Design of Vedic Multiplier By Using Neural Network Technology & Mac

B. Yerriswamy[1], E. Ramakrishna[2]
[2] Assistant Professor, HOD Dept of ECE
[1, 2] SKD Engineering College, Gooty

Abstract- *An Artificial neural network (ANN) is parallel Information processing structure consists of processing units. The processing unit decides while the network is efficient or not. So it is needed to design an efficient processing unit which provides better performance. The processing unit consists of MAC unit (Multiplication and Accumulation) and Activation unit. In an existing system, the processing MAC unit was designed by Booth multiplier and carry look ahead adder. The existing processing unit provides delay and consumes more area and power. To overcome the drawbacks, a new processing unit, with Vedic multiplier consisting of square root carry select adder (SQRT-CSLA) is designed. The proposed design overcomes the drawbacks of the existing system, and it also provides better performance to the entire network. The Activation function unit was designed by sigmoid neurons process. Entire processing unit was implemented and verified by using Verilog HDL language.*

Keywords- Artificial Neural Networks (ANN), MAC, Vedic multiplier, SQRT-CSLA, Booth multiplier, Verilog HDL.

I. INTRODUCTION

1NEURAL NETWORK

A neural network is interconnected with processing elements. The processing Element of the network is used to store the interconnection strength and weights. Neural networks are widely used for statistical analysis and data modeling techniques. Some examples are image and speech recognition, character recognition, financial prediction and geological survey. In the neural network consider the input as high dimensional and discrete or real valued function, the same way output function is also discrete or real vector-valued function.

An artificial neuron computational model is similar to the natural neurons. Natural neuron receives signals through synapses located on the membrane of the neurons. When the signal is received are enough, the neuron is activated and emits the signal though the axon. Likewise the signal sent to another synapse and might activate other neurons. Modeling the artificial neurons is done by multiplying the input (synapses), with weights, and then computed by mathematical function which determines the activation of the neurons, and compute the output of the artificial neurons. ANNs combine artificial neurons in order to process the information. It uses a distributed representation of the information stored in the network. Normally the neural network model takes an input samples and produces output samples. The relationship between the input and output function is determined by the network.

Artificial neural network (ANN) is characterized by a large number of simple processing neuron like processing elements. Three different processing elements in the networks are processing units and topology. Processing units are generally MAC and activation unit. Based on the processing unit the performance of the network is increased. Two topologies are in the ANNs, one is feed forward networks and another one is feedback or recurrent networks. Feed forward network consists of single layer, multilayer perception and radial basis function. Likewise the feedback networks consists of competitive networks, Hopfield network and ART models

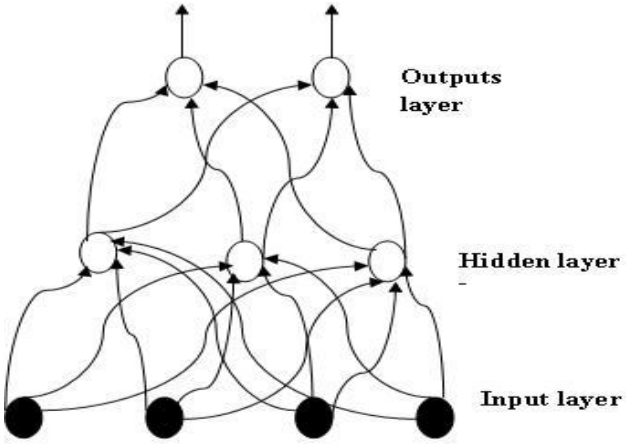
ANN can be dividing into feed forward and feedback network. In the feed forward network the input is directly feed to processing unit, after the completion of process forward to the output unit. The operation of the feed forward network shows the output is purely depends on present input only, not a previous one. But the feedback network is differ from feed forward, the output of the feedback network is depends on past output also. The output of the previous stage is taken as the feedback and given to the input unit. Application of the feed forward Networks is to develop nonlinear models that are used for pattern recognition and classification.

ARTIFICAL NEURAL NETWORKS (ANNs)

- **Neural Network Architecture**

An Artificial Neural Network is a parallel information processing consists ofprocessing units. The neural network was changed to Artificial Neural Networks, because it's not dealing with biological neural networks. ANN was deals with general computing architecture known as

Multiple Instruction Multiple Data (MIMD) parallel processing architecture.



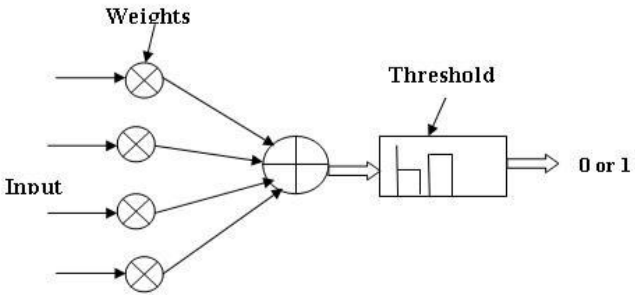
Simple neural network

Fig. shows the architecture of simple neural network, it contains three layers, input, hidden and output layer. Input data is feed forward to the output via the hidden layer. In-between the input and output, processing is performed by the help of processing unit.

B. Framework for ANN model

- There are different ANN models but each model can be specified by the following aspects:
- A set of processing units
- A state of activation for each unit
- An output for each unit
- Topology of the network
- An activation rule to update the activities of each unit
- An external environment provides information to the network
- A learning rule to modify the structure of connectivity by using information provided by the external environment.

After the processing of information, the output function uses the activation value to calculate the output of the unit. A simple artificial neuron tells how the process is done in the processing MAC unit. MAC operation is important one to get an accurate results from the neural networks.



Simple artificial neuron

Fig . shows the simple artificial neuron in the artificial neural network. It contains multiplication and accumulation unit. Each input are multiplied with weights individually, output from the multiplier is added by using addition unit. Added output is forward to activation unit, based on the threshold it produced the output 0 or 1.

II. LITERATURE SURVEY

EXISTING SYSTEM

In existing system, the processing MAC unit was designed by Booth multiplier and carry look ahead adder. The existing processing unit provides delay and consumes more area and power.

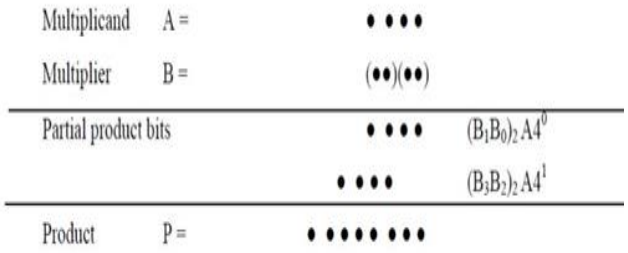
Booth Multiplier

It is a powerful algorithm for signed-number multiplication, which treats both positive and negative numbers uniformly. For the standard add and shift operation, each multiplier bit generates one multiple of the multiplicand to be added to the partial product. If the multiplier is very large, then a large number of multiplicands have to be added. In this case the delay of multiplier is determined mainly by the number of additions to be performed. If there is a way present to reduce the number of the additions, the performance will get better.

Booth algorithm is a method that will decrease the number of multiplicand multiples. For a given range of numbers to be represented, a higher representation radix leads to fewer digits. Since a k-bit binary number can be interpreted as K/2-digit radix-4 number, a K/3-digit radix-8 number, and so on, it can deal with more than one bit of the multiplier in each cycle by using high radix multiplication. This is shown for Radix-4 in the example below.

As shown in the Fig 2.1, if multiplication is done in radix 4, in each step, the partial product term (Bi+1Bi)2 A needs to be formed and added to the cumulative partial product. Whereas in radix-2 multiplication, each row of dots

in the partial products matrix represents 0 or a shifted version of A must be included and added. Table 1below is used to convert a binary number to radix-4 number.



Radix-4 multiplication in dot notation

Initially, a "0" is placed to the right most bit of the multiplier. Then 3 bits of the multiplicand is recoded according to table below or according to the following equation: $Zi = -2x_{i+1} + x_i + x_{i-1}$

Example:

Multiplier is equal to 0 1 0 1 1 10

then a 0 is placed to the right most bit which gives 0 1 0 1 1 10 0

the 3 digits are selected at a time with overlapping left most bit as follows:

Radix-4 Booth recoding

Xi+1	X	Xi-1	Zi/2
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	2
1	0	0	-2
1	0	1	-1
1	1	0	-1
1	1	1	0

For example, an unsigned number can be converted into a signed-digit number radix 4:(10 01 11 01 10 10 11 10)2 = (–2 2 –1 2 –1 –1 0 –2)4

Booth Multiplier Algorithm

Step 1: From two operands, determine the least transitions between bit and these operand assigned to X and other operand to Y. In addition to this, determination of –Y will helps to further process.

