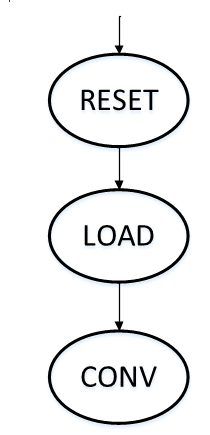
Fig.2 CNN NetworkArchitecture

* Please follow the tutorial of PowerPoint, and build a CNN Processing System by yourself. Run the system and let it pass the convolution layer 0 (Conv0) result for the image 0.

Simulation command: make rtl\_conv0

* Show your finite state machine (FSM) of the controller. Describe how your controller works.

Note: Please use software tools to clearly draw the FSM.

EX:

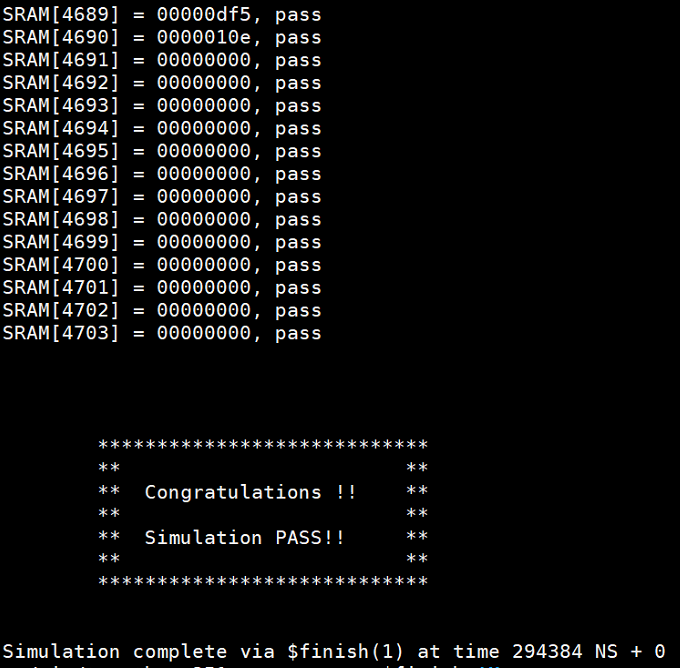
當Controller收到rst訊號時會進入RESET state，收到START訊號後會進入LOAD state。在LOAD state會去記憶體讀取convolution運算的資料(權重、partial sum、input feature map)，讀取完資料後進入CONV state做convolution運算。

|  |
| --- |
| Your FSM Design |
| 一張含有 文字, 地圖 的圖片  自動產生的描述  **Conv0:**    當Controller收到rst訊號時會進入INIT state，收到START訊號後會進入START state。下一個時刻就進入READ\_W，在READ\_W state會去記憶體讀取convolution運算的資料(權重、partial sum、input feature map)，讀取完資料後進入WRITE state做convolution運算，接著進入READ\_9跟READ\_C，一次讀九個feature map跟三個feature map，讀完後繼續做convolution運算，直到row到29，col到27，到conv0\_DONE的state，當conv0\_num<6的時候，代表還沒做完六堆，會從conv0\_DONE的state回到INIT state，然後一直重複做到做完六堆。  **Pooling1:**    當CONV0\_num大於五的時候，進入Pooling1\_INIT，下一個時間進入Pooling\_READ，這個時候它將CONV0做出來的結果一次吃四個值進來，吃進來之後進入CONV0\_WRITE然後將最大的值輸出，然後再回到Pooling\_READ，一直重複這個動作直到將CONV0的所有值都拿出來做比較輸出最大值之後，進入Pooling\_DONE。  **Conv2:**    當Pooling\_DONE之後，下一個進入的STATE是CONV2\_INIT，，接著進入CONV2\_READ\_W，在CONV2\_READ\_W state會去記憶體讀取convolution運算的資料(權重、partial sum、input feature map)，讀取完資料後進入CONV2\_WRITE state做convolution運算，接著進入CONV2\_READ\_9跟CONV2\_READ\_C，一次讀九個feature map跟三個feature map，讀完後繼續做convolution運算，直到row到11，col到13，接著到conv2\_DONE的state，當conv2\_num<6且conv2\_num<15，代表conv2的六層十五堆還沒做完，於是會回到CONV2\_INIT這個state，然後重複做直到六層十五堆做完。  **Pooling3:**    在CONV2的六層十五堆做完後，我們進入Pooling3\_INIT state，接著進入到Pooling3\_READ這個state，然後它就跟Pooling1一樣，先讀conv2的結果的四個值進來，然後進入Pooling3\_WRITE這個state，將四個數的最大值寫入，然後再回到Pooling3\_READ這個state，重複以上步驟，直到所有conv2的結果都讀完，然後就進入Pooling3\_DONE這個state。  **FC4:**    Pooling3\_DONE的下一個state是FC4\_INIT，在FC4\_READ\_9 state會去記憶體讀取convolution運算的資料(權重、partial sum、conv2的結果)，讀取完資料後進入FC4\_WRITE state做convolution運算，將運算完的結果寫入記憶體，一次讀九個數字做convolution，因為conv2的結果有540個，所以要讀六十次，FC4\_count要到六十，這540個結果都做完convolution且加在一起後，就是第一個輸出，然後我們要產生180個輸出，所以FC4\_num算到179後，FC4才算結束  **FC5:**    FC4做完後，下一個state是FC5\_INIT，在FC5\_READ\_9 state會去記憶體讀取convolution運算的資料(權重、partial sum、FC4的結果)，讀取完資料後進入FC5\_WRITE state做convolution運算，將運算完的結果寫入記憶體，一次讀九個數字做convolution，因為FC4的結果有180個，所以要讀二十次，FC5\_count要到二十，這180個結果都做完convolution且加在一起後，就是第一個輸出，然後我們要產生10個輸出，所以FC4\_num算到9後，FC4才算結束  **Decoder:**    讀FC5輸出的十個結果，然後將最大值的index輸出，那個index代表辨識出來的數字 |

