Abstract - At present era energy consumption in every portable and embedded device is main issue. As we know processor is part of portable and embedded device and multiplier unit is important part of processor. So this paper presents the FPGA comparison, delay and frequency analysis of proposed approximate MAC Unit. The MAC unit is a 8x8-bit Urdadhav multiplier with a 16-bit accumulator. The paper’s main focus was to analyze fast adder algorithms and multiplication schemes, and utilizes them in the design and implementation of a MAC unit. Major important issue in digital circuits besides speed, area, power consumption is accuracy. In this work, our main focus is on performance and accuracy, but we do provide some numbers for the arithmetic units relating to energy and power. This is to provide an estimate of the amount of energy and power consumed by the units we choose to implement. The priorities of this paper in order of importance, are, Robust and safe circuits, Design time, Area/speed balance. Using Vedic mathematics and approximation a new multiplier technique is proposed which reduces the delay and hardware complexity. Here a new approximate Multiplier technique is proposed whose the lsb four bits are approximated. The proposed accurate and Accurate multiplier and adder and Multiply and accumulate unit are designed on Xilinx FPGA Virtex 6 device and analyzed the results. The proposed MAC Unit provides an accuracy of . The overall area and Delay and Frequency analysis are presented and compared. From the results we can depict that up to 35% of reduction at all levels are achieved.

Keywords - Vedic Mathematics, Urdhva Tiryakbhyam, Multiplication, Approximation, Ripple Carry Adder, Carry-Look-Ahead Adder, Verilog HDL Simulation, MAC Unit

I. INTRODUCTION

Digital Signal Processing (DSP) is finding its way into more applications, and its popularity has materialized into a number of commercial processors]. Digital signal processors have different architectures and features than general purpose processors, and the performance gains of these features largely determine the performance of the whole processor. Widely used DSP algorithms include the Finite Impulse Response (FIR) filter, Infinite Impulse Response (IIR) filter, and Fast Fourier Transform (FFT). Efficient computation of these algorithms is a direct result of the efficient design of the underlying hardware.

One of the most important hardware structures in a DSP processor is the Multiply Accumulate (MAC) unit. This unit can calculate the running sum of products, which is at the heart of algorithms such as the FIR [5] and FFT [3]. The ability to compute with a fast MAC unit is essential to achieve high performance in many DSP algorithms, and is why there is at least one dedicated MAC unit in all of the modern commercial DSP processors.

The conventional high level model of the MAC unit after synthesis is shown in figure 1.1. The multiplier consists of a partial product multiplier that generates the result in carry-save format and a final carry-propagate adder, as the converter between the two different number representations. The final adder in figure 1.1 accumulates the new product to the sum of the previous clock cycle.

The Multipliers have an important effect in designing arithmetic, signal and image processors. Many mandatory functions in such processors make use of multipliers (for example, the basic building blocks in Fast Fourier transforms (FFTs) and multiply accumulate (MAC) are multipliers). The advanced digital processors now have fast bit-parallel multipliers embedded in them. Various methods exist for the reduction in the computation time involved by the multiplier with other factors as trade-offs. High-speed, bit-parallel multiplication can be classified into three types

(a) shift-and-add multipliers that generate partial products sequentially and accumulate. This requires more hardware and is the slowest multiplier. This is basically the array multiplier making use of the classical multiplying technique which consumes more time to perform two subtasks, addition and shifting of the bits and hence consumes 2 to 8 cycles of clock period.
(b) Generating all the partial product bits in parallel and accumulate them using a multi-operand adder. This is also called as parallel multiplier by using the techniques of Wallace tree [4] and Booth algorithm [1]

c) Using arrays of almost identical cells for generation of bit products and accumulation

This work proposes a Approximate multiplier providing the solution of the aforesaid problems adopting the sutra of Vedic Mathematics called Urdhva Tiryakbhyam (Vertically and Cross wise)[1,6,7]. It can be shown that the design MAC unit is highly efficient in terms silicon area/speed.

