Approximate CNN

M.Abdelmaseeh, M.Romani, K.Khallaf, M.Tarek and A.Ibrahim

December 2022

Github Link : https://github.com/MinaAbdelmaseeh/Approximate-CNN

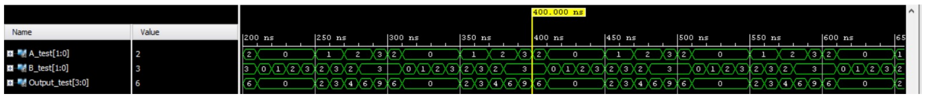
1 Introduction

Approximate computing is a powerful way to decrease the consumption power of digital integrated circuits while maintaining a decent accuracy. In this project a Convolutional neural network is implemented using both accurate and approximate paradigms. Additionally, the accuracy, the power and the area are compared.

2 Accurate 2x2 Multiplier

The implementation of the accurate multiplier providing the multiplication output of two numbers, where each number consists of two bits, is constructed using the presented configuration in the research paper [1] (such that the multiplication process is accomplished using 6 AND gates and 2 XOR gates). Then, the constructed 2bit*2bit multiplier is used to represent the building block for implementing an accurate 4bit*4bit multiplier and an accurate 8bit*8bit multiplier.

Accurate 2x2 Multiplier testbench results:



3 Approximate 2x2 Multiplier

the approximate 2x2 multiplier is implemented by reducing the output size from 4 bits to 3 bits so the result of

$$(11)_2 * (11)_2$$

is

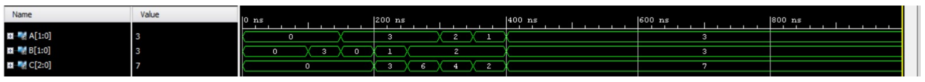
$$(111)_2$$

instead of

$$(1001)_2$$

. By this, we can save some power by reducing the umber of logic gates used ,but in the same time we sacrifice some precision and accuracy. the multiplier uses only 4 and gates and one or gate.

Approximate 2x2 Multiplier testbench results:



```
Name      Value  |0 ns          |200 ns          |400 ns   |600 ns   |800 ns
A[1:0]    3       0             3      2   1     3
B[1:0]    3       0    3   0    1   2            3
C[2:0]    7       0             3  6   4   2     7
```

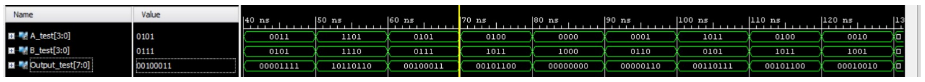
4 Building larger Multipliers

4.1 Accurate Multipliers

4.1.1 4x4 Accurate Multiplier

The accurate 4bit*4bit multiplier is constructed using the previously implemented 2bit*2bit accurate multiplier such that the output of four partial products represent the result of the multiplication process. Then, the sum of these four partial products is accomplished using a Wallace tree where the last stage is implemented using a carry ripple adder. The Wallace tree is implemented using two full-adders and two half-adders representing the first stage and another two full adders and two half adders for the second stage, while the last stage is implemented using a 4bit carry ripple adder.

4x4 Accurate Multiplier testbench results:

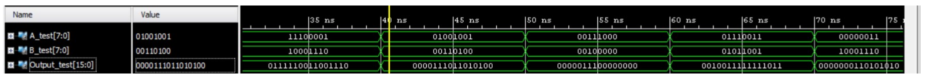


```
Name              Value     |40 ns    |50 ns    |60 ns    |70 ns    |80 ns    |90 ns    |100 ns   |110 ns   |120 ns   |13
A_test[3:0]       0101      0011      1101      0101      0100      0000      0001      1011      0100      0010
B_test[3:0]       0111      0101      1110      0111      1011      1000      0110      0101      1011      1001
Output_test[7:0]  00100011  00001111  10110110  00100011  00101100  00000000  00000110  00110111  00101100  00010010
```

4.1.2 8x8 Accurate Multiplier

The accurate 8bit*8bit multiplier is implemented using the illustrated 4bit*4bit accurate multiplier to produce the required partial products. Similarly, the sum of the partial products is accomplished using a Wallace tree, where the first stage is implemented using four full adders and four half adders, the second stage is implemented using another four full adders and four half adders, and the last stage is represented by a 10bit carry ripple adder.

