

Design and Implementation of LNS based Approximate Multiplier using Mitchell's Algorithm and Operand Decomposition

G. Mohanapriya¹, M. Nagiga Nasirin², Mrs. J. Mary Suji Mol³

^{1,2}Department of Electronics and Communication Engineering, Jeppiaar SRR Engineering College, Tamil Nadu, India

³Assistant professor, Dept. of Electronics and Communication Engineering, Jeppiaar SRR Engineering College, Tamil Nadu, India

Abstract - In this project we study the approximate logarithmic multiplier which is implemented via low power. It is mainly used to improve the consumption of convolutional neural networks for image classification. The main advantage of logarithmic multiplier is intrinsic tolerance to error. It converts multiplications into additions. Hence this paper shows improvement in accuracy and efficient logarithmic approach.

Key Words: Convolutional Neural Networks, Logarithm, Antilogarithm

1. INTRODUCTION

Mitchell algorithm (MA) is used along with Operand Decomposition (OD) in order to increase the accuracy and decrease the error percentage. First the inputs are divided into four using some basic logic operations. This is the main reason for the increase in accuracy of the existing system compared to Operand Decomposition. This method does not give the accurate output but gives the nearest result value compared to Mitchell Algorithm alone. Hence it is very much useful for applications like Convolutional Neural Networks (CNN), embedded systems, Digital Image Processing (DIP), Digital Signal Processing (DSP) and datacenters.

These applications needs low time delays and they don't need accurate exact values. So, Mitchell Algorithm is used here but near accurate values are needed in some applications of digital image processing. Hence Operand Decomposition adds some accuracy to the same applications by combining with Mitchell Algorithm. Mitchell Algorithm and Mitchell Algorithm with Operand Decomposition involves the same process but few steps may vary due to the increase in accuracy. The comparison of MA and MA with OD is clearly revealed in this project.

1.1 LOGARTHMIC NUMBER SYSTEM

Many applications like Digital Image Processing (DIP), Digital Signal Processing requires multiplication for calculations. Fixed point number systems gives multiplication but with circuit complexity. Logarithmic Number System (LNS) will overcome this circuit complexity by introducing some errors. These errors will makes the results less accurate.

Operand Decomposition will increase the accuracy level of results by performing logical operations of the input before performing the algorithm.

1.2 MITCHELL ALGORITHM

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.

2. DESIGN AND IMPLEMENTATION

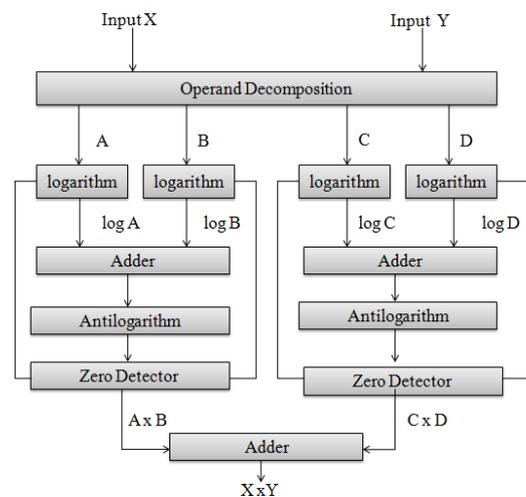


Fig -1: OD block

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.

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.

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 $C=(\sim X)\&Y$.

Step 6: Calculate D using the equation $D=X\&(\sim Y)$.

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
- Append X12 immediately after this one in OP2
- C.D= OP2

Step 9: Adding OP1 and OP2 we get OP

- $OP=OP1+OP2$.

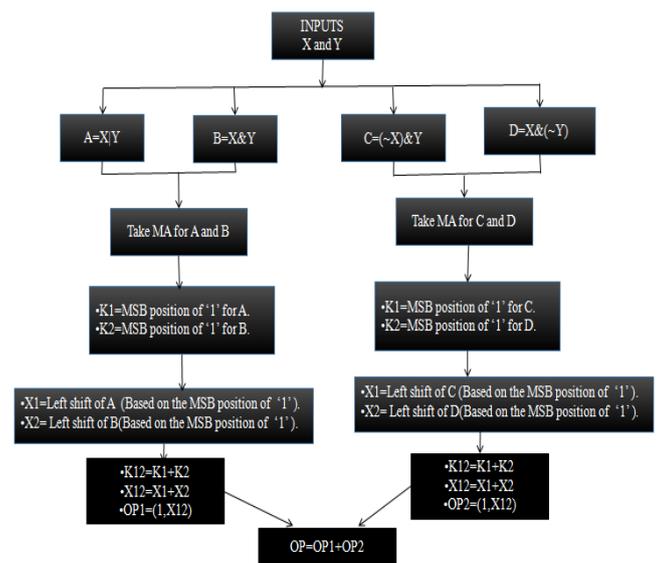


Fig -2: Flowchart

ERROR PERCENTAGE-It can be obtained by the following formula

$$\text{Error \%} = \frac{((\text{Original value}-\text{Obtained value})/\text{Original value}) * 100$$

