下世代記憶體內運算 AI 晶片

Energy Efficient Next-Generation in-Memory Computation base AI chip

組別： 梁凱傑
指導老師：張孟凡 教授

Abstract

To implementing the deep learning neuron network on the hardware, Computing in memory is a state-of-art method to efficiently increase the computation performance. However, there exists several issues causing bottleneck for CIM computation in deep learning. To implementing cloud deep learning, which does require high-precision multiply-accumulation like floating point MAC, CIM may not perform as well as edge deep learning (edge deep learning require more integer computation instead of floating-point computation).

In order to improve the performance of the CIM, therefore, we introduce three supplement circuits to eliminate the disadvantage in floating point computation: (1) pre-alignment circuit (2) Booth encoding data conversion circuit (3) hierarchy structure for floating point mode and integer mode switching. Pre-alignment circuit makes the input of the CIM be able to operate like the integer. Thus, this circuit would make the CIM computation to be alignment free. Besides, the Booth encoding method may significantly reduce the multiplication computation process. Lastly, the hierarchy structure may combine the FP engine and INT engine. We may switch the mode of computation simply by some control signal instead of switching the engine. As a result, this structure may reduce the overhead area of the CIM. Finally we achieve a low power consumption and small area module by using Verilog simulation.

Introduction

First, let's take the overview of the whole structure shown in Fig. 1.We have already stored all the data that we are going to compute in the small storage. The exponent stored in the exponent storage and mantissa and sign storage store in the mantissa and sign storage. The pre-alignment circuit directly access the data from each storage. After aligning the data, we use the booth encoder to encode the input than send each data to the CIM Macro. CIM do the MAC computation then send the result to the Subarray Adder. Subarray Adder add the compensation sum back to the output value. Later, the Macro Accumulator shift and add the output from different cycle. After those process, we can obtain the result of the calculation.

