IMPLEMENTATION OF LNS BASED APPROXIMATE LOGARITHMIC MULTIPLIER USING OPERAND DECOMPOSITION METHOD

Yesodha.P[1],Roshini.S[2], Sowmiya.R[3]

[1] *Asst. Prof, ECE, Prince Shri Venkateshwara Padmavathy Engineering College, Tamil Nadu, India*
[2] *Student, ECE, Prince Shri Venkateshwara Padmavathy Engineering College, Tamil Nadu, India*
[3]*Student, ECE, Prince Shri Venkateshwara Padmavathy Engineering College, Tamil Nadu, India*

ABSTRACT

Generally multiplication consumes more power, area and are complex. Multiplication plays a vital role in implementation of many Digital Signal Processing (DSP) applications. One feasible alternative solution to simplify multiplication in the use of Logarithmic Number System, instead of binary. For exact fixed-point multiplication a matrix multiplier is usually used. The LNS based Approximate Logarithmic Multiplier implemented using Mitchell's algorithm significantly reduces area, power and time delay while having low worst-case error which are used in the application of Convolutional neural networks. Mitchell algorithm is done by taking logarithm for the two binary inputs, adding them and finally taking antilogarithm for the added result. Thus, the multiplication is converted to addition. The proposed system, operand decomposition shows a significant increase in the amount of accuracy over the existing method when applied to the logarithmic multiplication. Operand Decomposition performs Mitchell algorithm twice according to the method, in order to improve the accuracy.
.

Keyword: -*convolution neural network, Mitchell's algorithm, low worst case error, operand decomposition, digital signal processing.*

1. INTRODUCTION

Multiplication plays a significant role in Digital signal processing (DSP) applications. It uses tasks such as Finite Impulse Response filtering, Fast Fourier Transform, and Discrete Cosine Transform, which involve heavy use of arithmetic operations such as addition, multiplication, and division. These algorithms involve repetitive multiplications which require more time. In DSP applications, time is a crucial factor than accuracy. The Digital Image processing applications like medical imaging, satellite imaging, Biometric trait images etc…, rely on multipliers to improve the quality of image. Various researches are going on to optimize the multipliers in terms of speed, area and power or a combination of these parameters. The traditional multipliers use large amount of hardware and are power hungry.

1.1 LNS

One of the alternate solutions is the implementation of Logarithmic Number System (LNS) in multipliers. However, there is trade-off between the low accuracy from LNS with the computation speed. Hence these multipliers are restricted to signal processing applications where accuracy is not only of prime importance but also a certain degree of error can be tolerated.

Several approaches have been suggested in the literature for improving the efficiency of logarithm based arithmetic. They can be broadly classified as

- Look Up Table (LUT)-based interpolation and
- Mitchell’s algorithm-based logarithm computation.

1.1.1 LUT

A **LUT**, which stands for **Look Up Table**, in general terms is basically a table that determines what the output is for any given input(s). In the context of combinational logic, it is the **truth table**. This truth table effectively defines how your combinatorial logic behaves.

1.1.2 MITCHELL’S ALGORITHM-BASED LOGARITHM COMPUTATION

Mitchell’s Algorithm(MA) approach is more popular method than LUT due to less complex hardware consumption which conserves the area. Piecewise linear approximation of the log curve introduces significant percentage of error into the system. This error percentage increases relatively with the increase of number of ‘1’ bits in mantissa. By successive multiplications, error in MA algorithm can be reduced because of the fact that error is always positive in this case. Error can be approximated to arbitrary value by introducing suitable error correction with iteration.

2. PROPOSED METHOD

In this method, the proposed new design flow power logarithmic multiplication approach is aimed to achieve faster computation. Multiplication is a significant operation in signal processing but slow and complex leading to high power consumption and area. Digital Signal Processing repetitively uses multiplication to carry out computations. LNS based methods mitigate time and power but with the compromise for introducing some errors. Operand decomposition method reduces the error rate and improves the accuracy rate.