Step 2: Make a four column to understand the data flow of the multiplication. The first column (U) holds the results from each step in the algorithm. The second column (V) holds the overflow from U when right-shifting. The third column holds the operand (X). The fourth column (X-1) holds the least significant bit from X before RSC. Initially set this to zero.
Step 3: Analyze the least significant bit of X and (X-1) in each step of algorithm. From that string take the operation as in table.

Decision making operation in Booth multiplier

Least Significant Bit of X	**X-1**	**Action**
0	0	No Action
0	1	Add Y to the value in U and right-shift the result
1	0	Subtract Y from the value in U and right-shift the result
1	1	Right-shift the value in U 1 bit position

This process repeats until the X has been RSC to its original position.

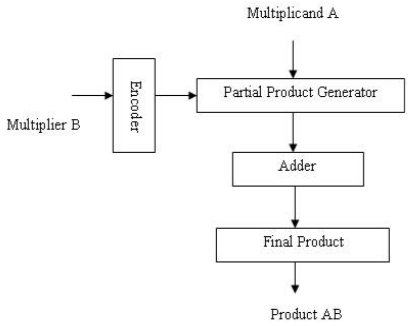
For instance, consider X= 0100 and Y= 1010. From this two operands, 1101 has less bit transitions so set X= 0100 and Y= 1010. The operation of multiplication by using booth algorithm is as follows

Take U and V together, we get 11101000 which are -24. i.e. 4X (-6) =-24.

Similarly, we can easily design the 8-bit, 16-bit and 32-bit multiplication based on booth algorithm. When implementing this algorithm through VLSI system design, high speed and low power has been obtained than normal array multiplier. This multiplier is one of the best multipliers in MAC design and digital FIR filter. Large endeavors have been working in booth multiplier to improve the performance of digital FIR filter. Further to improve the performance booth multiplier, modified booth multiplier has been developed in later. Description of modified booth multiplier is arrived in next section.

Modified Booth Multiplier

Booth multiplication consists of three major steps as partial product generation (booth recoding, reducing the partial product in two rows and addition of partial product that finalize the result for multiplication. For better understanding of modified booth multiplier, we must know each block of booth algorithm for multiplication process. The modified booth multiplier is used to perform high speed multiplication by using modified booth algorithm. In modified booth multiplier, computation time and logarithm of word length of operand are proportional to each other. Traditionally radix-4 booth algorithm is used to increase the speed of multiplier and reduces the area of the multiplier circuit. Every third column of booth multiplier table is multiplied by 0 or +1 or +2 or -1 or -2 in modified booth modifier instead of multiplying 0 or 1 after shifting and adding of every column of booth multiplier. Thus, half of the partial product can be reduced in modified booth multiplier. Grouping is started from least significant bit (LSB), in which only two bits of the booth multiplier are used and zero is padded as third bit.



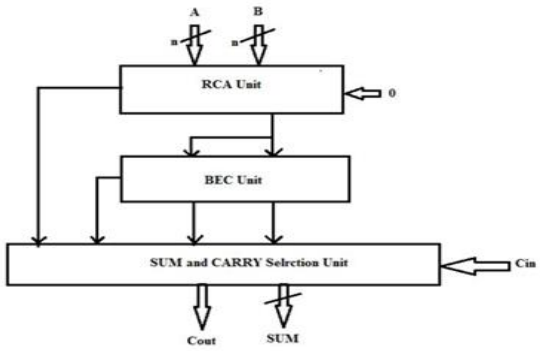
Modified Booth multiplier

Based on this booth recoding table, further the multiplication process is preceded as booth algorithm. The procedure for both booth and modified booth multiplier is represented in Fig 2.2. The performance of modified booth multiplier is better than booth multiplier in terms of LUT and Slices because half of the partial product can be reduced in modified booth multiplier. Next to modified booth multiplier, efficient multiplier structure called Vedic multiplier is developed in later for improve the performance in terms of area, delay and power.

Adder

Carry look ahead adder is being used in existing system. The Carry look ahead adder provides delay and consumes more area and power due to which efficiency of the system will get reduced.

To reduce the area and power consumption of regular CSLA, RCA unit with Cin =1 is replaced with BEC unit as shown in Fig 2.3. An (n+1) bit BEC replaces the n-bit RCA. The RCA calculate n-bit sum and carry corresponding to Cin=0. The BEC unit receives sum and carry from the RCA and generate (n+1) bit excess-1 code. The most significant bit (MSB) of BEC represents Cout and least significant bit (LSB) represents the sum corresponding to Cin =1.



Binary to Excess convertor based CSLA

PROPOSED SYSTEM

The existing processing unit provides delay and consumes more area and power, to overcome this, a new processing unit is designed which contains Vedic multiplier with square root carry select adder (SQRT-CSLA). The proposed design overcomes the drawbacks of the existing system, and it also provides better performance to the entire network. The Activation function unit was designed by sigmoid neurons process. Entire processing unit was implemented and verified by using Verilog HDL language.

Vedic Mathematics

Vedic mathematics is part of four Vedas (books of wisdom). It is part of Sthapatya- Veda (book on civil engineering and architecture), which is an upa-veda (supplement) of Atharva Veda.

It covers explanation of several modern mathematical terms including arithmetic, geometry (plane, co-ordinate),

trigonometry, quadratic equations, factorization and even calculus. His Holiness JagadguruShankaracharyaBharati Krishna Teerthaji Maharaja (1884-1960) comprised all this work together and gave its mathematical explanation while discussing it for various applications. Swamiji constructed 16 sutras (formulae) and 16 Upa sutras (sub formulae) after extensive research in Atharva Veda. The very word "Veda" has the derivational meaning i.e. the fountainhead and illimitable storehouse of all knowledge. Vedic mathematics is the name given to the ancient system of mathematics or, to be precise a unique technique of calculations based on simple rules and principles with which many mathematical problems can be solved, be it arithmetic, algebra, geometry or trigonometry.

The system is based on 16 Vedic sutras or aphorisms, which are actually word formulae describing natural ways of solving a whole range of mathematical problems. The beauty of Vedic mathematics lies in the fact that it reduces the cumbersome-looking calculations in conventional mathematics to a very simple one. This is so because the Vedic formulae are claimed to be based on the natural principles on which the human mind works. This is a very interesting field and presents some effective algorithms which can be applied to various branches of engineering such as computing and digital signal processing.

Vedic Mathematics existed in ancient India and was rediscovered by a popular mathematician, Sri Bharati Krishna Tirthaji. He divided Vedic mathematics into 16 simple sutras (formulae). These Sutras deal with Arithmetic, Algebra, Geometry, Trigonometry, Analytical Geometry etc. The simplicity in the Vedic mathematics sutras paves way for its application in several prominent domains of engineering like Signal Processing, Control Engineering and VLSI.

1. (Anurupye) Shunyamanyat -If one is in ratio, the other is zero.
2. ChalanaKalanabyham -Differences and similarities.
3. EkadhikinaPurvena- By one more than the previous One.
4. EkanyunenaPurvena -By one less than the previous one.
5. Gunakasamuchyah-Factors of the sum is equal to the sum of factors.
6. Gunitasamuchyah-The product of sum is equal to sum of the product.
7. NikhilamNavatashcaramamDashatah -All from 9 and last from 10.
8. ParaavartyaYojayet-Transpose and adjust.
9. Puranapuranabyham -By the completion or noncompletion.
10. Sankalana- vyavakalanabhyam -By addition and by subtraction.
11. ShesanyankenaCharamena- The remainders by the last digit.
12. ShunyamSaamyasamuccaye -When the sum is same then sum is zero.
13. Sopaantyadvayamantyam -The ultimate and twice the penultimate.
14. Urdhva-tiryakbhyam -Vertically and crosswise.
15. Vyashtisamanstih -Part and Whole.
16. Yaavadunam- Whatever the extent of its deficiency.

Vedic Mathematics can be bifurcated into 16 different sutras to perform mathematical operations. Among these surtrastheUrdhwaTiryakbhyam Sutra is one of the most highly preferred algorithms for performing multiplication. The algorithm is competent enough to be employed for the multiplication of integers as well as binary numbers. The term "UrdhwaTiryakbhyam" originated from 2 Sanskrit words Urdhwa and Tiryakbhyam which mean "vertically" and "crosswise" respectively. The mainadvantage of utilizing this algorithm in comparison with the existing multiplication techniques, is the fact that it utilizes only logical "AND" operations, half adders and full adders to complete the multiplication operation. Also, the partial products required for multiplication are generated in parallel and apriority to the actual addition thus saving a lot of processing time.