8x8 Accurate Multiplier testbench results:

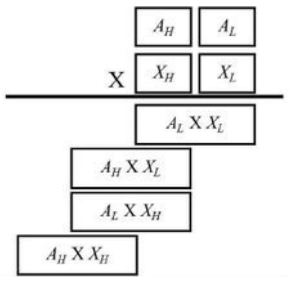


```
Name               Value             |35 ns  |40 ns  |45 ns  |50 ns  |55 ns  |60 ns  |65 ns  |70 ns  |75
A_test[7:0]        01001001          11100001          01001001          00111000          01110011          00000011
B_test[7:0]        00110100          10001110          00110100          00100000          01011001          10001110
Output_test[15:0]  0000111011010100  0111110011001110  0000111011010100  0000011100000000  0010011111111011  0000000110101010
```

4.2 Approximate Multipliers

4.2.1 4x4 Approximate Multiplier

the approximate 4x4 multiplier is an ordinary multiplier that depends on the implementation of the 2x2 inaccurate multiplier as sub-units with the method shown in the figure below.



after that the multiplicands were summed with each other using Wallace tree method as shown in the figure below. 4 Full adders with 5 half adders were utilized for the tree compared to 8 full adders and 4 half adders in the accurate 4x4 multiplier.

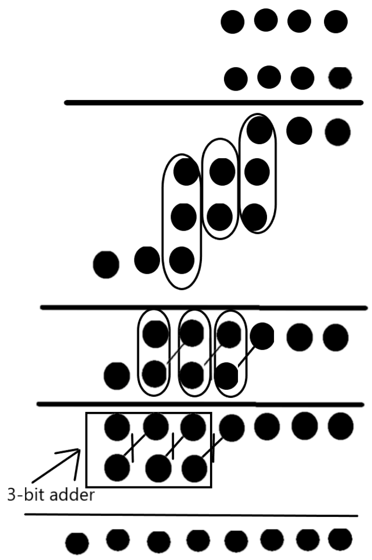
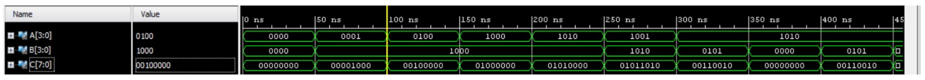


Figure 1: Wallace tree for 4x4 approx. multiplier

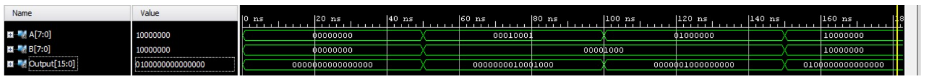
4x4 Approximate Multiplier testbench results:



4.2.2 8x8 Approximate Multiplier

the 8x8 multiplier is implemented by using the 4x4 approximate multipliers as sub-units but the Wallace tree implementation is the same as the accurate 8x8 multiplier as the output of the 4x4 approximate multiplier is 8 bits which is the same size of the output of the accurate one. The first stage is implemented using four full adders and four half adders, the second stage is implemented using another four full adders and four half adders, and the last stage is represented by a 10-bit carry ripple adder.

8x8 Accurate Multiplier testbench results:



5 Fixed Point Representation

In our implementation, an 8-bit fixed point numbers are used. we assigned 4 bits for the integer part and 4 bits for the decimal part. On multiplying two 8-bit fixed point numbers the result is as follows:

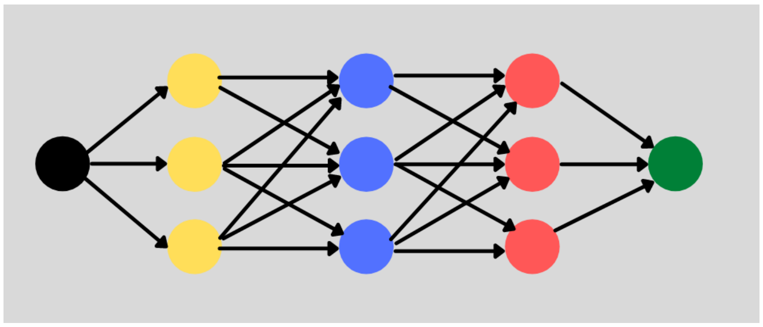
$$X_{(7:4)}.X_{(3:0)} \textbf{ x } Y_{(7:4)}.Y_{(3:0)} = M_{(15:8)}.M_{(7:0)} \tag{1}$$

On multiplying two 8-bit numbers, the result is represented in 16 bits. However, in fixed point representation the result can be truncated with a loss in precision. The result is then represented as $M_{(11:8)}.M_{(7:4)}$.

In order to take into account the overflow case, before the truncation process of the result from 16 bit to 8 bit, the number is checked if it was greater than the maximum representable number in 8 bits and the result is then the maximum representable number(either positive or negative).

The maximum representable positive number is ”0111.1111” ,while the maximum representable negative number is ”1000.0000”.