2.1 OPERAND DECOMPOSITION

The proposed, operand decomposition approach is to improve the accuracy of Mitchell’s Algorithm is described in this section. The operand decomposition approach previously has been used to reduce switching activity in binary multipliers. In this work, the use of operand decomposition is for MA based LNS multiplication. The operand decomposition of two n-bit binary numbers X and Y to four n-bit binary numbers A, B, C and D is performed using the following equations, $a = x \& y$, $b = x \mid y$, $c = (\sim x) \& y$ and $d = x \& (\sim y)$. The product of the number X and Y is then computed from the decomposed operands using the equation 1.

$$X * Y = (C * D) + (A * B) \qquad 1$$

The approach increases the number of zero bits in the decomposed multiplications and hence decreases the switching power in the multiplication operation. An example illustrating the operand decomposition approach for the binary multiplication. The example illustrated, shows the logarithmic multiplication of integers 18 and 60. Since the operand decomposition strategy increases the number of zeroes in the multiplicands, the value of the fraction part in logarithmic approximation decreases and hence the error. Secondly, the DA and other correction term based approaches tries to follow the logarithm curve closely by adding the average error slack. The operand decomposition process increases the flexibility in selecting the coefficients for these equations. Next, the explaination of the functional architecture of the Operand Decomposed (OD)-Mitchell’s logarithm multiplication algorithm.

- **DESCRIPTION-**Two inputs with n numbers of bits are taken and directed towards OD block for and the two inputs are divided into four for further process.
- **LOGARITHM-**Here multiplication is the main process. Logarithm is used for converting multiplication into addition so the hardware complexity is being reduced.
- **ADDER-**Adder is used to add the input values. Two types of adder is used here, namely half adder and full adder.
- **ZERO DETECTOR-** Zero detector is used to detect whether the inputs are zero.

- **ANTILOGARITHM-**After applying MA the result is concatenated with 1 and number of zeros is appended based on MSB position

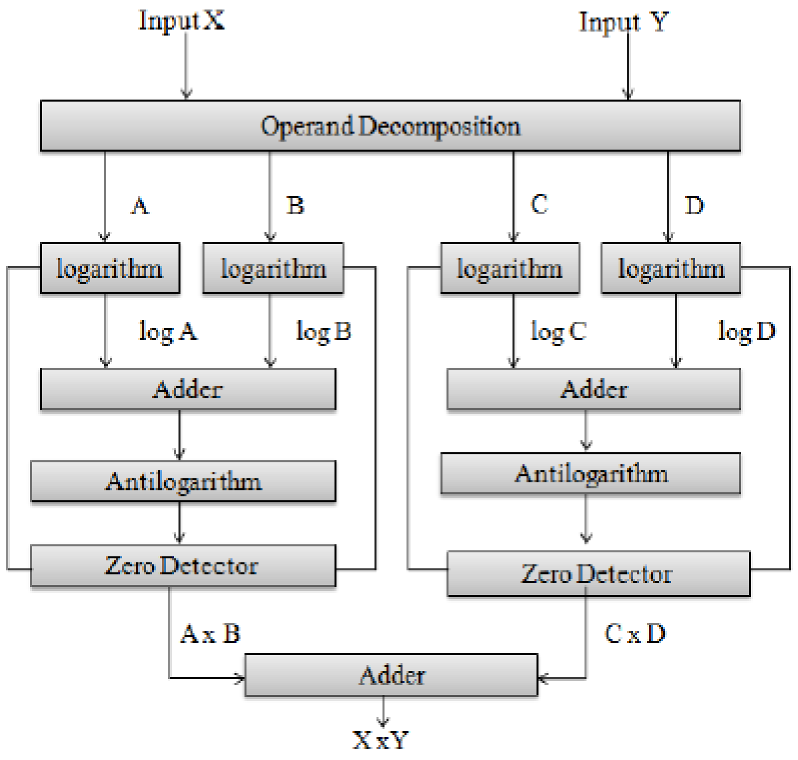


Fig 1 OD BLOCK

2.1.1 PROCESS

Two binary inputs X and Y are given in binary form. X and Y are divided into A, B, C, and D. Taking MA separately for A, B and C, D. LOD detects the leading bit position. If the leading ‘1’ is in 7th position the inputs are directly entered to the barrel shifter else ENC and NOT gate will decide the shifting count. L-Barr shift will then do the left shifting operation. Adder will do the further adding operation of the inputs. Is Zero block is used to check the two input values. So, if any one of the input is zero it will directly give the result as zero instead of doing all the operation. Here multiplication is converted as addition. The results are added and concatenated with 1 and number of zeros is appended based on MSB position.