URDHWA TIRYAKBHYAM ALGORITHM

Let us consider two 8 bit numbers X7-X0 and Y7-Y0, where 0 to 7 represent bits from the Least Significant Bit (LSB) to the Most Significant Bit (MSB). P0 to P15 represent each bit of the final computed product. It can be seen from equation (1) to (15), that P0 to P15 are calculated by adding partial products, which are calculated previously using the logical AND operation. The individual bits obtained from equations (1) to (15), in turn when concatenated produce the final product of multiplication which is depicted in (16).The carry bits generated during the calculation of the individual bits of the final product are represented from C1 to C30. The carry bits generated in (14) and (15) are ignored since they are superfluous.

$$P0 = A0 * B0 \quad (1)$$

$$C1P1 = (A1 * B0) + (A0 * B1) \quad (2)$$

$$C3C2P2 = (A2 * B0) + (A0 * B2) + (A1 * B1) + C1 \quad (3)$$

$$C5C4P3 = (A3 * B0) + (A2 * B1) + (A1 * B2) + (A0 * B3) + C2$$

$$C7C6P4 = (A4 * B0) + (A3 * B1) + (A2 * B2) + (A1 * B3) + (A0 * B4) + C3 + C4 \quad (5)$$

$$C10C9C8P5 = (A5 * B0) + (A4 * B1) + (A3 * B2) + (A2 * B3) + (A1 * B4) + (A0 * B5) + C5 + C6 \quad (6)$$

$$C13C12C11P6 = (A6 * B0) + (A5 * B1) + (A4 * B2) + (A3 * B3) + (A2 * B4) + (A1 * B5) + (A0 * B6) + C7 + C8 \quad (7)$$

$$C16C15C14P7 = (A7 * B0) + (A6 * B1) + (A5 * B2) + (A4 * B3) + (A2 * B5) + (A1 * B6) + (A0 * B7) + C9 + C11 \quad (8)$$

$$C19C18C17P8 = (A7 * B1) + (A6 * B2) + (A5 * B3) + (A4 * B4) + (A3 * B5) + (A2 * B6) + (A1 * B7) + C10 + C12 + C14 \quad (9)$$

$$C22C21C20P9 = (A7 * B2) + (A6 * B3) + (A5 * B4) + (A4 * B5) + (A3 * B6) + (A2 * B7) + C13 + C15 + C17 \quad (10)$$

$$C25C24C23P10 = (A7 * B3) + (A6 * B4) + (A5 * B5) + (A4 * B6) + (A3 * B7) + C16 + C18 + C20 \quad (11)$$

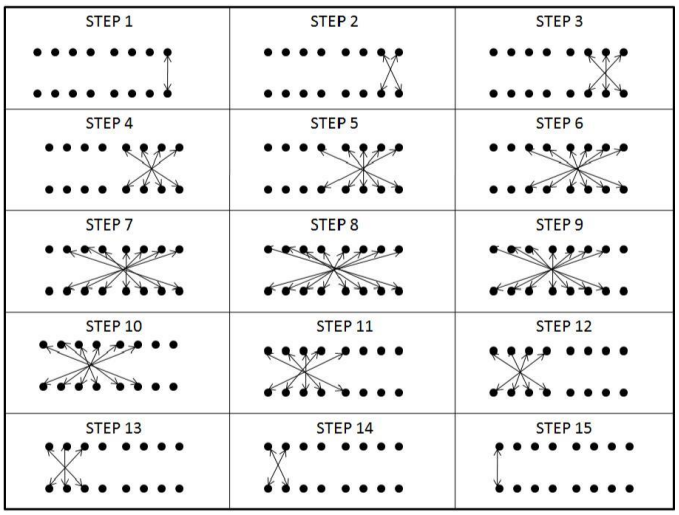
$$C27C26P11 = (A7 * B4) + (A6 * B5) + (A5 * B6) + (A4 * B7) + C19 + C21 + C23 \quad (12)$$

$$C29C28P12 = (A7 * B5) + (A5 * B6) + (A5 * B7) + C22 + C24 + C26 \quad (13)$$

$$C30P13 = (A7 * B6) + (A6 * B7) + C25 + C27 + C28$$

$$P14 = (A7 * B7) + C29 + C30 \quad (15)$$

$$P15 = (A7 * B7) \quad (16)$$



Pictorial Illustration of UrdhwaTiryakbhyam Sutra

Graphically illustrates the step by step method of multiplying two 8 bit numbers using the UrdhwaTiryakbyam Sutra. The black circles indicate the bits of the multiplier and multiplicand, and the two-way arrows indicate the bits to be multiplied in order to arrive at the individual bits of the final product. The hardware architecture of the 8x8 Urdhwa multiplier has been designed and shown in Fig.

Example for UrdhwaTiryakbhyam Algorithm

Now let an example to explain the multiplication procedure of UrdhwaTiryakbhyam Sutra. The 3 by 3 decimal number is taken and the procedure is shown.

654 – Multiplier
321 – Multiplicand

1) The first step is like as shown in the figure. The multiplication should done between the first digits in each numbers from right hand side.

2) The second step is multiplying cross wise first digit in first number and second digit in second number and vice versa.

4×2=8
5×1=5

And now add 8 and 5 gives 13. In number 13 tens place number is the Carry.

3) The third step is multiplying first digit in first number with third digit in second number , second digit in both numbers and first digit in second number with third digit in first number

4×3=12
5×2=10
6×1=6

Here we got 2 digit number, so we have another carry which will propagate to next stage.

4) Next step is multiplying second digit in first number with third digit in second number and vice

5×3=15
6×2=12

Here we got 2 digit number, so we have another carry which will propagate to next stage.

5) Final step is multiplying last digits in both numbers with each other. 6×3=18

And now add this partial product and previous carry that results final partial product 18+2=20.

6) Now arrange partial products without carry from bottom to top in a sequence from left to right, we will get final product

Result = 209934

The procedure which we use here is used for binary numbers also. Now, let applying same procedure for two 8 bit numbers as shown below. Partial products can be denoted as P0 to P15.

P=10110110 – multiplier
Q=11011001– multiplicand

Now the partial products are from P14 to P0. The partial product has two parts carry and product. Right most digit in the partial product is the product and remaining part is the carry. To get final product write the each product from P14 to P0,this is the required final product.

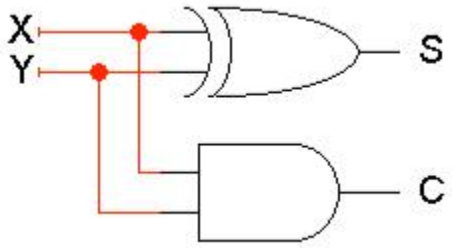
Final Result = 1 0 0 1 1 0 1 0 0 1 0 0 0 1 1 0

The number of multiplications required is same for the UrdhwaTiryakbhyam algorithm and normal multiplication but the advantage of the UrdhwaTiryakbhyam algorithm is while implementing the circuit the partial product come directly, and also there is no need of complex time consuming operations like shifting and rotating. This algorithm eliminates sequential operations so speed of operation increases. So UT algorithm is best suited for the hard ware implementation with less complexity.

Half Adder

Half adder is the basic unit of this multiplier circuit. The half adder circuit adds two bits and gives two outputs, sum and the carry. The sum is the result of the XOR gate and carry is the result of the AND gate. The circuit of the half adder is shown in below Fig 2.5.

Sum = A XOR B
Carry = A AND B



Half Adder circuit diagram

The truth table of the half adder is shown in below table

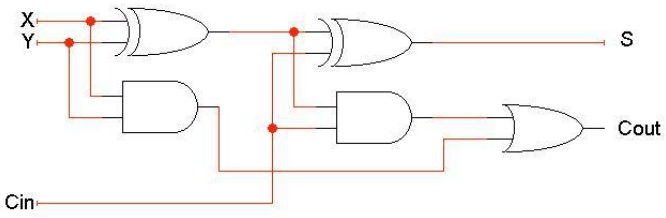
Half adder Truth table

X	Y	SUM	CARRY
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Full Adder

The full adder is the circuit which will add two bits besides carry, totally three bits, same as half adder it gives two outputs sum and carry. The full adder is combination of two half adders and one or gate. The circuit diagram is shown in Fig.

Sum = X XOR Y XOR C_{IN}
CARRY = (X XOR Y) C_{IN} + XY



Full Adder circuit diagram

Full adder Truth table

X	Y	CIN	SUM	CARRY
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

As mentioned earlier, the partial products obtained are added with the help of full adders and half adders. It can be seen, from equation (1) to (16) that in few equations there is a necessity of adding more than 3 bits at a time. This leads to additional hardware and additional stages, since the full adder is capable of adding only 3 bits at a time. Fig 2.7 shows Hardware architecture of UrdhvaTiryakbhyam multiplier with full adders and half adders, with reduced architecture and increased efficiency in terms of speed.