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: $C = (\sim X) \& Y$

$$C = (01110011) \& (00100101)$$

$$C = 00100001$$

Step 6: $D = X\&(\sim Y)$

$$D = (10001100) \& (11011010)$$

$$D = 10001000$$

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 K1 in binary form

MSB of C is '5'

$$K1 = 101$$

- MSB position of '1' for K2 in 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

$$OP = OP1 + OP2$$

$$OP = 0000001010110100 + 001000110000000$$

$$OP = 0010110000110100 (5172)$$

$$\text{Error \%} = ((5180 - 5172) / 5180) * 100$$

$$\text{Error \%} = 0.15$$

Table-1: Comparison between Mitchell Algorithm and Operand Decomposition

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

The above table shows the comparison of errors between Mitchell Algorithm and Operand Decomposition. From this we came to know the efficiency of Operand Decomposition.

SOFTWARE REQUIREMENTS

Synthesis tool- Xilinx ISE 14.5.

Verification tool- ModelSim 6.4c.

SNAPSHOTS

Picture tells everything. The below snapshots are very much helpful for us in understanding about this paper. The snapshots of output waveform, Look Up Tables (LUT) and blocks are listed below.



Fig-3: Output waveform

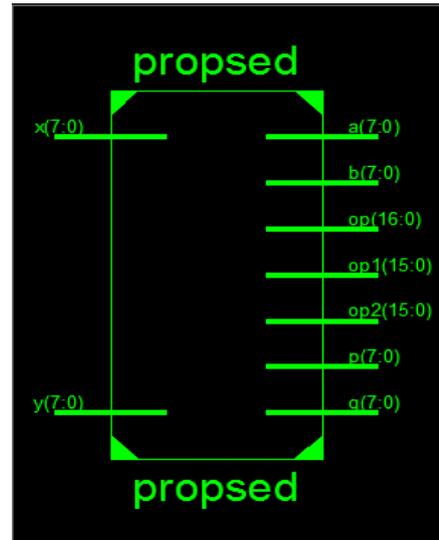


Fig-4: Block diagram

This output and the block diagram clearly shows the two inputs and the output.

The above diagram shows the several blocks inside the main block which are responsible for several operations.

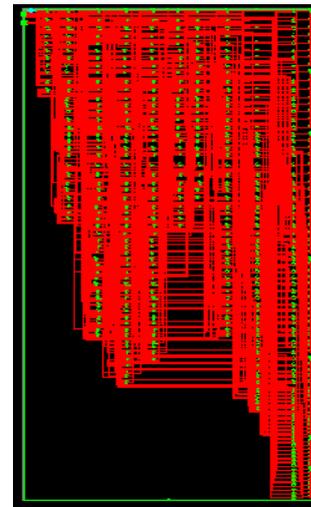


Fig-5 LUT

APPLICATIONS

- Digital Image Processing.
- Digital Signal Processing.
- Embedded systems and data centres.

3. CONCLUSION

From this we can clearly understand about multipliers. By reading this report it is also easy to analyze and choose the correct method for applications. If we need speed with some accuracy Mitchell Algorithm along with Operand Decomposition is the right technique to choose.

REFERENCES

- [1] Z. Babic', A. Avramovic', P. Bulic' (2010) An iterative logarithmic multiplier.
- [2] DurgeshNandan, JitendraKanungo, Anurag Mahajan (2017) An efficient VLSI architecture for Iterative Logarithmic Multiplier.
- [3] John n. Mitchell, jr. t associate, ire (1962) Computer Multiplication and Division Using Binary Logarithms.
- [4] UrosLoric, PatricioBulic (2012) Applicability of approximate multipliers in hardware neural networks.
- [5] ZdenkaBabic, AleksejAvramovic, PatricioBulic (2008) An iterative Mitchell's Algorithm Based Multiplier.