* Paste the simulation result on the terminal.

Note: Don’t paste all the results. You only need to paste the result of the simulation end.

EX:



|  |
| --- |
| Simulation Result |
|  |

*Part2*

* Run the system and pass the pooling layer 1 (Pooling1) result of the image 0.

Simulation command: make rtl\_pool1

* Paste the simulation result on the terminal.

Note: Don’t paste all the results. You only need to paste the result of the simulation end.

|  |
| --- |
| Simulation Result |
|  |

*Part3*

* Run the RTL code. Let the CNN Processing System classify 200 images.

Simulation command: make rtl\_full

* Paste the simulation result on the terminal.

Note: Don’t paste all the results. You only need to paste the result of the simulation end.

|  |
| --- |
| Simulation Result |
|  |

*Part4*

* Run the post-synthesis simulation. Let the CNN Processing System classify 10 images.

Simulation command: make syn\_full

* Paste the simulation result on the terminal.

Note: Don’t paste all the results. You only need to paste the result of the simulation end.

|  |
| --- |
| Simulation Result |
|  |

*Part5*

* Show your Superlint coverage

Note: Use the following command to get the lines of your code.

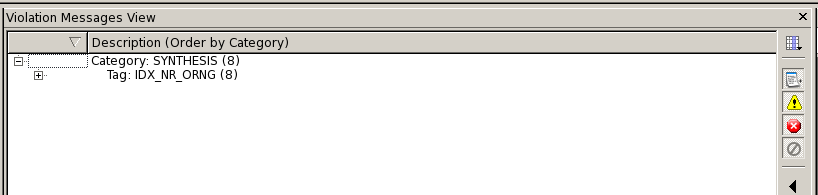
Command: wc –l ./src/\*

EX:

Number of lines of RTL codes:

Total lines = 1458

Number of warnings or errors on Superlint: 8



Coverage:

Coverage = 99.45%

|  |
| --- |
| Superlint Coverage |
| Total lines = 2903  Number of warnings or errors on Superlint: 2  Coverage = 99.93% |

* Lesson Learned

E24076938黃子青:

剛開始在聽助教宣布project的題目時，以為這會是一場非常艱難的挑戰，可是實際做過後才發覺 ”no magic, only basic”，基本上要用到的觀念在上一次Lab6就已經學過了，像是設多個counter來延長state的時間用以讀weight跟feature，READC一次讀一行，要有一個write state，要晚一個CLOCK來吃記憶體送過來的值等等，只是這次架構比較龐大而已，而且這次小module很多要自己打，所以做起來相當費時。

我們這次有遇到幾個大問題:

第一個問題=>**Pooling的做法與之前做的covolution不太一樣**

之前的convolution有三條PE，一次讓一條吃一次feature跟weight，然後吃三次再換另一條，一直到三條都吃滿值在讓他們做完convolution過後的值加起來，而這次的pooling只有一條，所以變成要一次給它4個clock讓它吃四次值，而且它要吃得值是來自SRAM而不是ROM，控制訊號要注意有沒有開對，取址的位址寫法也要不同，以上的差異讓我們從CONV0做完後在pooling花了不少時間理解怎麼做。

第二個問題=> **conv2有6個layer要做**

當做完pooling進入conv2時，我看不太懂簡報中的conv2到底是如何實踐的，後來仔細思考才得知，原來是6層的feature要跟6層的weight同時convoluion，可是問題是要怎麼用code實踐它 ? 我繼續專研簡報