Vedic Multiplier

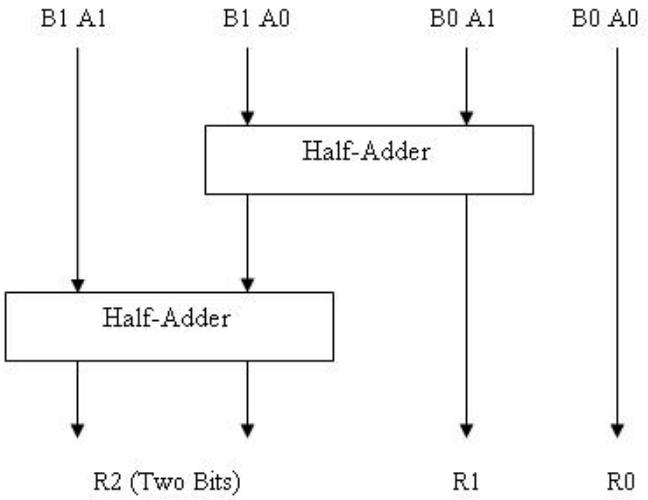
Vedic mathematics is the name given to the ancient Indian system of mathematics that was rediscovered in early twentieth century. This mathematics is mainly based on sixteen principles which are termed as Sutras. In other words, Vedic multiplier architecture based on the Urdhva Tiryakbhyam (Vertically and cross wise) Sutra is presented. This Sutra was traditionally used in ancient India for the multiplication of two decimal numbers in relatively less time. The hardware architecture of Vedic multiplier is to be very similar to that of popular array multiplier.

Vedic mathematics fulfils the explanation of various modern mathematical terms including arithmetic, geometry (plane, co-ordinate), trigonometry, quadratic equation, factorization and even calculus. The advantage of Vedic mathematics lies in the fact that it reduces the otherwise cumbersome-looking calculations in conventional mathematics to a very simple one. This multiplication is very interesting field and presents some effective algorithms which can be applied to various branches of engineering such as computing and digital signal processing.

Implementation of Multiplier using Vedic Multiplier

The hardware architecture of 2X2 Vedic multiplier is illustrated in Fig.
"Urdhva Tiryakbhyam" (Vertically and crosswise) sutra is used to propose such architecture for the multiplication of two binary numbers. The main benefit of Vedic multiplier is that partial product generation and additions are done concurrently. Hence, it is well adapted to parallel processing. The features of this makes more attractive for binary multiplication.

The 2X2 Vedic multiplier is implemented using four AND gates and two half-adders which is displayed in its block diagram. It is found that the hardware architecture of 2X2 Vedic multiplier is same as the hardware architecture of 2X2 array multiplier. Therefore multiplication of 2 bit number using Vedic method does not made significant effect in improvement of multiplier's efficiency. So we switch over to the implementation of 4X4 bit Vedic multiplier which uses the 2X2 bit Vedic multiplier as a basic building block.



2X2 Vedic Multiplier

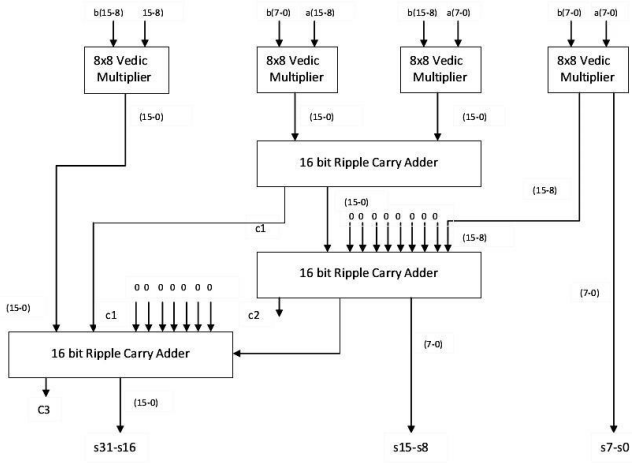
The same method can be extended for input bits 4 and 8. But for higher number of bits in input, little modification is required. The structure of 16 bit Vedic

multiplier by using 8X8 Vedic multiplier is illustrated in Fig 2.9. This structure consists of three 16 bit ripple carry adder for addition process.

The performance of 16-bit Vedic multiplier is increased in terms of area, delay and power when compared to performance of 16-bit array multiplier. We can easily design 32-bit and 64-bit Vedic multiplier with help of 16X16 and 32X32 bit Vedic multiplier. Hence from above consecution, it is clear that for large bit multiplication, Vedic multiplier gives more advantage than array multiplier.

However, there are some limitations in Vedic multiplier. In each NXN Vedic multiplier, three sets of ripple carry adders are essential for perform addition process of partial products. This multiplier's performance is better than array and modified booth multiplier's performance. Further to improve the performances of Vedic multiplier, different types of adders like Carry Save Adder (CSA), Carry Look-ahead Adder (CLA), Square Root Cary Select Adder (SQRT CSLA) is incorporated into Vedic multiplier. In order to reduce the limitation of Vedic multiplier, Wallace tree multiplier has been developed in later.

An adder is the main component of an arithmetic unit. Adders are commonly found in many building blocks of microprocessors and digital signal processing chip.



8X8 Vedic Multiplier

Adders are essential not only for addition, but also for subtraction, multiplication and division. Addition is one of the fundamental arithmetic operations. A fast and accurate of digital system is greatly influenced by the performance of the resident adders.

The most important for measuring the quality of adder design were propagation delay and area. Application where these adders are used are multipliers, Digital signal processing to execute like fast Fourier transform (FFT), finite impulse response (FIR) and infinite impulse response (IIR).

Carry Select Adder

Design of area and power efficient high speed data logic systems are one of the most substantial areas of research in VLSI system design. Addition usually impacts widely the overall performance of digital systems and an arithmetic function. In electronics applications adders are most widely used. In multipliers, DSP to execute various algorithms like FFT, FIR and IIR. In digital adders, the speed of addition is limited by the time required to propagate a carry through the adder. The sum for each bit position in an elementary adder is generated sequentially only after the previous bit position has been summed and a carry propagated into the next position.

The CSLA is used in many computational systems to alleviate the problem of carry propagation delay by independent generation multiple carries and then select a carry to generate the sum. However, the CSLA is not an area efficient because it uses multiple pairs of Ripple Carry Adders(RCA) to generate partial sum and carry by considering carry input as Cin=0 and Cin=1, then the final summation and carry are selected by the multiplexers.

The Carry select adders are classified as Linear Carry select adder and Square-root Carry select adder.

Linear Carry Select Adder

The linear carry select adder is constructed by chaining a number of equal length adder stages. For an n-bit adder, it could be implemented with equal length of carry select adder and is called as linear carry select adder.

Square-Root Carry Select Adder

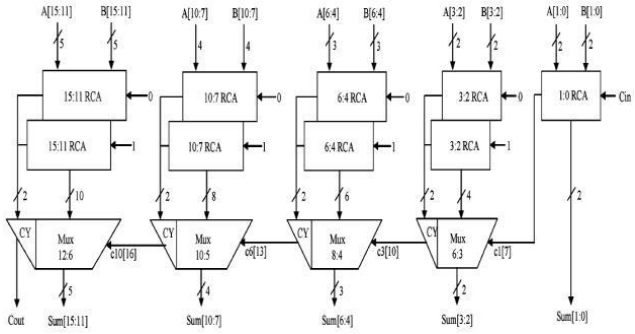
The square-Root carry select adder is constructed by equalizing the delay through two carry chains and the block multiplexer signal from previous stage. It is also called as non-linear carry select adder.

The existing modified SQRT CSLA uses Binary to Excess-1 Converter (BEC) instead of RCA with Cin=1 in the regular CSLA to achieve lower delay with slightly increase in area. The basic idea of the proposed architecture is that which replaces the BEC logic by Common Boolean Logic. The proposed architecture generates a duplicate sum and carry-out signal by using NOT and OR gate and select value with the

help of multiplexer. The multiplexer is used to select the correct output according to its previously carry-out signal.

Regular SQRT Carry Select Adder

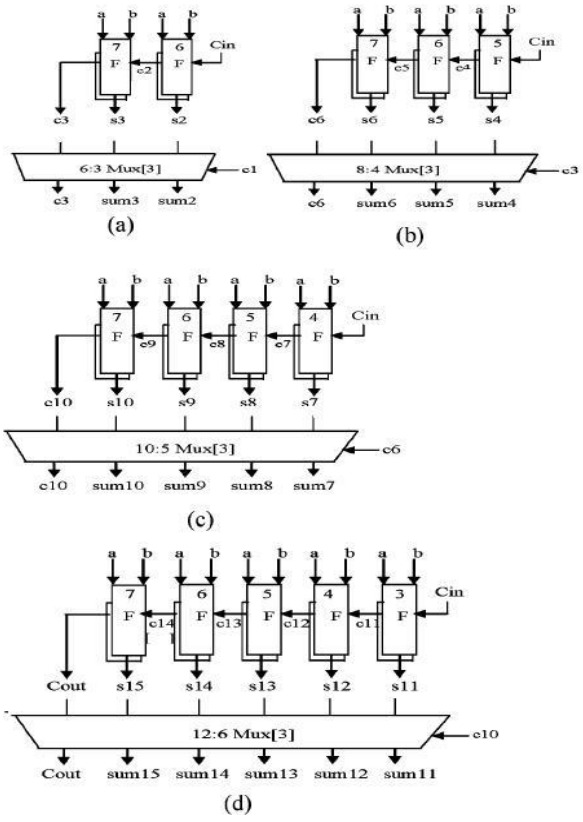
The basic square-Root Carry Select adder has a dual ripple carry adder with 2:1 multiplexer, the main disadvantage of regular CSLA is the large area due to the multiple pairs of ripple carry adder. The regular 16-bit SQRT Carry select adder is shown in Fig. These 16-bits are divided into five groups with different bit sizes of ripple carry adders.