6 Convolutional Neural Network



The convolutional neural network is composed of multiple neural layers, each has an set of inputs (n) equal to the number of neurons in that layer and a set of outputs (m) equal to the inputs of the next layer. The neural layer can then be represented as the following equation

$$f(\begin{bmatrix} W_{00} & \dots & W_{0n} \\ \vdots & \ddots & \vdots \\ W_{0m} & \dots & W_{mn} \end{bmatrix} \cdot \begin{bmatrix} In_0 \\ \vdots \\ In_n \end{bmatrix} + \begin{bmatrix} B_0 \\ \vdots \\ B_n \end{bmatrix}) = \begin{bmatrix} Out_0 \\ \vdots \\ Out_m \end{bmatrix} \tag{2}$$

where $[W]$ is the weight matrix, $[B]$ is the bias vector, $[In]$ is the input to the neural layer , $[Out]$ is the output of the neural layer and f is the nonlinear activation function.

6.1 Activation function

for our implementation we used basic ReLU function defined as follows :

$$f(x) = \begin{cases} 0 & \text{if } x \leq 0 \\ x & \text{if } x > 0 \end{cases} \tag{3}$$

This can be easily implemented in 8-fixed point representation by checking on the most significant bit if it is equal to 1 ”negative number” output 0 ,otherwise output = input.

7 Results

7.1 Functional testing : Test Case

for a neural network of 4 layers (2 hidden layers) : 2 - 4 - 4 - 2 :

$$In = \begin{bmatrix} 0.C \\ 0.9 \end{bmatrix}_{16} = \begin{bmatrix} 0.75 \\ 0.5625 \end{bmatrix}_{10} \tag{4}$$

Layer 2 :

$$W_2 = \begin{bmatrix} 0.7 & 0.7 \\ 0.7 & 0.7 \\ 0.7 & 0.7 \\ 0.7 & 0.7 \end{bmatrix}_{16}, B_2 = \begin{bmatrix} 0.7 \\ 0.7 \\ 0.7 \\ 0.7 \end{bmatrix}_{16} \tag{5}$$

Layer 3 :

$$W_3 = \begin{bmatrix} 0.7 & 0.7 & 0.7 & 0.7 \\ 0.8 & 0.8 & 0.8 & 0.8 \\ 0.8 & 0.8 & 0.8 & 0.8 \\ 0.8 & 0.8 & 0.8 & 0.8 \end{bmatrix}_{16}, B_3 = \begin{bmatrix} 0.8 \\ 0.8 \\ 0.8 \\ 0.8 \end{bmatrix}_{16} \tag{6}$$

Layer 4 :

$$W_4 = \begin{bmatrix} 0.8 & 0.8 & 0.8 & 0.8 \\ 0.8 & 0.8 & 0.8 & 0.8 \end{bmatrix}_{16}, B_4 = \begin{bmatrix} 0.8 \\ 0.8 \end{bmatrix}_{16} \tag{7}$$

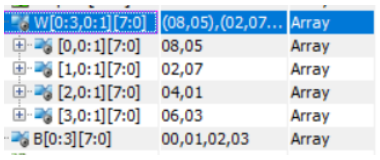
$$\textbf{Expected Out} = \begin{bmatrix} 5.6 \\ 5.6 \end{bmatrix}_{16} = \begin{bmatrix} 5.4204 \\ 5.4204 \end{bmatrix}_{10} \tag{8}$$

$$\textbf{Simulated Out (Accurate Neural network)} = \begin{bmatrix} 4.E \\ 4.E \end{bmatrix}_{16} = \begin{bmatrix} 4.875 \\ 4.875 \end{bmatrix}_{10} \tag{9}$$

$$\textbf{Simulated Out (Approx. Neural network)} = \begin{bmatrix} 4.C \\ 4.C \end{bmatrix}_{16} = \begin{bmatrix} 4.75 \\ 4.75 \end{bmatrix}_{10} \tag{10}$$

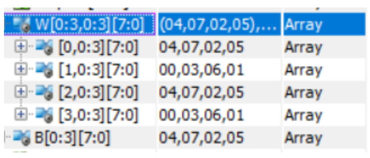
Note that the expected output is slightly different than the simulated out, this is due to the truncation of the 8x8 multiplier which results in a decrease in accuracy.

7.2 Functional Testing : more test cases ...



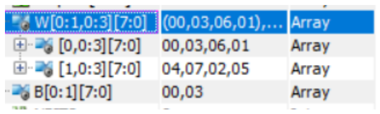
W[0:3,0:1][7:0]	(08,05),(02,07...	Array
[0,0:1][7:0]	08,05	Array
[1,0:1][7:0]	02,07	Array
[2,0:1][7:0]	04,01	Array
[3,0:1][7:0]	06,03	Array
B[0:3][7:0]	00,01,02,03	Array