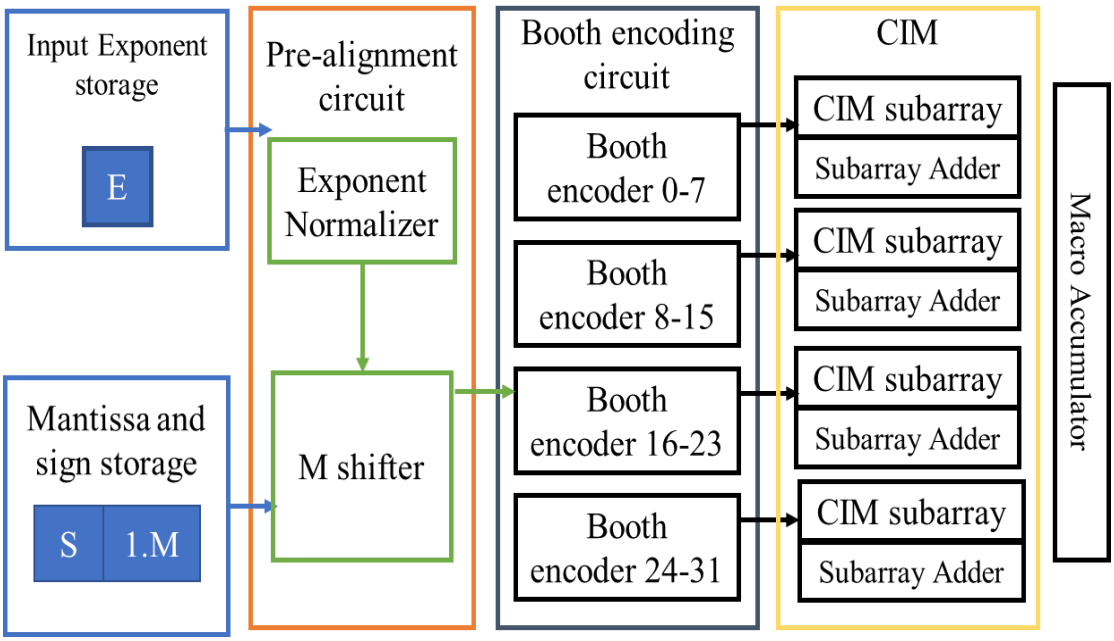

Fig. 1: Overview of the structure

(i) pre-alignment circuit

To understand the three introduced circuits, we first look at the previous CIM design. When CIM are calculating the floating point, we have the input in the form of {Exponent, Sign, Mantissa}. While we are using CIM to do the MAC operation we need to align the multiplication result before accumulation. This alignment operation is resource consuming. Thus, we come up with our first supplement circuit "pre-alignment circuit". If we can align all the input before entering the Memory, we can use the same operation of computing integers in CIM, creating the alignment-free CIM. To make a proper pre-alignment, we first find the maximum exponential value of the input and shift other input exponent relative to the maximum exponential to achieve the pre-alignment input. Another advantage of using pre-alignment method is that we may possibly lose less overflow bits in the pre-alignment system. We first access the exponent data from the input exponent storage. We have 32 input ports for each pre-alignment unit. In order to find out the maximum exponent, we construct the comparison tree to find out the maximum exponent. After finding the maximum exponent, we use the maximum exponent to subtract each exponent. The yield results would be the aligned data. We also keep the Emax value in order to calculate the final E_out. Later, we shift the mantissa base on its output value of Emax – E. The output value of the E_out will be equal to Emax + Ewmax (max Exponent of weight) – 127. This value would be the exponent of all the data computing in the Memory.

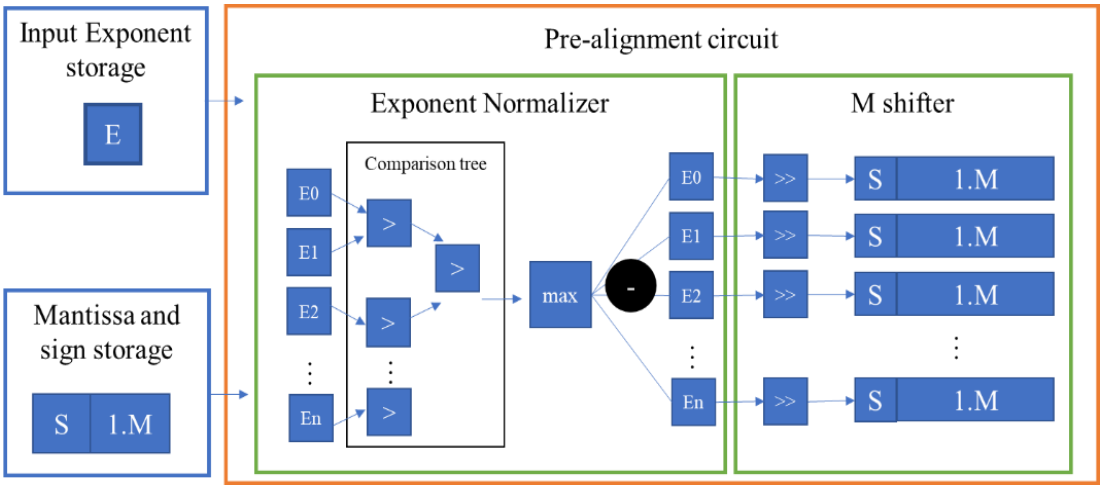

Fig. 2: The pre-alignment circuit

(ii) Booth encoder

Original multiplication required multiplying each digit and adding and shifting each digit. Introducing the Booth encoding may change the adding and shifting operation to adding/subtracting and shifting operation (Base on Radix-4 Booth). We may see the example as follows:

$$M \times 2b'011111 = M \times (2^4 + 2^3 + 2^2 + 2^1 + 2^0) = M \times 31 \quad (1)$$

$$M \times 2b'011111 = M \times 2b'(100000 - 1) = M \times (2 * 4^2 - 4^0) = M \times 31 \quad (2)$$

From this example, we may clearly see that original we need to accumulate for five time in the (1) formula. However, if we encode the data to the second form, it only requires two accumulations to obtain the final result. This alternation may reduce the number of adding operation. We have the encoder truth table as the following Table 1.