Binary [3:0]	Excess-1 [3:0]
0000	0001
0001	0010
:	:
:	:
1110	1111
1111	0000

Regular 16-bit SQRT CSLA

From the structure of regular SQRT CSLA, there is scope for reducing delay and area utilization. The carry out is calculated from the last stage, in this the selection is done by using a multiplexer. The internal structures of the group2, group3, group4 and group5 of the regular 16-bit SQRT CSLA is shown in Fig.



Individual groups of regular 16-bit SQRT CSLA

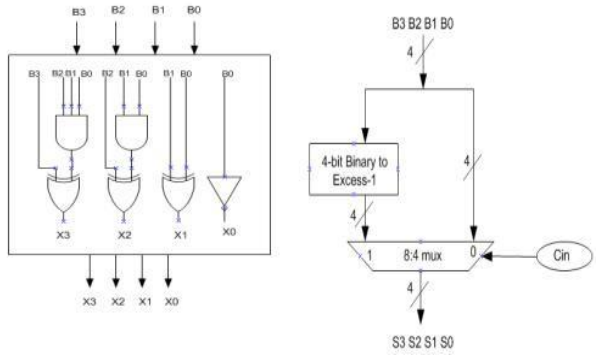
Modified SQRT Carry Select Adder

The main idea of this work is to use BEC instead of the RCA with Cin=1 in order to reduce the delay and area utilization of the regular SQRT CSLA. To replace the n-bit RCA, a n+1 bit BEC is required. The structure of a 4-bit BEC is shown in Fig. and the function table given in Table I.

Fig. illustrates how the basic function of the CSLA is obtained by using the 4-bit BEC together with the mux. In this structure one input of the 8:4 mux gets as it input (B3, B2, B1, and B0) and another input of the mux is the BEC output. This produces the two possible partial outputs in parallel according to the control signal Cin The importance of the BEC logic stems from the large silicon area reduction when the CSLA with large number of bits are designed. The Boolean expression of the 4-bit BEC is listed as below.

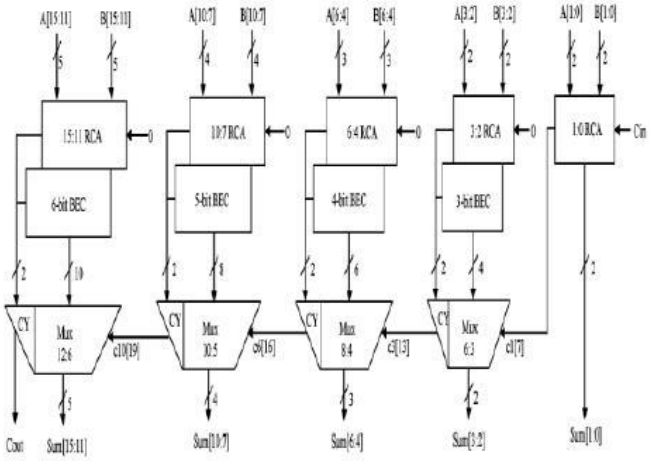
X0 = ~B0
X1 = B0^B1
X2 = B2 ^ (B0 & B1)
X3 = B3 ^ (B0 & B1 & B2).

Function table of the 4-bit BEC



Structure of 4-bit BEC with 8:4 MUX

The modified 16-bit SQRT CSLA using BEC is shown in Fig.. The structure is again divided into five groups with different sizes of Ripple carry adder and BEC. The group2, group3, group4 and group5 of 16-bit SQRT CSLA are shown in Fig.. The parallel Ripple carry adder with Cin=1 is replaced with BEC. One input to the multiplexer goes from the RCA with Cin=0 and other input from BEC. Comparing the individual groups of both regular and modified SQRT CSLA, it is clear that the BEC structure reduces delay. But the disadvantage of BEC method is that the area is increasing than the regular SQRT CSLA.



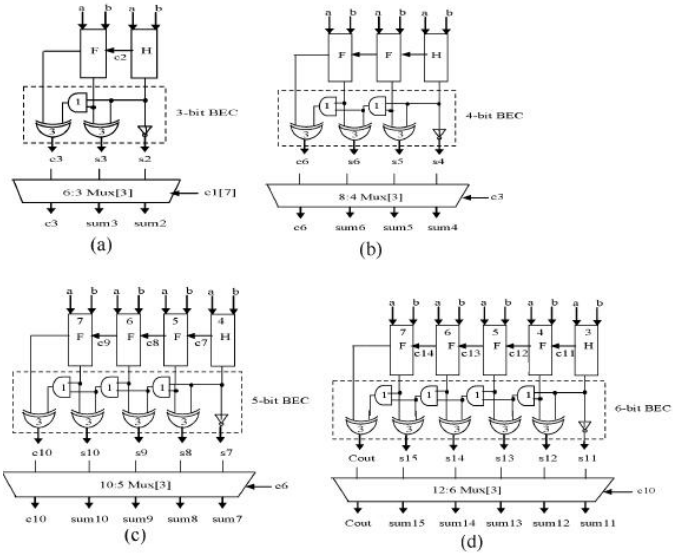
Modified 16-bit SQRT CSLA

Carry Select Adder (CSLA) is one of the fastest adders used in many data processing processors to perform fast arithmetic functions. It alleviates the problem of carry propagation delay by independently generating multiple carriers and then selects a carry to generate the sum. CSLA are used for high speed carry propagation delay. The basic operation of CSLA is parallel computation. The CSLA provide a compromise between small area but longer ripple carry adder (RCA) and larger area with shorter delay carry-look ahead adder.

The CSLA has two units:

1) The sum and carry generation unit (SCG).
2) The sum and carry selection unit (SCS).

The SCG unit consumes most of the logic resources of CSLA and significantly contributes to the critical path.



Individual groups of modified 16-bit SQRT CSLA

Applications

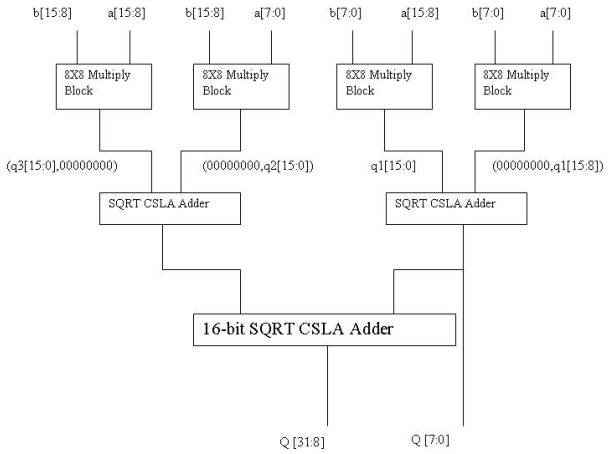
- Digital signal processors.
- Microprocessors.
- Multipliers.
- Digital circuits.

SYSTEM DESIGN

PROCESSING UNIT (MAC)

Multiplication and Accumulation (MAC) is one of the processing units in the neural network. Based on the performance of the MAC only, the accuracy of the network is obtained. MAC operation was performed, using the Vedic multiplier with SQRT-CSLA.

Vedic multiplier



Block Diagram of 16 X 16 Vedic Multiplier

Multiplication is one of the important arithmetic operations in signal processing applications. Signal processing involves multiplication, speed and accuracy is the main constraint in the multiplication process. Speed can be achieved by reducing the computation process in the multiplication technique. Vedic multiplier is efficient multiplication technique.

Fig. shows the architecture of 16-bit Vedic multiplier. It was designed by four 8 X 8 Vedic multiplier, each multiplier perform the operation separately. Partial products are added by 16-bit SQRT-CSLA; finally get a 32-bit multiplication output.