Figure 2: The Weights and Biases of the 1st layer



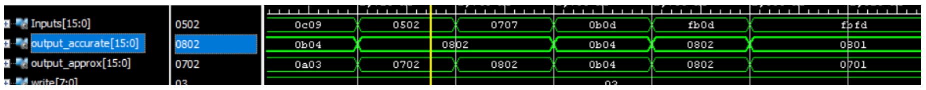
W[0:3,0:3][7:0]	(04,07,02,05),...	Array
[0,0:3][7:0]	04,07,02,05	Array
[1,0:3][7:0]	00,03,06,01	Array
[2,0:3][7:0]	04,07,02,05	Array
[3,0:3][7:0]	00,03,06,01	Array
B[0:3][7:0]	04,07,02,05	Array

Figure 3: The Weights and Biases of the 2nd layer



```
W[0:1,0:3][7:0]    (00,03,06,01),...   Array
  [0,0:3][7:0]     00,03,06,01         Array
  [1,0:3][7:0]     04,07,02,05         Array
B[0:1][7:0]        00,03               Array
```

Figure 4: The Weights and Biases of the 3rd layer



```
Inputs[15:0]           0502   0c09   0502   0707   0b0d   fb0d   fbfd
output_accurate[15:0]  0802   0b04   0802          0b04   0802   0801
output_approx[15:0]    0702   0a03   0702   0802   0b04   0802   0701
write[7:0]             03                   03
```

Figure 5: The result of the simulation. As shown in the figure, some cases occur where the output of the approximate neural network is different than the output of the accurate neural network.

7.3 Power, Area and frequency Analysis

For 8-bit multipliers, In the following table a comparison between the accurate and the approximate 8x8 multiplier is presented, As shown in the table, the approximate multiplier consume less Dynamic power by 7% and less LUTs by 46%. The Frequency chosen to operate the multipliers is 100 MHz in order to ensure low power consumption :

-	Accurate	Approximate
Static power	0.081	0.082 W
Dynamic power	0.076	0.071 W
Total power	0.157 W	0.152 W
LUTs	114	78
Frequency	100 MHz	100 MHz

For 2 - 4 - 4 - 2 neural network, The approximate power consumption is less by 6% and less LUTs by 10% than the accurate neural network and less area by 10%. The Frequency chosen to operate the neural network is 100 MHz in order to ensure low power consumption:

-	Accurate	Approximate
Static power	0.081 W	0.081 W
Dynamic power	0.098 W	0.088 W
Total power	0.179 W	0.169 W
LUTs	3347	3037
Registers	336	336
Frequency	100 MHz	100 MHz

For 2 - 10 - 10 - 2 neural network, The approximate power consumption is less by 7% and less LUTs by 13% than the accurate neural network and less area by 9%. The Frequency chosen to operate the neural network is 100 MHz in order to ensure low power efficiency:

-	Accurate	Approximate
Static power	0.082 W	0.082 W
Dynamic power	0.392 W	0.357 W
Total power	0.474 W	0.439 W
LUTs	14206	12568
Registers	1296	1296
Frequency	100 MHz	100 MHz

7.4 Accuracy estimation

In order to esitimate the accuracy of approximate neural network, we initiated several pseudo random neural networks and gathered the following results:

Input	Accurate output	Approximate output
0x0.c 0x0.9	0x0.7 0x0.9	0x0.6 0x0.9
0x0.5 0x0.2	0x0.6 0x0.7	0x0.6 0x0.7
0x0.7 0x0.7	0x0.6 0x0.7	0x0.7 0x0.9
0x0.b 0x0.d	0x0.7 0x0.9	0x0.6 0x0.9
0xf.b 0x0.d	0x0.6 0x0.6	0x0.6 0x0.6
0xf.b 0xf.d	0x0.6 0x0.6	0x0.6 0x0.6
0x2.3 0x1.6	0x0.8 0x0.d	0x0.8 0x0.c

The error is calculated as:

$$Err = \frac{\left|\mathbf{accurate} - \mathbf{approx}\right|}{\sum \mathbf{accurate}} \tag{11}$$

From the above table , the error is estimated to be around 5.82%.

Note : The results are checked using a python code that simulates the accurate neural network. The code can be found in the Github link in testbenches folder.

8 Conclusion

As it was shown in the results section, the power efficiency difference between the approximate neural networks and the accurate neural network increases as the system gets more complex. Hence, Approximate neural network proved to decrease the power consumption and area utilization by at least 10% for small circuits.

Although the ratio between the utilization of approximate and the accurate implementations of 8x8 multipliers is around 50%, The power consumption seemed to stall at 10-13% for small circuits. This is because FPGAs uses LUTs so the reduction in the schematic might not be very powerful when implemented using FPGAs. However, It is expected to have better power enhancement when implemented as an ASIC.

9 References

[1] P. Kulkarni, P. Gupta, and M. Ercegovac. 2011. Trading Accuracy for Power in a Multiplier Architecture. J.Low Power Electronics 7, 4, 490-501.