Table 1: Encoder Truth Table

i+1	i	i-1	NEG	TWO	ZERO
0	0	0	0	0	1
0	0	1	0	0	0
0	1	0	0	0	0
0	1	1	0	1	0
1	0	0	1	1	0
1	0	1	1	0	0
1	1	0	1	0	0
1	1	1	1	0	1

Having the booth encoder, our cycle for inputting data will decrease causing our amount of computation in the Memory will significantly decrease, achieving better results in CIM computation.

We use the logic circuit below to implement the function of the truth table. Our booth encoder has several output ports. They are NEG, TWO, ZERO, Compensation SUM (for each three digits we have one NEG, TWO and ZERO, but only one Compensation sum for the whole). NEG and TWO would be sent to the CIM MACRO as its input for each column of the memory. In the first cycle, we would obtain two digits of input. From the Fig2.2-3 we may know that we may obtain a new NEG, TWO and ZERO in each cycle. We encode the input in the encoder region obtaining NEG, TWO and ZERO. To calculate the compensation, we connect the out put of the encoder to another logic called compensation. From the compensation circuit, we obtain the partial compensation. After computing all the partial compensation, we add up all the compensations and obtain the compensation sum.

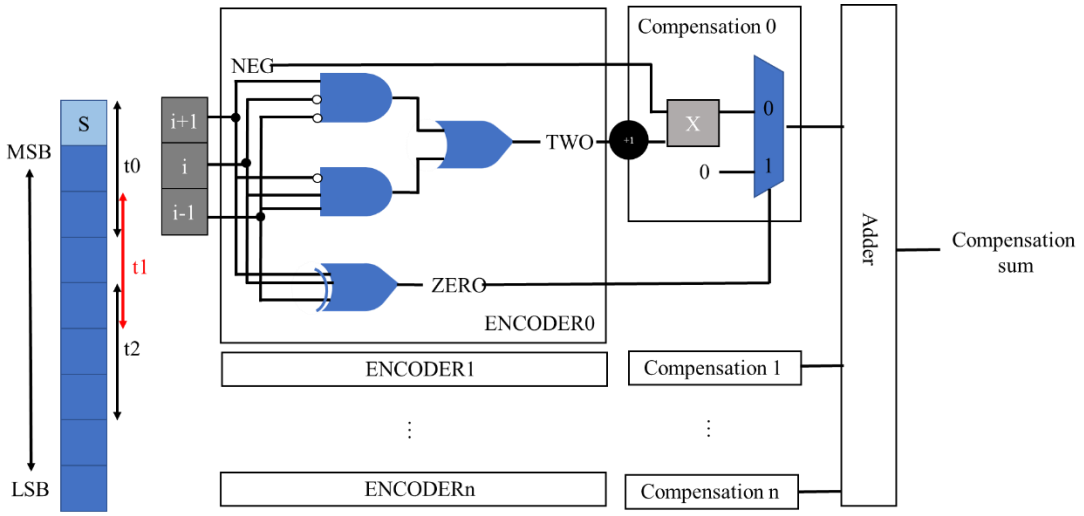

Fig.3 Booth encoder

(iii) CIM simulation & control Signal

Since we don't have the direct cell-base module of CIM to implement. We use the behavior model to simulate the SRAM CIM array. How we simulate the CIM operation is shown in the Fig.2.2-5. We may see that we have two inputs which is NEG and TWO. We have the weight already stored in each CIM element (in the figure, W is the weight). With different inputs we have related output.

After building the behavior model of the CIM elements, we construct the whole SRAM-CIM ARRAY. The configuration of the input and output of SRA M CIM is shown in Fig.4