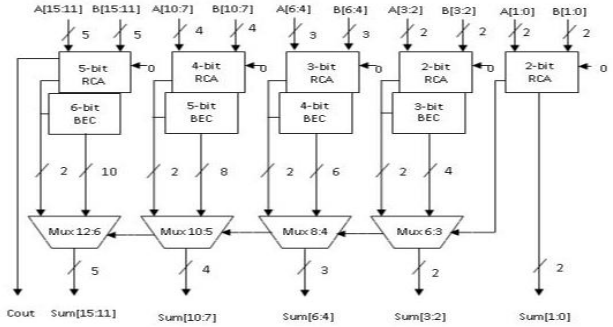
The efficient Vedic multiplication technique is used. The 16-bit Vedic multiplier is designed by using four 8X8 Vedic multiplier and square root carry select adder (SQRT-CSLA). The 16-bit input sequence is divided into two 4-bit numbers. Input to the 8-bit multiplier are a[7:0] & b[7:0], a[15:8] & b[7:0], a[7:0] & b[15:8], a[15:8] & b[15:8]. Intermediate partial products output are added using the three modified adder, named as SQRT-CSLA.

SQRT-CSLA Adder

Carry propagation delay and low complexity are recognized as high potential in every addition circuit. To achieve an efficient output, the proposed SQRT-CSLA structure has designed. SQRT-CSLA adder circuit is classified into two types based on selecting the carry inputs. a) Dual RCA based SQRT CSLA; b) BEC based SQRT CSLA.

In the dual RCA (Ripple Carry Adder) based SQRT CSLA circuit, each group has dual RCA pair for providing carry select signals. RCA circuit would be more disadvantageous due to the increasing propagation delay. To overcome the problem, Binary to Excess 1 converter circuit has been suggested in the SQRT-CSLA adder.

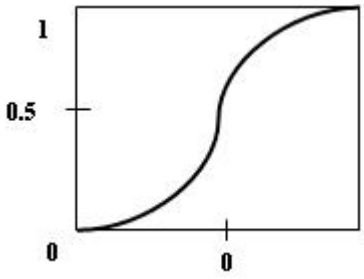
Fig. shows the Architecture of BEC Based SQRT CSLA, it contain BEC, RCA and mux. Half adders, full adders and multiplexers are used for providing partial product addition results. BEC circuits are used to provide same RCA functions, but have different architectures with less gate count.



Architecture of BEC based SQRT CSLA

Activation function unit

The Activation function of a single neuron in the artificial neural network is determined as function of the output in that neuron. Binary threshold neuron, Sigmoid neuron shown in Fig 3.3, and Rectified linear neuron are the different activation function used in the neural network. Output functions are varied by the activation function.



Sigmoid Neuron

$$y = \frac{1}{1 + e^{-z}}$$

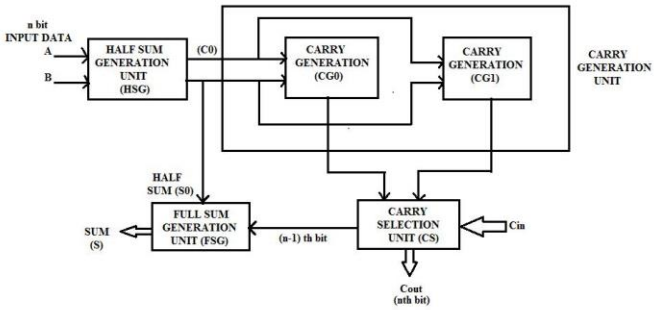
y is denotes the output function,**z** is denotes as impulse response function.

The above equation describes the function of the sigmoid neuron. Sigmoid neuron is used as the activation function of the neural network. The above function is efficient compared to other activation functions.

SYSTEM ARCHITECTURE

The overall system architecture of the proposed work is presented in Fig 3.4. It consists of 4 different modules namely half sum generation, carry generation, carry selection and full sum generation.

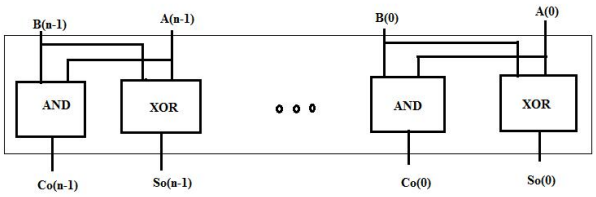
All the redundant logic operation present in the conventional CSLA is eliminated and a new logic formulation is used in the proposed CSLA design.



System Architecture of Reduced Area-Delay-Power (ADP) - SQRT-CSLA

Half Sum Generation Unit

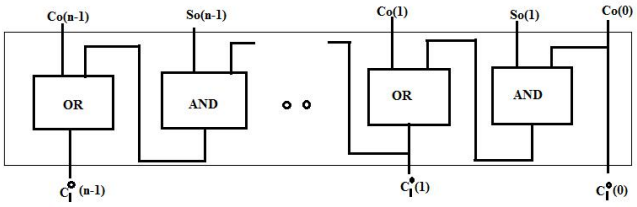
The Half Sum Generation units (HSG) receives the two n-bit operands (A and B) and generates the half sum word (S0) and half carry word (C0) of width n-bit each. The logic diagram of the HSG unit is shown in Fig 3.5. The Carry Generation unit receives both the half sum and half carry words from the HSG unit.



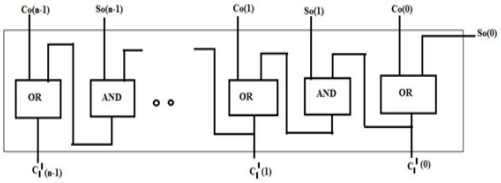
Half Sum generation unit

Carry Generation Unit

The Carry Generation (CG) unit composed of two CGs (CG0 and CG1) corresponding to input carries 0 and 1. Both CG0 and CG1 receives half sum word (S0) and half carry word (C0) from the HSG unit and generate two n-bit full carry word C_1^0 and C_1^1 corresponding to the input carry 0 and 1 respectively. The logic circuits of CG0 and CG1 are optimized to take advantage of the fixed input carry bit. The optimized design of CG0 and CG1 are shown in Fig.



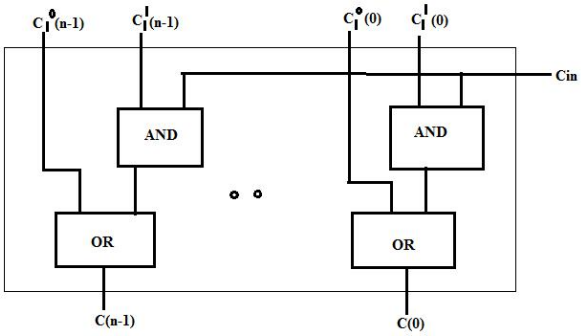
Carry Generation Unit for Input Carry = 0



Carry Generation Unit for Input Carry = 1

Carry Selection Unit

The Carry Selection (CS) unit selects one final carry word from the two carry words available at the input line using the control signal $C_{in.}$ It selects C10 when Cin = 0; otherwise it selects C_1^1. The CS unit can be implemented using an n-bit 2-to-1 MUX. However, the carry word C_1^0 and C_1^1 follow a specific bit pattern. If $C_1^0 = 1$ then $C_1^1(i) = 1$ irrespective of S0(i) and C0(i), for $0 \leq i \leq n-1$. This feature is used for logic optimization of Carry Selection unit. The optimized design of the CS unit is shown in Fig which composed of n AND- OR gates.

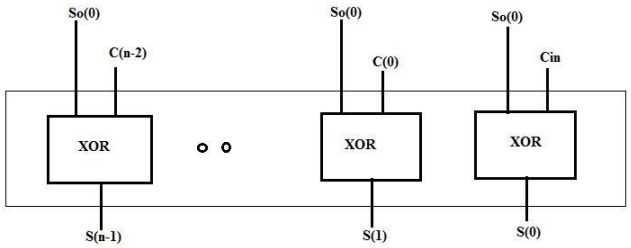


Carry Selection Unit

Full Sum Generation Unit

The final carry word C is obtained from the carry Selection unit. The most significant bit (MSB) is sent to output as Cout, and (n-1) least significant bits are XORed with (n-1) MSBs of half sum (S0) in the full sum generation unit to obtain (n-1) MSBs of final sum (S). The LSB of S0 is XORed

with Cin to obtain the LSB of S. The logic diagram of Full Sum Generation (FSG) unit is shown in Fig.



