Digital System Design (Fall 2022)

Progress Report of Term Projects

Group Number (組別) : 11
Group Member 1 (組員 1) : Student ID：0812501　Name：許瀚宇
Group Member 2 (組員 2) : Student ID：109612041　Name：吳伯諺

Title (標題) : 矩陣處理加速器

A. Problem Description (問題敘述):

近年來深度學習網路的快速發展，不論是在電腦視覺或是自然語言處理等領域都大放光彩，其中深度學習模型的基礎是一連串矩陣運算的串接，因此如何快速地去計算一連串矩陣，變成為了決定模型推導效率的關鍵。

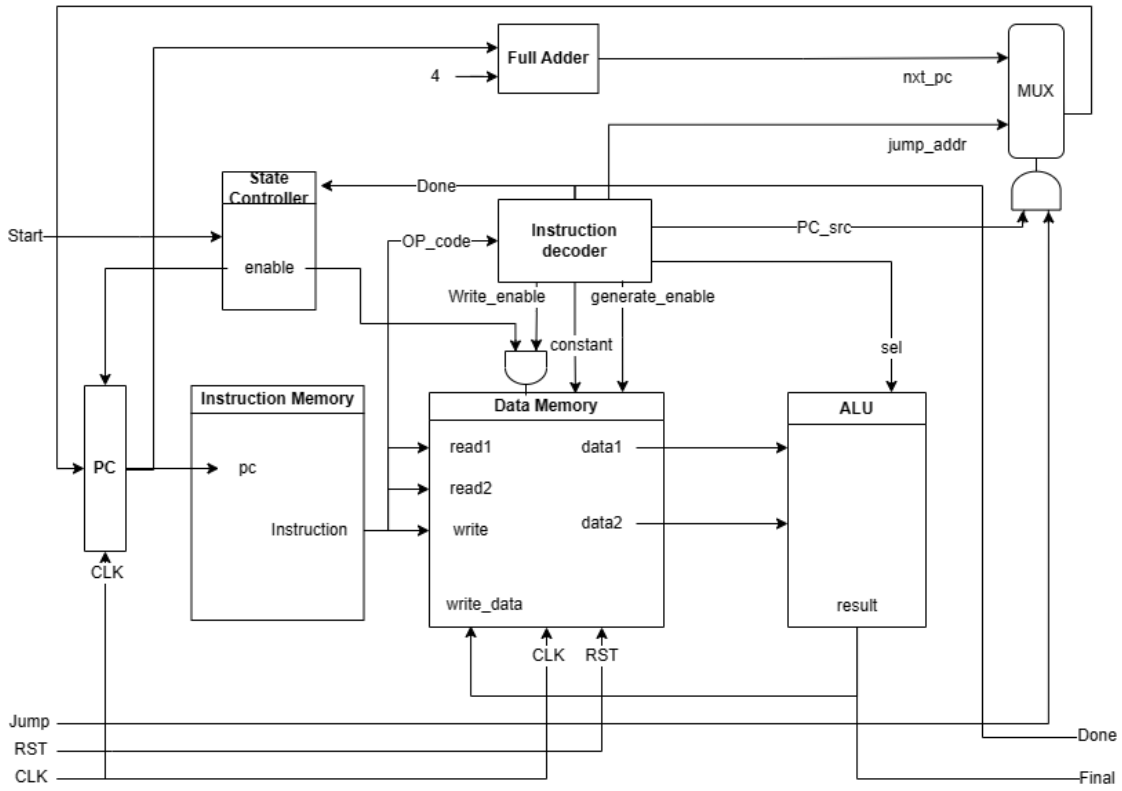
在本次專題中，想藉由對於矩陣做特別的處理器，其中包含了矩陣乘法、加法、減法、元素乘法等包含但不限於此的功能做出矩陣運算處理器，我們會將矩陣乘法視為本次專題的重點，希望能夠做出有效快速的處理矩陣運算的電路。