II. ADDER ALGORITHMS AND IMPLEMENTATIONS

A. Basic Adder Block

1) Half Adder

The Half Adder (HA) is the most basic adder. It takes in two bits of the same weight, and creates a sum and a carryout. Table 2.1a shows the truth table for this adder. If the two inputs a and b have a weight of 2^i (where i is an integer), sum has a weight of 2^i, and carryout has a weight of 2(i+1).

\[ \text{sum} = a \oplus b \]
\[ \text{carryout} = a \land b \]

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 2.1a Truth table for a Half Adder

\[ \text{Table 2.2a Extended Truth Table for a 1-bit adder} \]

2) Full Adder

The Full Adder (FA) is useful for additions that have multiple bits in each of its operands. It takes in three inputs and creates two outputs, a sum and a carryout. The inputs have the same weight, 2^i, the sum output has a weight of 2^i, and the carryout output has a weight of 2(i+1). The truth table for the FA is shown in Table 2.2a. The FA differs from the HA in that it has a carrying as one of its inputs, allowing for the cascading of this structure which is explored possible implementation using logic gates to realize the full adder.

\[ \text{sum} = a_i \oplus b_i \oplus c_i \]
\[ \text{carryout}_{i+1} = a_i \land b_i \land c_i + a_i \land b_i \land c_i \land a_{i+1} \land c_i \land d \land b_{i+1} \]

\[ \text{Table 2.2a Truth table for a Full Adder} \]

3) Partial Full Adder

The Partial Full Adder (PFA) is a structure that implements intermediate signals that can be used in the calculation of the carry bit. Revisiting the truth table for a FA (Table 2.2a), we extend it to include the signals generate (g), delete (d), and propagate (p). When g=1, it means carryout will be 1 (generated) regardless of carryin. When d=1, it means carryout will be 0 (deleted) regardless of carryin. When p=1, it means carryout will equal carryin (carryin will be propagated).

\[ \text{sum} = a_i \oplus b_i \oplus c_i \]
\[ \text{carryout}_{i+1} = a_i \land b_i \land c_i + a_i \land b_i \land c_i \land a_{i+1} \land c_i \land d \land b_{i+1} \land p \]

\[ \text{Table 2.2b Extended Truth Table for a Partial Full Adder} \]

B. Adder Algorithm

a. Ripple Carry Adder[9]

The Ripple Carry Adder (RCA) is one of the simplest adders to implement. This adder takes in two N-bit inputs (where N is a positive integer) and produces (N + 1) output bits (an N-bit sum and a 1-bit
The RCA is built from N full adders cascaded together, with the carryout bit of one FA tied to the carryin bit of the next FA. Figure 3 shows the schematic for an N-bit RCA. The input operands worst case (when all the carry outs are 1), this carry bit needs to ripple across the structure from the least. Hence, the time for this implementation of the adder the delay for the carryout of a FA and tRCAsum is the delay for the sum of a FA.

\[
\text{Propagation Delay} (t_{\text{RCAprop}}) = (N - 1) \cdot t_{\text{RCAcarry}} + t_{\text{RCAsum}}
\]

b. Carry Look Ahead Adder[8]

The CLA adder uses partial full adders to calculate the generate and propagate signals needed for the carryout equations. Figure 9 shows the schematic for a 4-bit CLA adder. For a 4-bit CLA adder the 4th carryout signal can also be considered as the 5th sum bit. Although it is impractical to have a single level of carry look ahead logic for long adders, this can be solved by adding another level of carry look ahead logic. To achieve this, each adder block requires two additional signals: a group generate and a group propagate.

\[
c_1 = g_0 + p_0 . c_0 \\
c_2 = g_1 + p_1 + c_1 = g_1 + p_1 + g_0 + p_1 + p_0 + c_0 \\
c_3 = g_2 + p_2 + c_2 = g_2 + p_2 + g_1 + p_2 . p_1 . g_0 + p_2 . p_1 . p_0 . c_0 \\
c_4 = g_3 + p_3 . c_3 = g_3 + p_3 . g_2 + p_3 . p_2 . g_1 + p_3 . p_2 . p_1 . g_0 + p_3 . p_2 . p_1 . p_0 . c_0 \\
group generate = g_3 + p_3 . g_2 + p_3 . p_2 . g_1 + p_3 . p_2 . p_1 . c_3 \\
group propagate = p_0 . p_1 . p_2 . p_3
\]

III. MULTIPLICATION SCHEMES

Multiplication hardware often consumes much time and area compared to other arithmetic operations. Digital signal processors use a multiplier/MAC unit as a basic building block [10] and the algorithms they run are often multiply-intensive. In this chapter, we discuss different architectures for multiplication and the methods that improve speed and/or area. Also, it is important to consider these methods in the context of VLSI design. It is beneficial to find structures that are modular and easy to layout. Many of the architectures described in this chapter will be used in the implementation of the multiply-accumulate unit for AsAP [11].