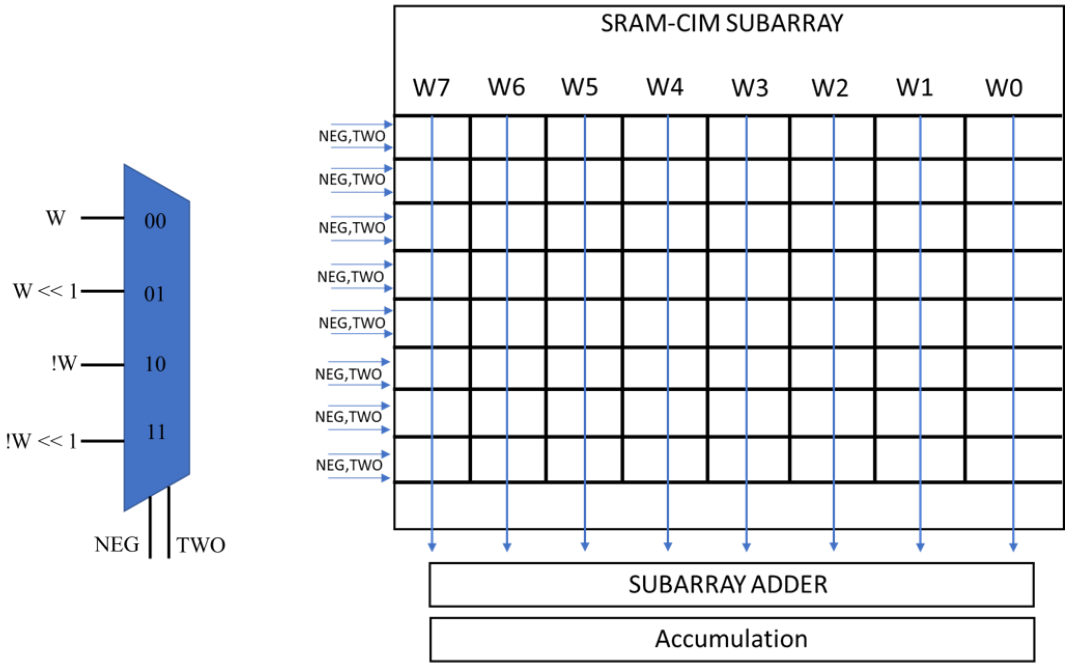

Fig.4 CIM Array

After the computation in the CIM Array, we sent the data to the subarray adder. The subarray adder transforms the input data P into the output data Q. From the Booth encode part we have the output of compensation. The compensation value is not involved in the CIM computation. We process the compensation in the Subarray Adder. In the Fig6, we may clearly see how we transform the P into Q. As w know, there are 8 different channels in each row. Thus, before entering the 12b Adder, we have to wait for all 8 data to complete transformed. The circuit of Accumulator and Subarray adder are shown in Fig.5