目前設定的目標為希望可以做出一個類似 CPU 的東西，讀取指令並根據指令去做對應的矩陣運算。

B. Flowchart or Procedure (流程圖或運作程序) :

1. Program counter（instruction fetch）
2. Load Memory & Instruction decode
3. ALU & Next program counter
4. Write memory

C. Block Diagram (方塊圖) :



D. Definition of Inputs, Outputs, Control Signals, and Status Signals (輸入、輸出、控制訊號、及狀態訊號之定義):

1. 輸入

訊號	說明
Start	用來告訴處理器開始處理
Jump	用來控制 Program counter 是否跳至 Jump_addr 的外部輸入，考慮到 branch 在這個處理器的設計中不適合放入，因此改用外部輸入控制是否 Jump。
Instruction memory	Instruction 控制下個 PC、矩陣在 Data memory 位置、矩陣運算方式。Instruction memory 則是儲存一系列的 Instruction，告訴處理器要按照什麼流程運算
CLK	就是 Clock
RST	重置 memory

※註：memory 是用 verilog file io 去寫入的

2. 輸出

訊號	說明
Done	當所有的 instruction.都完成的時候，就會把 Done 設成 True
Final	就是 ALU 計算後的結果，但只有當 Done 為 True 時會被取用

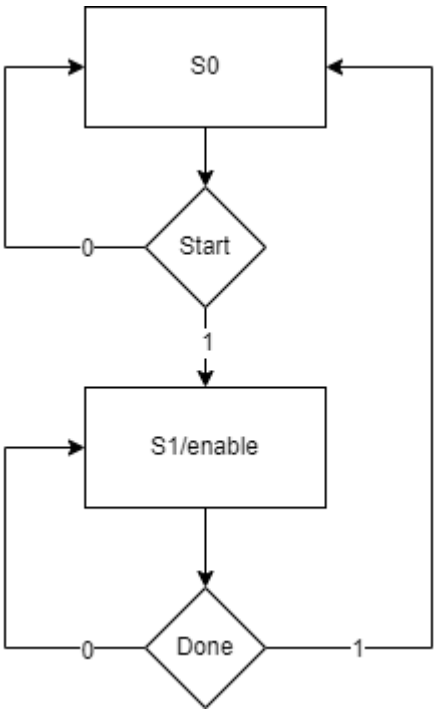
3. 控制訊號

訊號	說明
enable	只有 True 的時候才運作處理器（更新 Program counter、寫入記憶體）
OP_code	作為所有 instruction 的對應代號
write_enable	只有 True 的時候，ALU 的計算結果才會被寫回 Data memory
read1/2 write	作為用於操作/寫回矩陣的代號，以找到對應的記憶體 address
data1/2	從記憶體找到的矩陣內容
generate_enable	選擇 Read_data/Generated_data 作為 instruction 中第二個資料
sel	指定 ALU 用哪個操作為最後輸出
PC_src	使用哪一種 Program counter 做為下一個 CLK 的 Program counter

4. 狀態訊號

訊號	說明
0	Idle，處理器不會運作
1	Working，處理器運作中

E. State Machine Chart (SM Chart) or State Graph (狀態機器圖或狀態圖):



F. Description of Verilog Code (Verilog 電路模組說明):

instr_memory	儲存 instruction 的 memory
controller	控制是否運行下去的模組
decoder	解讀 instruction，生成對應控制訊號
data_memory	儲存矩陣資料的 memory
element_add/sub/mul/div/mod & matrix_mul	對矩陣進行各種操作的模組
ALU	多工器

G. Description of Test Bench (Verilog 測試模組說明):

透過 verilog file io 讀進以 python 生成的各個 instruction 對應的答案，將其與 processor 輸出的 result 比對是否一致，如果正確就顯示"instruction i is correct"否則顯示"instruction i is wrong"，i 是 instruction 的 index。

tb_generator.py 會生成 instructions.txt, data.txt, ans.txt 三個文字檔:

instructions.txt	將對隨機兩筆 data 做隨機一種操作的指令編碼為 2 進位的指令 最後 6 筆會對隨機一筆 data 與一個隨機常數矩陣分別使用 6 種操作
data.txt	隨機產生矩陣，padding 成符合電路模組輸入的大小，再將所有元素轉成 2 進位後串接成一行
ans.txt	每筆 instruction 對應生成的 data 後計算的結果，矩陣大小與輸入相同，將所有元素轉成 2 進位後串接成一行