- **Multiplication Definition**
  To perform an M-bit by N-bit multiplication as shown in Figure 1, the M-bit multiplicand \( A = a(M-1)a(M-2)\ldots a1a0 \) is multiplied by the N-bit multiplier \( B = b(N-1)b(N-2)\ldots b1b0 \) to produce the M+N-bit product \( P = p(M+N-1)p(M+N-2)\ldots p1p0 \).
  A multiplication operation can be broken down into two steps:
  1) Generate the partial products.
  2) Accumulate (add) the partial products.
3.1) Array Multiplier
Each multiplicand is multiplied by a bit in the multiplier, generating N partial products. Each of these partial products is either the multiplicand shifted by some amount, or 0. This is illustrated in Figure 3 for an M * N multiplies operation. This figure can map directly into hardware and is called the array multiplier. The partial products are added in ripple fashion with half and full adders. A full adder's inputs require the carryin from the adjacent full adder in its row and the sum from a full adder in the above row. Abdelgawad [10] states that finding the critical path in this structure is non-trivial, but once identified, results in multiple critical paths. The delay basically comes down to a ripple delay through a row, and then down a column, so it is linearly proportional (td *(M +N)) to the sum of the sizes of the input operands.

3.2) Vedic Multiplication Algorithms
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 Jagadguru Shankaracharya Bharat Krishna Teerthaji Maharaja (1884-1960) comprised all this work together and gave its mathematical explanation while discussing it for various applications. Swami ji constructed 16 sutras (formulae) and 16 Upa sutras (sub formulae) after extensive research in Atharva Veda. Obviously these formulae are not to be found in present text of Atharva Veda because these formulae were constructed by Swami ji himself. Vedic mathematics is not only a mathematical wonder but also it is logical. Especially, methods of basic arithmetic are extremely simple and powerful [2, 7].

The word „Vedic” is derived from the word „Veda” which means the store-house of all knowledge. 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 [6,7]. The multiplier architecture can be generally classified into three categories. First is the serial multiplier which emphasizes on hardware and
minimum amount of chip area. Second is parallel multiplier (array and tree) which carries out high speed mathematical operations. But the drawback is the relatively larger chip area consumption. Third is serial-parallel multiplier which serves as a good trade-off between the times consuming serial multiplier and the area consuming parallel multipliers.

- Urdhva-Tiryakbhyam (Vertically and Crosswise)

Urdhva-Tiryakbhyam Sutra is a general multiplication formula applicable to all cases of multiplication. It literally means “Vertically and crosswise”. The conventional methods already known to us will require 16 multiplications and 15 additions. An alternative method of multiplication using the numbers to be multiplied are written on two consecutive sides of the square as shown in the figure. The square is divided into rows and columns where each row/column corresponds to one of the digits of either a multiplier or a multiplicand. Thus, each digit of the multiplier has a small box common to a digit of the multiplicand.

These small boxes are partitioned into two halves by the crosswise lines. Each digit of the multiplier is then independently multiplied with every digit of the multiplicand and the two-digit product is written in the common box. All the digits lying on a crosswise dotted line are added to the previous carry. The least significant digit of the obtained number acts as the result digit and the rest as the carry for the next step. Carry for the first step (i.e., the dotted line on the extreme right side) is taken to be zero.

a) Algorithm for 4 x 4 bit Vedic multiplier Using Urdhva Tiryakbhyam (Vertically and crosswise) for two Binary numbers

b) Algorithm for 8 X 8 Bit Multiplication Using Urdhva - Tiryakbhyam (Vertically and crosswise) for two Binary numbers

The design starts first with Multiplier design that is 2x2 bit multiplier as shown in figure 2. Here, “Urdhva Tiryakbhyam Sutra” or “Vertically and Crosswise Algorithm” for multiplication has been effectively used to develop digital multiplier architecture.

This algorithm is quite different from the traditional method of multiplication that is to add and shift the partial products.

Urdhva-Tiryakbhyam [1] (Vertically and crosswise) deals with the Multiplication of numbers. The sutra has been traditionally used for the Multiplication of decimal number [7]. These are having several steps to be followed:- let us assume 2 digits no, 46 and 55.