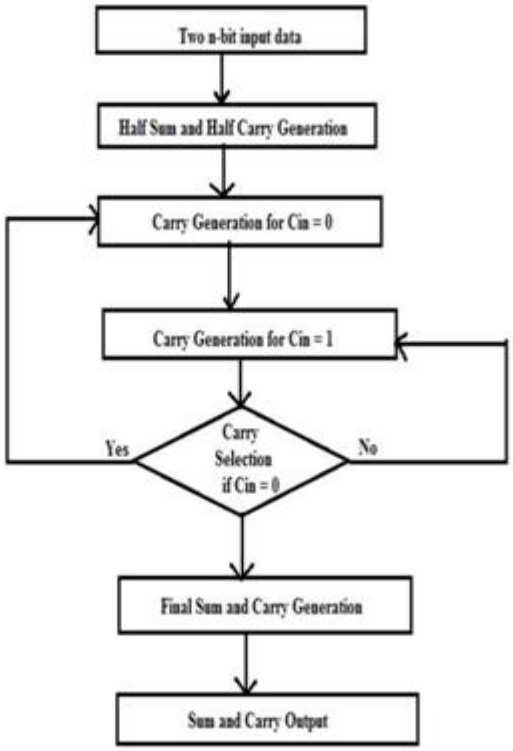
Full Sum Generation Unit

FLOW DIAGRAM

The flow chart shown in Fig describes the generation of sum and carries word.

The HSG receives two n-bit input data and generate the half-sum word and half-carry word of width of n-bit each. Both CG0 and CG1 receives half sum and half carry word from the HSG unit and generate the two n-bit full-carry word corresponding to input carry 0 and 1 respectively. The CS unit selects one final carry word from the two carry word available at its input line using the control signal Cin.

The final carry word is obtained from the CS unit. The final sum is generated by the FSG unit.



Flow Chart of Reduced ADP- SQRT – CSLA

III. REQUIREMENT SPECIFICATIONS

SOFTWARE REQUIREMENTS

- Synthesis Tool

-Xilinx ISE 10.2

- Verification Tool

-Modelsim 6.3c

XILINX ISE

The Integrated Software Environment (ISE) is the Xilinx design software suite that allows us to take our design from design entry through Xilinx device programming. The ISE Project Navigator manages and processes our design through the following steps in the ISE design flow.

DESIGN ENTRY

Design entry is the first step in the ISE design flow. During design entry, we create our source files based on our design objectives. We can create our top-level design file using a Hardware Description Language (HDL), such as VHDL, Verilog, or ABEL, or using a schematic. We can use multiple formats for the lower-level source files in our design.

SYNTHESIS

After design entry and optional simulation, we run synthesis. During this step, VHDL, Verilog, or mixed language designs become netlist files that are accepted as input to the implementation step.

IMPLEMENTATION

After synthesis, we run design implementation, which converts the logical design into a physical file format that can be downloaded to the selected target device. From Project Navigator, we can run the implementation process in one step, or we can run each of the implementation processes separately.

Implementation processes vary depending on whether we are targeting a Field Programmable Gate Array (FPGA) or a Complex Programmable Logic Device (CPLD).

VERIFICATION

We can verify the functionality of our design at several points in the design flow. We can use simulator

software to verify the functionality and timing of our design or a portion of our design.

The simulator interprets VHDL or Verilog code into circuit functionality and displays logical results of the described HDL to determine correct circuit operation. Simulation allows us to create and verify complex functions in a relatively small amount of time. We can also run in-circuit verification after programming your device.

DEVICE CONFIGURATION

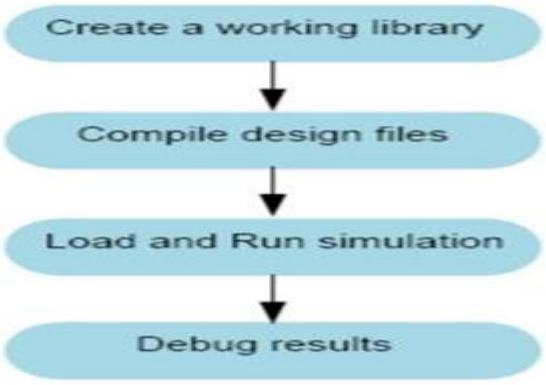
After generating a programming file, we configure our device. During configuration, we generate configuration files and download the programming files from a host computer to a Xilinx device.

MODELSIM 6.3C

ModelSim is a verification and simulation tool for VHDL, Verilog, System Verilog, and mixed language designs.

BASIC SIMULATION FLOW

Basic simulation flow is shown in Fig.The simulation flow comprises of creating a working library, in which compilation of designed files happen, which is further loaded and simulated and finally the output is debugged.



Basic Simulation Flow

CREATING A WORKING LIBRARY

In ModelSim, all designs are compiled into a library. We typically start a new simulation in ModelSim by creating a working library called "work," which is the default library name used by the compiler as the default destination for compiled design units.

COMPILING DESIGN

After creating the working library, we compile our design units into it. The ModelSim library format is compatible across all supported platforms. We can simulate our design on any platform without having to recompile our design.

LOAD AND RUN SIMULATION

With the design compiled, we can load the simulator with our design by invoking the simulator on a top-level module (Verilog) or a configuration or entity/architecture pair (VHDL).

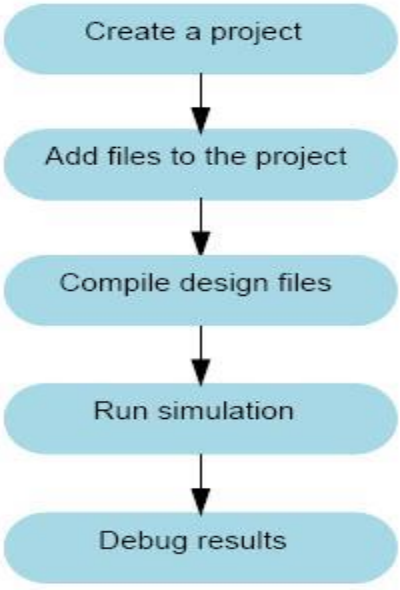
DEBUGGING

If we don't get the results we expect, we can use ModelSim's robust debugging environment to track down the cause of the problem.

PROJECT FLOW

MODELSIM PROJECT

A project is a collection mechanism for an HDL design under specification or test. Even though we don't have to use projects in ModelSim, they may ease interaction with the tool and are useful for organizing files and specifying simulation settings. The Fig shows the basic steps for simulating a design within a ModelSim project.



Project Flow

MULTIPLE LIBRARY FLOW

ModelSim uses libraries in two ways:

1) As a local working library that contains the compiled version of our design.
2) As a resource library.

The contents of our working library will change as we update our design and recompile. A resource library is typically static and serves as a parts source for our design. We can create our own resource libraries, or they may be supplied by another design team or a third party (e.g., a silicon vendor).

We specify which resource libraries will be used when the design is compiled, and there are rules to specify in which order they are searched. A common example of using both a working library and a resource library is one where your gate level design and test bench are compiled into the working library, and the design references gate-level models in a separate resource library.

DEBUGGING TOOLS

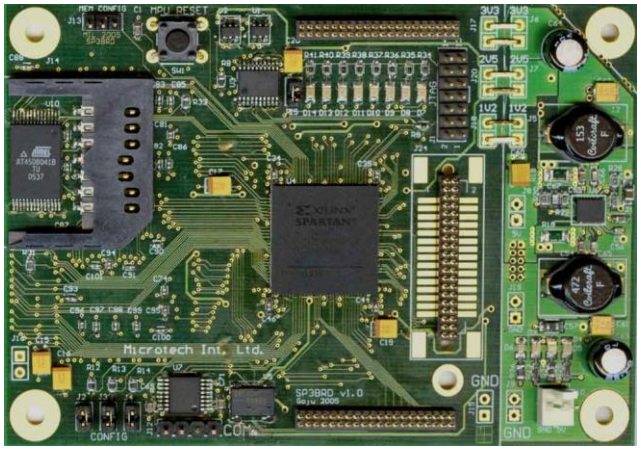
ModelSim offers numerous tools for debugging and analyzing your design. Several of these tools are covered in subsequent lessons, including:

- Using projects
- Working with multiple libraries
- Setting breakpoints and stepping through the source code
- Viewing waveforms and measuring time
- Viewing and initializing memories
- Creating stimulus with the Waveform Editor
- Automating simulation

IV. HARDWARE TOOLS

SPARTAN 3

The family of Field-Programmable Gate Arrays is specifically designed to meet the needs of high volume, cost-sensitive consumer electronic applications. The eight-member family offers densities ranging from 50,000 to five million system gates.



Spartan-3

The Spartan-3family shown in Fig builds on the success of the earlier Spartan-IIE family by increasing the amount of logic resources, the capacity of internal RAM, the total number of I/Os, and the overall level of performance as well as by improving clock management functions. Numerous enhancements derive from the Vertex®-II platform technology.

These Spartan-3 FPGA enhancements, combined with advanced process technology, deliver more functionality and bandwidth per dollar than was previously possible, setting new standards in the programmable logic industry. Because of their exceptionally low cost, Spartan-3 FPGAs are ideally suited to a wide range of consumer electronics applications, including broadband access, home networking, display/projection and digital television equipment. The Spartan-3 family is a superior alternative to mask programmed ASICs. FPGAs avoid the high initial cost, the lengthy development cycles, and the inherent inflexibility of conventional ASICs. Also, FPGA programmability permits design upgrades in the field with no hardware replacement necessary, an impossibility with ASICs.