2.1.2 OD PROCEDURE

Step1: X, Y: n-bit binary multiplicands, OP=0:2n-bits approximate product.
Step 2: Calculate A, B, C, D value using X and Y.
Step 3: Calculate A using the equation A=X|Y.
Step 4: Calculate B using the equation B=X&Y.
Step 5: Calculate C using the equation C= (~X) &Y.
Step 6: Calculate D using the equation D=X& (~Y).
Step 7: To take MA for A and B.

- A, B: n-bit binary multiplicands, OP1= 0: 2n-bits approximate product.
- Determine K1, leading ‘1’ position of 1st number, A
- Determine K2, leading ‘1’ position of 1st number, B

- Evaluate X1 by shifting A by N-K1 bits towards left
- Evaluate X2 by shifting B by N-K2 bits towards left
- Calculate K12=K1+K2
- Calculate X12=X1+X2
- Decode K12 and insert ‘1’ in that position of OP1
- Append X12 immediately after this one in OP1
- A.B= OP1

Step 8: To take MA for C and D.

- C, D: n-bit binary multiplicands, OP2= 0: 2n-bits approximate product.
- Determine K1, leading ‘1’ position of 1st number, C
- Determine K2, leading ‘1’ position of 1st number, D
- Evaluate X1 by shifting C by N-K1 bits towards left
- Evaluate X2 by shifting D by N-K2 bits towards left
- Calculate K12=K1+K2
- Calculate X12=X1+X2
- Decode K12 and insert ‘1’ in that position of OP2
- Decode K12 and insert ‘1’ in that position of OP2
- C.D= OP2

Step 9: Adding OP1 and OP2 we get OP

- OP=OP1+OP2.

2.1.3 OD EXAMPLE

Step 1: Inputs
X=10001100(140), Y=00100101(37).
Step 2: Calculate A, B, C, D value using X and Y.
Step 3: A=X|Y
A= (10001100) | (00100101)
A=10101101
Step 4: B=X&Y
B= (10001100) & (00100101)
B=00000100
Step 5: C= (~X) &Y
C= (01110011) & (00100101)
C=00100001
Step 6: D=X& (~Y)
D= (10001100) & (11011010)
D=10001000
Step 7: Take MA for A and B

- Inputs
  A=10101101,
  B=00000100
- MSB position of ‘1’ for K1 in binary form
  MSB of A is ‘7’
  K1=111
- MSB position of ‘1’ for K2 in binary form
  MSB of B is ‘2’
  K2=010
- After left shifting A
  MSB of A is 7. So left shift by 0 (~ (K1))
  X1=0101101
- After left shifting B

MSB of B is 2. So left shift by 5 (~ (K2))
X2=0000000.

- Adding K1 and K2
  K12=K1+K2
  K12=111+010=1001.
- Adding X1 and X2
  X12=X1+X2
  X12=0101101+0000000
  X12=0101101.
- Concatenate 1 an X12
  OP1= (1, X12)
  OP1=1010110100

Step 8: Take MA for C and D

- Inputs
  C=00100001,
  D=10001000.
- MSB position of ‘1’ for K1in binary form
  MSB of C is ‘5’
  K1=101
- MSB position of ‘1’ for K2in binary form
  MSB of D is ‘7’
  K2=111
- After left shifting C
  MSB of C is 5. So left shift by 2 (~ (K1))
  X1=00001
- After left shifting D
  MSB of D is 7. So left shift by 0 (~ (K2))
  X2=00010
- Adding K1 and K2
  K12=K1+K2
  K12=101+111=1001
- Adding X1 and X2
  X12=X1+X2
  X12=00001+00010
  X12=00011
- Concatenate 1 an X12
  OP2= (1, X12)
  OP2=1000110000000

Step 9: Adding OP1 and OP2
OP=OP1+OP2
OP=0000001010110100+001000110000000
OP=0010110000110100(5172)