- Multiply the unit place digit, store its value in unit place while carry generated is forwarded to next no.(place value).
- Now perform the Cross Multiplication of Unit and Ten’s place value and store the result in addition with previous carry and multiplication value. If carry generated forward it to next place value.
- Now next perform the multiplication to the tens place digit and then add the result with forwarded carry.
IV. APPROXIMATION

Typically, embedded computing systems are required to achieve a required level of computing performance, with simultaneous and severe constraints on their characteristic such as power consumption, mobility and size. Moore’s law and the associated shrinking of transistor sizes, increase in mobility, decrease in size and power consumption has served as a driver for the proliferation and ubiquity of embedded systems. It is desirable for this trend to continue, to enable new applications and novel contexts in which embedded systems could be used. However, our ability to miniaturize silicon-based transistors is under serious jeopardy. These challenges can broadly be classified under two categories (i) the change in the nature of materials and material properties as the sizes of the transistors decrease and (ii) our inability to fabricate identical and reliable nanometer-sized silicon devices and achieve uniform behavioral characteristics. These challenges affect the physical characteristics of transistors and hence computing platforms in many ways [15]. “Relaxing the requirement of 100% correctness for devices and interconnects may dramatically reduce costs of manufacturing, verification, and test. Such a paradigm shift is likely forced in any case by technology scaling, which leads to more transient and permanent failures of signals, logic values, devices, and interconnects.”. This nonuniform, probabilistic and unreliable behavior of transistors has an impact on the desirable characteristics of embedded systems. A comprehensive survey of such challenges to nanometer-sized devices and beyond may be found in. Several approaches have been adopted to address these challenges to Moore’s law. These approaches include rigorous test mechanisms, techniques which correct errors incurred by architectural primitives using temporal and spatial redundancy, an increase in parallelism without an increase in the frequency of operation of computing devices, research into novel non-silicon materials for computing, including molecular devices, graphene and optoelectronics, and design automation-based approaches to reduce the impact of undesirable effects such as timing variations and noise susceptibility. By contrast, the central theme of our work, which we refer to as probabilistic and approximate design, is to design computing systems using circuit components which are susceptible to perturbations. Our research on probabilistic and approximate design has three interrelated aspects drawing on theoretical background from diverse disciplines such as probabilistic algorithms, theory of digital signal processing, thermodynamics, computer arithmetic and mathematical logic. The three aspects are:

1. Applications: In general, the set of all embedded applications may be classified under three categories. Those which benefit from perturbations, those which can tolerate but do not benefit from perturbations, and those which cannot tolerate perturbations. Our work aims to implement applications derived from the former two categories.

2. Device properties: In the application context outlined above, energy and performance efficiency can be obtained if the computing devices used for the implementation provide some mechanism through which “correctness” may be traded for cost. In our work, though our principles are general, we consider implementations based on complementary metal oxide semiconductor (cmos) technology. In this context, we have considered two phenomena in cmos: (i) the relationship between the probability of correct switching and energy consumption.

3. Design practice: Given applications which can benefit from or can tolerate perturbations, and cmos devices which exhibit a trade-off between perturbations and cost, we need a design methodology through which these applications may be implemented using these computing devices. Our design methodology is rooted in the theory of probabilistic Boolean logic (pbl), probabilistic arithmetic and approximate arithmetic. We distinguish between probabilistic design where the behavior of the computing substrate is probabilistic, and approximate design, where the behavior of the computing substrate is deterministic, but erroneous. In the second thread on approximate design spanning we survey our work which uses the relationship between energy consumption and switching speed of cmos devices to achieve a cost-quality trade-off in dsp applications. We note that the former (probabilistic design) approach is relevant for future technology generations where noise is likely to be comparable to signal levels and the latter (approximate design) approach based on conventional cmos technology is relevant for energy efficient implementations using current-day technology generations. The world accepts “analog computation,” which generates “good enough” results rather than totally accurate results. The data processed by many digital systems may already contain errors. In many applications, such as a communication system, the analog signal coming from the outside world must first be sampled before being converted to digital data. The digital data are
then processed and transmitted in a noisy channel before converting back to an analog signal. During this process, errors may occur anywhere. Furthermore, due to the advances in transistor size scaling, factors such as noise and process variations which are previously insignificant are becoming important in today’s digital IC design. Based on the characteristic of digital VLSI design, some novel concepts and design techniques have been proposed. The concept of error tolerance (ET) and the PCMOS technology are two of them. According to the definition, a circuit is error tolerant if: 1) it contains defects that cause internal and may cause external errors and 2) the system that incorporates this circuit produces acceptable results. The “imperfect” attribute seems to be not appealing. However, the need for the error-tolerant circuit was foretold in the 2003 International Technology Roadmap for Semiconductors (ITRS) To deal with error-tolerant problems, some truncated adders/multipliers have been reported but are not able to perform well in its speed, power, area, or accuracy. The “flagged prefixed adder” performs better than the non flagged version with a 1.3% speed enhancement but at the expense of 2% extra silicon area. As for the “lower-error area-efficient fixed-width multipliers” it may have an area improvement of 46.67% but has average error reaching 12.4%. Of course, not all digital systems can engage the error-tolerant concept. In digital systems such as control systems, the correctness of the output signal is extremely important, and this denies the use of the error tolerant circuit. However, for many digital signal processing (DSP) systems that process signals relating to human senses such as hearing, sight, smell, and touch, e.g., the image processing and speech processing systems, the error-tolerant circuits may be applicable.