，發現adder可以從SRAM吃一個psum的訊號，才想到原來conv2也可以用1層的convolution來實現它，只要每次做完一個element就把這個element放到其對應的位址，當下一層在做convolution時，就可以從這個位址取出psum，與做完convolution的值相加再存到這個位址，每個element都是這樣做，一直做到六層結束，再連續做15堆就可以做完conv2了，至於bias怎麼加，我們是在每一堆的最後一層加bias的。

我之後也就用同樣的觀念做完FC4，將540切成60組，每組做完的psum就存到SRAM，下一組做完的結果再加上psum再存到SRAM一直做到最後一組再加上bias然後存到SRAM，然後連續做180堆就完成了。FC5也是相同的道理。

第三個問題=> **跑make rtl\_full發現結果全部猜0**

當做完全部，跑make rtl\_full過後發現我們的結果全部都猜0，查了波形圖過後發現SRAM讀出的值全是XXX，然後我的decoder是預設output為0，所以才會有這樣的結果，翻了一下簡報才發覺”原來還要自己寫給decode的state，我原本還以為是testbench會自己比對SRAM的值”，後來便趕緊加了讓decoder取址取值的state，再跑一次make rtl\_full，結果就有數字跑出來了，只是又遇到下一個問題

第四個問題=> **跑make rtl\_full發現結果只對23個**

這時我們就攤開波形圖，從conv2地毯式收尋到底哪裡有錯，後來一直到

FC4才發現取psum的位址寫錯了，難怪幾乎全錯，這時再make rtl\_full，命中率瞬間飆到70%，這時我們的自信就來了，只要找出小bug就能結束這次冗長的project了，後來我們很快的發現我們在FC5的weight不小心用到fc4裡的變數了，才會導致出錯，然而make rtl\_full過後還是有21個錯，可是已經半夜2點半了，就決定先回去睡了。

第五個問題=> **只錯21個，而且都是錯0的圖**

回到寢室後一直在想，怎麼可能只錯0的圖，如果是前面就有寫錯的話沒道理只錯0的圖啊，後來才想到，啊!，會不會是decoder少讀到一個值，

因為我的decoder是從9讀到0的，所以有可能最後那個0沒有被讀到，

隔天一早就衝去soc，給decoder的state多加一個counter過後，按下

**make rtl\_full ，**偌大的Simulation Pass 印在螢幕上，也深深的重擊 我的臉頰，讓喜極而泣的淚水潸然烙下我的歡顏，成功了，we did it，趕緊de superlint的warning，然後確認slack≥0後跑合成，再按下make syn\_full，等了3分鐘過後，第一張圖跑完了但結果是x，此時我意識到這絕對不是我們code的問題，便趕緊呼叫隊友向fb的ivcad社團求助。

第六個問題=> **合成過後，結果都猜xxxx**

助教說可以去testbench調整clock的時間看看，我們便把10改成20， 再跑make syn\_full，發現第一張圖猜對了，在螢幕前靜待30分鐘後，偌大的Simulation Pass 又印在螢幕上，此時才終於得以喘息，結束了這場project大戰。

心得:

總之在這場project壕溝戰之中，有淚水有歡笑，且最後的結果是功德圓滿且收穫滿滿的，不只更精進自己的verilog coding能力，也稍微接觸到了一點深度學習類神經網路的概念，真的很感謝助教們設計了這樣的題目來培養我們的實力，不論將來去做研究或是去業界，我想我們已經有基礎的能力來應付了。

最後，數位的部分就在這告一段落了，很感謝這10個禮拜以來每個禮拜四晚上9點都被我跟同學敲門進實驗室問問題的助教，你們不厭其煩的教導讓我們verilog coding能力平步青雲，同時也培養起好的coding style。

劉育誠:

這次的實驗，我們做了一個很大很複雜的project，在這裡，我要感謝我的隊友，頻藉著他強大的實力與認真的態度，我們終於做了出來，這半個學期，我們對於verilog已經有初步的了解了，未來我們將繼續增強我們的實力。

* Contribution

E24076938黃子青 70 % , E24076069劉育誠 30%

*Appendix*

1. Simulation Requirements

You should make sure that your code can be simulated with specified commands in Table A-1. TA will use the same commands to check your design under SoC Lab environment. If your code can’t be recompiled by TA, you will get no credit.

TA will also see how many problems do you finish to decide which command TA will run. For example, if you only finish convolution layer 0, TA will run “make rtl\_conv0”. However, if you finish all layers, then TA will run “make rtl\_full”.

Table A-1: Simulation Commands

|  |  |
| --- | --- |
| **Command** | **Description** |
| make rtl\_conv0 | Run RTL simulation of convolution layer 0 for the image 0. |
| make rtl\_pool1 | Run RTL simulation of pooling layer 1 for the image 0. |
| make rtl\_full | Run RTL simulation of the CNN for 200 images. |
| make syn\_full | Run post-synthesis simulation of the CNN for 10 images. |