3. ERROR PERCENTAGE

It can be obtained

- **Error %= ((Original value-Obtained value)/Original value)*100**
  Error %= ((5180-5172)/5180)*100
  Error %= 0.15

ACCURACY%=100-ERROR%

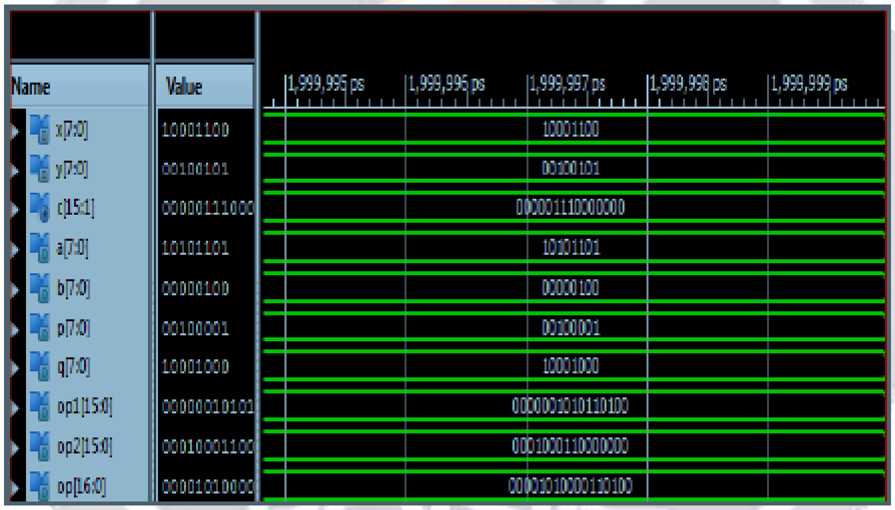
ACCURACY%=100-0.15=99.85.

X	Y	MA ERROR%	OD ERROR%
140	37	1.15	0.15
117	157	5.92	1.57
203	183	10.41	0.76

TABLE 1 COMPARISON BETWEEN MITCHELL'S AND OPERAND DECOMPOSITION ALGORITHM

4. RESULT

The simulation of OD algorithm in which the variable x indicates the input X and y indicates the input Y. The variable a,b,p,q indicates the splitted output of the input X and Y. OP1 indicates the mitchell's output of splitted variable of a and b whereas the OP2 indicates the mitchell's output of splitted variable of p and q. OP indicates the final output of OP simulation which is the added value of the OP1 and OP2



```
Name        Value          1,999,995 ps  1,999,996 ps  1,999,997 ps  1,999,998 ps  1,999,999 ps
x[7:0]      10001100       10001100
y[7:0]      00100101       00100101
c[15:1]     00000111000    000001110000000
a[7:0]      10101101       10101101
b[7:0]      00000100       00000100
p[7:0]      00100001       00100001
q[7:0]      10001000       10001000
op1[15:0]   00000010101    0000001010110100
op2[15:0]   00010001100    0001000110000000
op[16:0]    00001010000    00001010000110100
```

FIG 2 SIMULATION OF OD

5. CONCLUSION

In this work, we use operand decomposition for Mitchell algorithm based LNS multiplication. The proposed operand decomposition approach to improve the accuracy of Mitchell's Algorithm is described. Further the few methods which could make the system much better in terms of power and area could be implemented here by using various algorithms so that the throughput of the system will be increased and can be further used in many applications which involves multiplication in it. Also the methods of obtaining maximum accuracy without making the system complex

6. REFERENCE

[1]. Babic, Z. and Avramovic, A. and Bulic, P.(2010) 'An iterative logarithmic multiplier'.

[2].Babic, Z. and Avramovic, A. and Bulic, P.(2008) 'An Iterative Mitchell's Algorithm Based Multiplier'.

[3].Deeksha, R.S. and Patil, S.(2013) 'Improving Accuracy in Mitchell's Logarithmic Multiplication Using Iterative Multiplier for Image Processing Application'.

[4]. Durgesh, N. and Kanungo, J. and Mahajan, A.(2017) ‘An efficient VLSI architecture for Iterative Logarithmic Multiplier’.

[5].Khalid, H.A. and Raymond, E.S. (2003) ‘CMOS Implementation of low power logarithmic converter’.