FEATURES

- Low-cost, high-performance logic solution for high-volume, consumer-oriented applications
- Automotive Spartan-3 XA Family variant
- Select IO™ interface signaling
- Double Data Rate (DDR) support
- DDR, DDR2 SDRAM support up to 333 Mbps
- Abundant logic cells with shift register capability
- Wide, fast multiplexers
- Fast look-ahead carry logic
- Dedicated 18 x 18 multipliers
- Select RAM hierarchical memory
- Up to 1,872 Kbits of total block RAM

- Up to 520 Kbits of total distributed RAM
- Digital Clock Manager (up to four DCMs)
- Fully supported by Xilinx ISE and Web PACK

ARCHITECTURAL OVERVIEW

The Spartan-3 family architecture consists of five fundamental programmable functional elements:

- Configurable Logic Blocks (CLBs) contain RAM-based Look-Up Tables (LUTs) to implement logic and storage elements that can be used as flip-flops or latches. CLBs can be programmed to perform a wide variety of logical functions as well as to store data.
- Input/Output Blocks (IOBs) control the flow of data between the I/O pins and the internal logic of the device. Each IOB supports bidirectional data flow plus 3-state operation. Twenty-six different signal standards, including eight high-performance differential standards. Double Data-Rate(DDR) registers are included. The Digitally Controlled Impedance (DCI) feature provides automatic on-chip terminations, simplifying board designs.
- Block RAM provides data storage in the form of 18-Kbit dual-port blocks.
- Multiplier blocks accept two 18-bit binary numbers as inputs and calculate the product.
- Digital Clock Manager (DCM) blocks provide self-calibrating, fully digital solutions for distributing, delaying, multiplying, dividing, and phase shifting clock signals.

These elements are organized in a ring of IOBs surrounds a regular array of CLBs. The XC3S50 has a single column of block RAM embedded in the array. Those devices ranging from the XC3S200 to the XC3S2000 have two columns of block RAM. The XC3S4000 and XC3S5000 devices have four RAM columns.

Each column is made up of several 18-Kbit RAM blocks; each block is associated with a dedicated multiplier. The DCMs are positioned at the ends of the outer block RAM columns.

The Spartan-3 family features a rich network of traces and switches that interconnect all five functional elements, transmitting signals among them. Each functional element has an associated switch matrix that permits multiple connections to the routing.

CONFIGURATION

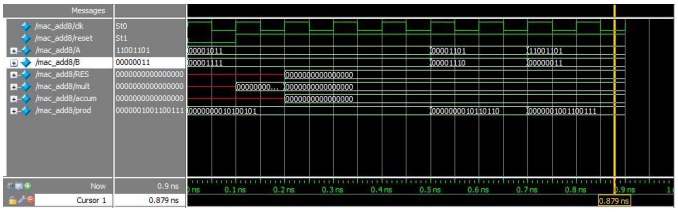
Spartan-3 FPGAs are programmed by loading configuration data into robust, reprogrammable, static CMOS configuration latches (CCLs) that collectively control all functional elements and routing resources. Before powering on the FPGA, configuration data is stored externally in a PROM or some other nonvolatile medium either on or off the board. After applying power, the configuration data is written to the FPGA using any of five different modes: Master Parallel, Slave Parallel, Master Serial, Slave Serial, and Boundary Scan (JTAG). The Master and Slave Parallel modes use an 8-bit-wide Select MAP port. The recommended memory for storing the configuration data is the low-cost Xilinx Platform Flash PROM family, which includes the XCF00S PROMs for serial configuration and the higher density XCF00P PROMs for parallel or serial configuration.

V. RESULT AND COMPARISON

FPGA kit

The Fig is Spartan-3 FPGA.Spartan-3 FPGAs are programmed by loading configuration data into robust, reprogrammable, static CMOS configuration latches (CCLs) that collectively control all functional elements and routing resources. Before powering on the FPGA, configuration data is stored externally in a PROM or some other nonvolatile medium either on or off the board. After applying power, the configuration data is written to the FPGA using any of five different modes: Master Parallel, Slave Parallel, Master Serial, Slave Serial, and Boundary Scan (JTAG). The Master and Slave Parallel modes use an 8-bit-wide Select MAP port. The recommended memory for storing the configuration data is the low-cost Xilinx Platform Flash PROM family, which includes the XCF00S PROMs for serial configuration and the higher density XCF00P PROMs for parallel or serial configuration

RESULT OF VEDIC MULTIPLIER USING MODELSIM



MAC Unit Output

Fig . shows the output form of the MAC unit. Follow the below procedure to generate output wave form.

- Click on simulate button on top of ModelSim
- Work window will be popped up, select the Mac_add8 from work window and click OK button.
- Right click on Mac_add8 and select Add all signals to wave.
- Wave form window is generated which is shown in fig.
- Right click on clk and set it to clock.
- Right click on Reset and force reset value to 0.
- Right click on mac_add8/A and force value to 11001101 which is 205 in unsigned value.
- Right click on mac_add8/B and force value to 00000011 which is 3 in unsigned value.
- Click on Run button which is on top of the wave window.
- Output will be generated by multiplying 205 and 3 and comes output as 615.
- Right click on Reset and force it to 1.
- Click on Run button continuously.
- Output will be generated by adding 205 to each cumulative output and will display in Accumulator line.

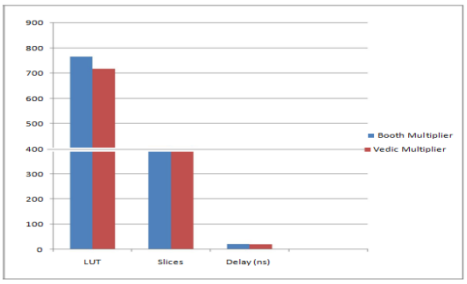
Results of various multipliers

Parameters	Existing Booth Multiplier	Proposed Vedic Multiplier
LUT	764	717
Slices	402	397
Delay (ns)	19.114	19.1

From the above table it is clear that the Area, Delay and Power is reduced respectively for each Multipliers. It is clearly showing that numbers of LUTs, slices, Delay are reduced from existing booth multiplier and proposed vedic multiplier.

USING GRAPH

Fig shows the graphical representation of MAC output using Booth multiplier and Vedic multiplier where it is clearly showing the difference in LUTs, number of slices and delay between Booth and Vedic multipliers.



Graphical Representation of MAC Unit Output

VI. CONCLUSION AND FUTURE SCOPE

CONCLUSION

Artificial Neural Networks are used in many applications, to analyze the methodology. MAC unit is one of the processing units in the artificial neural network. MAC decides the output function is efficient or not. So, a new MAC unit with the help of Vedic multiplier with SQRT-CSLA is designed. It produced the accurate and efficient output, compared to the existing booth multiplier with carry look ahead adder. Our proposed MAC increases the speed of the neural network. The MAC operation is performed well, entire network performance also increased.

In future the multiplier circuit is designed by using Reversible logic gates. It consumes less power compared to our ordinary logic gates. So this technique is applied to the neural network, get a better results.

FUTURE SCOPE

- Advanced Carry Select Adder can be replaced with Fast adder for reducing delay
- Input sample can be given above and below the Threshold value.

Threshold value and M sample value can be changed for future scope.

REFERENCES

[1] Saman Razavi and Bryan A. Tolson, "A New formulation for feedforward neural

networks" IEEE Transactions on neural networks, vol.22, October 2011.

[2] Richard L. et al. "Comparison of feedforward and feedback neural networkarchitectures for short term wind speed prediction", International joint conference on neural networks, June 2009.

[3] Premananda B.S. et al. "Design and Implementation of 8-bit Vedic multiplier",International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering, vol. 2, Issue 12, December 2013.

[4] Damarla paradhasaradhi and K. Anusudha,"An area efficient SQRT carry select adder", International Journal of Engineering Research and Applications, vol. 3, Issue 6, Dec 2013.

[5] Hariprasath S and T.N. Prabakar," FPGA Implementation of multilayer feed forward neural network architecture using VHDL".S. Coric, I.Latinovic and A.Pavasovic," A Neural Network FPGA Implementation" ,IEEE, Neural-2000.

[6] G.Ganesh Kumar, V. Charishma,"Design of High speed Vedic multiplier using Vedic mathematics Techniques", International Journal of scientific andResearch publication, volume 2, Issue 3, march 2012.

[7] K.Saranya,"Low power and area efficient carry select adder", International Journal of soft computing and Engineering, volume-2, Issue-6, January 2013.

[8] B. Ramkumar and Harish M kittur,"Low power and area efficient carry select adder", IEEE Transaction on Very large scale Integration (VLSI) systems, vol. 20.