H. Simulation Results (模擬結果):

以下是 6 種操作的模擬結果

element add:

```
op: 000 time:   280
data1:
   -85    -43    -96    -64
   -83    -40     61     58
   -22     48    -84    -42
   -52     34    -24    -75

data2:
   -14    -18    -72     61
    14    -53     28    -22
    -2     71     94     -3
    89     92    -44    -31

ans:
   -99    -61   -168     -3
   -69    -93     89     36
   -24    119     10    -45
    37    126    -68   -106

result:
   -99    -61   -168     -3
   -69    -93     89     36
   -24    119     10    -45
    37    126    -68   -106

instruction  13 is correct
```

element sub:

```
op: 001 time:   260
data1:
     2     -7    -70    -10
    87     66     33     27
   -84     28     33     25
   -43     46    -76     62

data2:
    61    -21     76    -28
   -75     37    -93    -51
   -82     38    -39    -19
   -82    -83    -89     76

ans:
   -59     14   -146     18
   162     29    126     78
    -2    -10     72     44
    39    129     13    -14

result:
   -59     14   -146     18
   162     29    126     78
    -2    -10     72     44
    39    129     13    -14

instruction  12 is correct
```

element mul:

```
op: 010 time:   240
data1:
   -39      8     19    -73
   -55     53     39     99
     5     -8     65    -70
   -37     19     79    -86

data2:
   -78    -52     78     -7
    88     78     81     44
   -64     76    -18    -88
     0      5     93     15

ans:
  3042   -416   1482    511
 -4840   4134   3159   4356
  -320   -608  -1170   6160
     0     95   7347  -1290

result:
  3042   -416   1482    511
 -4840   4134   3159   4356
  -320   -608  -1170   6160
     0     95   7347  -1290

instruction  11 is correct
```

element div:

```
op: 011 time:   300
data1:
   -14    -18    -72     61
    14    -53     28    -22
    -2     71     94     -3
    89     92    -44    -31

data2:
    53     55    -79     40
   -77     25      6     35
    -5    -63     73     88
   -57      3    -85   -100

ans:
     0      0      0      1
     0     -2      4      0
     0     -1      1      0
    -1     30      0      0

result:
     0      0      0      1
     0     -2      4      0
     0     -1      1      0
    -1     30      0      0

instruction  14 is correct
```

element mod:

```
op: 100 time:   180
data1:
   -85    -43    -96    -64
   -83    -40     61     58
   -22     48    -84    -42
   -52     34    -24    -75

data2:
   -50    -80     22     29
    90    -86     79    -22
    87     50    -55     93
   -93    -43     71    -28

ans:
   -35    -43     -8     -6
   -83    -40     61     14
   -22     48    -29    -42
   -52     34    -24    -19

result:
   -35    -43     -8     -6
   -83    -40     61     14
   -22     48    -29    -42
   -52     34    -24    -19

instruction   8 is correct
```

matrix mul(dot product):

```
op: 101 time:   200
data1:
   -85    -43    -96    -64
   -83    -40     61     58
   -22     48    -84    -42
   -52     34    -24    -75

data2:
     2     -7    -70    -10
    87     66     33     27
   -84     28     33     25
   -43     46    -76     62

ans:
 10927 -13008   9704 -13444
-13372  18283   3010   3575
  5735  -4444  -5256  -3501
 11155  -6471  12493   4313

result:
 10927 -13008   9704 -13444
-13372  18283   3010   3575
  5735  -4444  -5256  -3501
 11155  -6471  12493   4313

instruction   9 is correct
```