4.1.1 Approximate Arithmetic
In this context, let us define an addition operator + \( \delta \) to be approximately \( \delta \) correct, if for any 0 \( \leq a, b \leq n \), let \((a + \delta b) - (a + b) \leq \delta \). We wish to distinguish approximately correct addition and probabilistic addition, by noting that the former is deterministic, whereas the latter is probabilistic. That is, for any two fixed inputs, the result of approximate addition (though incorrect) is the same across multiple additions, however in probabilistic addition, across multiple additions, the results may vary for the same inputs. If some property of cmos devices exists such that an implementation with a higher \( \delta \) incurs less energy when compared to a lower \( \delta \), these arithmetic operations could be used to implement dsp applications and would provide a novel way to trade quality of solution for cost. We have demonstrated that voltage overscaling—operating arithmetic circuits at a lower voltage such that their operating frequency violates their critical path delay—is a vlsi technique that can be used to implement approximately correct arithmetic operators.

1.1.2 Implementing Approximate Arithmetic
To illustrate the implementation of approximate arithmetic in vlsi, let us consider the case of an 8-bit ripple carry adder, composed of eight full adders. Let these full adders be labeled FA\( \delta \), FA6, . . . , FA0 with the adder FA\( i \) computing the ith bit of the output. We consider the case where each of these full adders are operated at the same supply voltage, thus incurring a delay \( d \) to compute the carry. In the conventionally correct operating scenario, the ripple carry adder would be operated at a frequency \( f = 1/8d \), since in the worst case, the critical path delay is 8\( d \). In the voltage overscaled case, to obtain energy efficiency, each full adder may be operated at a lower supply voltage, such that each of the full adders incur a delay \( d' > d \)—thus in the worst case, the critical path delay is 8\( d' \). Now to retain the performance as before, the ripple carry adder is operated at frequency \( f = 1/8d > 1/8d' \). In such a scenario, for certain input combinations—for example, the case where \( 11111111 \) is added to \( 00000001 \) and the carry produced at the least significant position needs to be propagated to the most significant position—the result of the adder would be read before the computation is completed, thereby producing an incorrect result. This is a technique of implementing approximate arithmetic in vlsi and the case where all of the full adders are operated at identical voltage levels is referred to as the uniform voltage scaling case or the uvos case.

V. MULTIPLY-ACCUMULATE UNIT
The Multiply-Accumulate (MAC) unit performs the Multiply instruction and the MAC instruction, which are essential for all DSP processors. In order to achieve high performance, the MAC unit is pipelined into three stages. This chapter discusses the design and implementation of the multiply-accumulate unit for the AsAP. The multiplier consists of a partial product multiplier that generates the result and a final carry-look ahead adder, as the converter between the two different number representations.

Another method to improve the speed of the multiplication operation is to improve the partial product generation step. This can be done in two ways:
1) Generate the partial products in a faster manner.
2) Reduce the number of partial products that need to be generated.

The first option can only be achieved if a different architecture for partial product generation is used. Considering that a bit is generated with just an AND gate delay (NAND followed by inverter in implementation), it seems that the partial product bits are already calculated in the fastest way possible. The second option will most likely take longer than an AND gate, but the reduced number of partial products...
results in a smaller tree and therefore reduces the time during the tree reduction step. As long as the time saved during tree reduction is greater than the extra time it takes to generate fewer partial products, it is beneficial to implement this option. The second step can be easily achieved by using Urdhva Multiplier. This hardware design Urdhva Multiplier is very similar to that of the famous array multiplier where an array of adders is required to arrive at the final product. All the partial products are calculated in parallel and the delay associated is mainly the time taken by the carry to propagate through the adders which form the multiplication array.