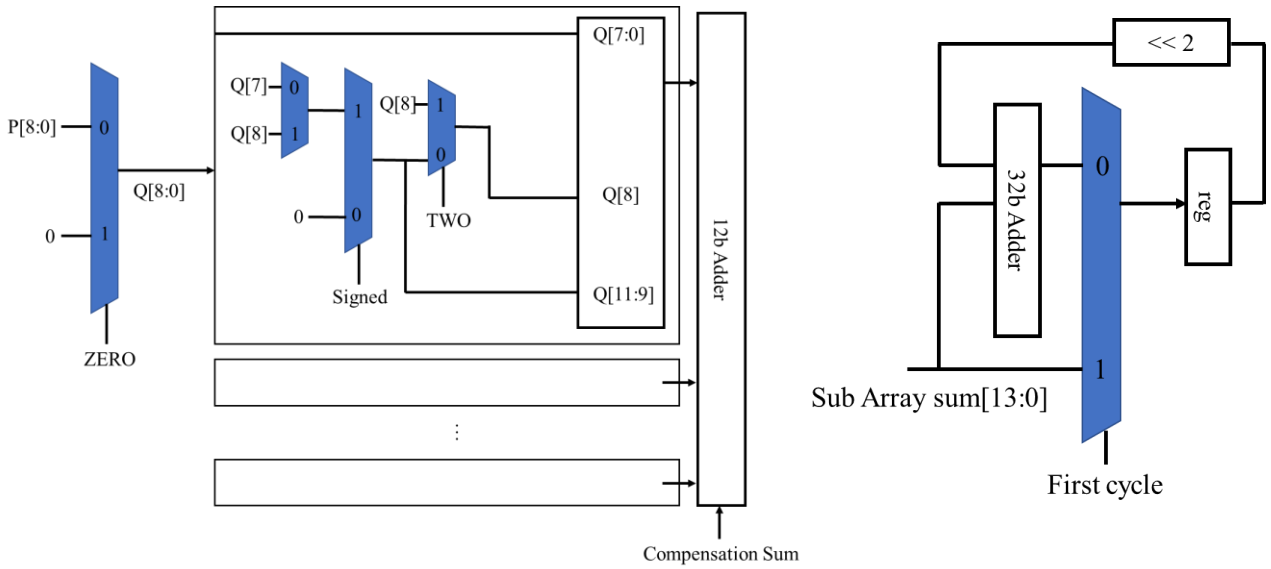

Fig.5 Accumulator and SubArray Adder

With the help of supplement circuits, we successfully overcome several problems in floating point computation in CIM. The pre-alignment circuit deduct the alignment process in CIM. The Booth encoding process reduce the computation cycle. The control signal in subarray adder helps us to switch mode without constructing two engines for floating point computation and integer computation.

Results

```
Number of ports:                         2960
Number of nets:                          8225
Number of cells:                         6502
Number of combinational cells:           5582
Number of sequential cells:               918
Number of macros/black boxes:               0
Number of buf/inv:                       1048
Number of references:                      86

Combinational area:              48326.674995
Buf/Inv area:                     3730.885143
Noncombinational area:           29166.423332
Macro/Black Box area:                0.000000
Net Interconnect area:      undefined  (No wire load specified)

Total cell area:                 77493.098327
```

Fig.5-1 The area of the simulation (um^2)

```
  Net Switching Power  = 2.868e-04   (11.32%)
  Cell Internal Power  = 2.182e-03   (86.14%)
  Cell Leakage Power   = 6.412e-05   ( 2.53%)
    Intrinsic Leakage  = 6.412e-05
    Gate Leakage       =    0.0000
                         ---------
Total Power            = 2.533e-03  (100.00%)
```

Fig.5-2 The power consumption of the simulation

Conclusion

如 Fig.5 和 Fig.6 所見，此架構的 power 消耗相當之低，一組浮點數的乘加運算的元件僅需要 2.5mW，而也因 pipeline 的架構使計算時間並不長。而 area 的部分雖然我們並沒有將硬體的 CIM Macro layout 給套用進來，但也可以看出面積還是相當小的（約 $0.077mm^2$），若我們有更多工具的話，或許可以針對這部分再做更多的優化。

而藉由這個題目，我們希望能展望軟硬體結合的成果，並且解決部分神經網路需要浮點數，或是高精度的運算需求時，這種與 CIM 結合的架構可以讓耗能跟面積等能夠得到更好的結果。另外，這個架構還可以根據神經網路的運算特色再去做更多效能的改進（Zero Skipping 等），是我們還在進行發想的部分。

Reference

[1] Fengbin Tu1, Yiqi Wang, Zihan Wu1, Ling Liang, Yufei Ding, Bongjin Kim2, Leibo Liu1, Shaojun Wei1, Yuan Xie2, Shouyi Yin1 “A 28nm 29.2TFLOPS/W BF16 and 36.5TOPS/W INT8 Reconfigurable Digital CIM Processor with Unified FP/INT Pipeline and Bitwise In-Memory Booth Multiplication for Cloud Deep Learning Acceleration” ISSCC, 2022

心得感想

在接受教授的專題訓練之前，我們對 CIM 和 VLSI 相關的領域是一無所知，所以非常榮幸有這樣的機會能夠在實驗室裡跟隨教授去學習。也謝謝學長們耐心教導，讓我們能夠使用模擬用的工具軟體去更深的理解研究論文的重點。沒有他們的幫忙，我們不可能完成這項專題。

在學長的分配下，我們的研究主題是 Digital 的相關領域。而在我們學習 CIM 和 CNN 加速器的基本概念之後，我們開始去想如何將 CIM 的結合再 CNN 的計算元件裡面，以此發揮 CIM 那省耗能也省面積的優勢。在這過程中，我們參考了許多相關論文，也從專門研究 CIM 的同期專題生以及實驗室的學長姐們討論，商量各種做法的優劣和可能會遇到的困難，這些過程都是非常寶貴的經驗。未來研究時，這樣與其他不同研究方向的夥伴合作的機會應該也會越來越多，而希望這樣的經驗能夠讓我們更有效率的達成目標。