以下是 instruction 中 constant 不為 0 的情況下 6 種操作的模擬結果

element add:

```
op: 000 time:   220
data1:
   -86    -36     38     32
    95    -15     81    -49
   -16     13    -73     79
   -69      6     28    -60

data2:
     8      8      8      8
     8      8      8      8
     8      8      8      8
     8      8      8      8

ans:
   -78    -28     46     40
   103     -7     89    -41
    -8     21    -65     87
   -61     14     36    -52

result:
   -78    -28     46     40
   103     -7     89    -41
    -8     21    -65     87
   -61     14     36    -52

instruction  10 is correct
```

element sub:

```
op: 001 time:   240
data1:
   -37     16    -65     91
    45    -81     44    -66
   -68     61      3    -10
   -56     79     -7    -99

data2:
     3      3      3      3
     3      3      3      3
     3      3      3      3
     3      3      3      3

ans:
   -40     13    -68     88
    42    -84     41    -69
   -71     58      0    -13
   -59     76    -10   -102

result:
   -40     13    -68     88
    42    -84     41    -69
   -71     58      0    -13
   -59     76    -10   -102

instruction  11 is correct
```

element mul:

```
op: 010 time:   260
data1:
   -60     21     -8    -14
    52    -78    -50     49
    99     75    -63      7
    81     54    -87     73

data2:
     6      6      6      6
     6      6      6      6
     6      6      6      6
     6      6      6      6

ans:
  -360    126    -48    -84
   312   -468   -300    294
   594    450   -378     42
   486    324   -522    438

result:
  -360    126    -48    -84
   312   -468   -300    294
   594    450   -378     42
   486    324   -522    438

instruction  12 is correct
```

element div:

```
op: 011 time:   280
data1:
   -69     73     90      7
    77     53    -80     91
   -55     73     85    -82
    28     50    -22     47

data2:
     5      5      5      5
     5      5      5      5
     5      5      5      5
     5      5      5      5

ans:
   -13     14     18      1
    15     10    -16     18
   -11     14     17    -16
     5     10     -4      9

result:
   -13     14     18      1
    15     10    -16     18
   -11     14     17    -16
     5     10     -4      9

instruction  13 is correct
```

element mod:

```
op: 100 time:   300
data1:
    85    -90    -20     66
    11     76      1     73
   -66     -5     49     73
    -7    -15     73     93

data2:
    10     10     10     10
    10     10     10     10
    10     10     10     10
    10     10     10     10

ans:
     5      0      0      6
     1      6      1      3
    -6     -5      9      3
    -7     -5      3      3

result:
     5      0      0      6
     1      6      1      3
    -6     -5      9      3
    -7     -5      3      3

instruction  14 is correct
```

matrix mul(dot product):

```
op: 101 time:   320
data1:
    85    -90    -20     66
    11     76      1     73
   -66     -5     49     73
    -7    -15     73     93

data2:
     6      6      6      6
     6      6      6      6
     6      6      6      6
     6      6      6      6

ans:
  -366   -366   -366   -366
   858    858    858    858
   444    444    444    444
   888    888    888    888

result:
  -366   -366   -366   -366
   858    858    858    858
   444    444    444    444
   888    888    888    888

instruction  15 is correct
```

I. Conclusions and Discussions (心得、感想、結論、及討論):

- 心得

吳伯諺	經過這學期的課程，學到更多 verilog 的語法，也透過講義上許多範例與之前作業的練習了解了各種電路要怎麼在 verilog 中表現出來，在寫這份期末專題雖然一直出現問題，但藉由寫作業經驗，最後還是有辦法把 bug 一個一個解決，也終於完成這個作業。
許瀚宇	從深度學習網路出來之後，除了減少網路的參數量、浮點數的精度以增加推倒速度外，也可以為此特別訂製專用電路，而我本次雖說沒有完全以深度學習網路特製，但這樣的設計能夠應用於任何需要矩陣運算的情境。

- 結論

透過 Strassen 演算法以及透過 generate 在 verilog 中達到遞迴的效果，我們可以將大矩陣分成若干個小矩陣再透過 Strassen 演算法，使用比一般的做法用更少的加法與乘法完成矩陣乘法的運算

References (參考資料) : (請說明各參考項目對你的專題提供那方面資料)

[1] https://hackmd.io/@siahuat0727/HJscCbWgV?type=view 矩陣運算的演算法，如 Strassen 演算法、以及 SIMD 指令集如何計算矩陣

[2] https://www.twblogs.net/a/5ef6bb318e9d5dfc3360c98e 參考現在的科技公司如 Nvidia、Google 如何處理矩陣乘法

[3] 蔡文錦 計算機組織講義 CPU architecture