VI. PROBLEM IDENTIFICATION

1. The Approximate MAC unit which can achieve significant throughput improvement and total power reduction over proposed MAC unit designs which can be used in accuracy-configurable applications, and improves the achievable tradeoff between performance/Power and quality. The approximate designs produce almost-correct results at the given required accuracy, and obtain power reductions or performance improvements in return.

2. In order to improve the speed of the MAC unit, there are two major bottlenecks that need to be considered. The first one is the partial products reduction network that is used in the multiplication block and the second one is the accumulator. Both of these stages require addition of large operands that involve long paths for carry propagation. The approximate MAC unit is based on increasing the overall speed and throughput of the MAC and reducing the overall area of it.

3. The current methodology of multiplication leads to more consumption of power and reduction in efficiency. Performance and power evaluations are functions of the synthesis methodology.

4. Multipliers and dividers are basic blocks in Convolution and Deconvolution implementation. They consume much of time. With advances in technology, many researchers have tried and are trying to design multipliers and dividers which offer either of the following- high speed, low power consumption, regularity of layout and hence less area or even combination of them in multiplier and divider.

5. Hardware Complexity is the main problem. The multiplier which is based on Urdhva Tiryakbyham (Vertical & Crosswise) which is one of the sutra of ancient Indian Vedic Mathematics enables parallel generation of partial product and eliminates unwanted multiplication steps. Thus Vedic multipliers ensure substantial reduction of propagation delay in FFT processor. The FFT processor employing Vedic multiplier reduces hardware complexity in area and power in FPGA implementation.

PROPOSED METHODOLOGY

- PROPOSED MULTIPLY-ACCUMULATE UNIT (MAC) UNIT

The Multiply-Accumulate (MAC) unit performs the Multiply instruction and the MAC instruction, which are essential for all DSP processors. In order to achieve high performance, the MAC unit is pipelined into three stages. This chapter discusses the design and implementation of the multiply-accumulate unit for the AsAP. The multiplier consists of a partial product multiplier that generates the result and a final carry look-ahead adder, as the converter between the two different number representations. Another method to improve the speed of the multiplication operation is to improve the partial product generation step. This can be done in two ways:

1) Generate the partial products in a faster manner.
2) Reduce the number of partial products that need to be generated.

The first option can only be achieved if a different architecture for partial product generation is used. Considering that a bit is generated with just an AND gate delay (NAND followed by inverter in implementation), it seems that the partial product bits are already calculated in the fastest way possible. The second option will most likely take longer than an AND gate, but the reduced number of partial products results in a smaller tree and therefore reduces the time during the tree reduction step. As long as the time saved during tree reduction is greater than the extra time it takes to generate fewer partial products, it is beneficial to implement this option.

The second step can be easily achieved by using Urdhva multiplier. The below figure shows a partial product generation. Let’s take two inputs, each of 2 bits; say A1A0 and B1B0. Since output can be of four digits, say Q3Q2Q1Q0. As per basic method of multiplication, result is obtained after getting partial product and doing addition.

\[
\begin{array}{c}
A1 & A0 \\
X & B1 \ B0 \\
\end{array}
\]

\[
\begin{array}{c}
A1B0 & A0B0 \\
\end{array}
\]

\[
\begin{array}{c}
A1B1 & A0B1 \\
\end{array}
\]

\[
\begin{array}{c}
Q3 & Q2 & Q1 & Q0 \\
\end{array}
\]

In Vedic method, Q0 is vertical product of bit A0 and B0, Q1 is addition of crosswise bit multiplication i.e. A1 & B0 and A0 and B1, and Q2 is again vertical product of bits A1 and B1 with the carry generated, if any, from the previous addition during Q1. Q3 output is nothing but carry generated during Q2 calculation. This module is known as 2x2 multiplier block [21].
This hardware design Urdhav Multiplier is very similar to that of the famous array multiplier where an array of adders is required to arrive at the final product. All the partial products are calculated in parallel and the delay associated is mainly the time taken by the carry to propagate through the adders which form the multiplication array. In our design it deals with first of all the partial product generation method second the accumulation method.

1.2 Partial Product Generation
The first step in a multiplication is to generate the partial product bits. In this implementation, Urdhav multiplication approach is chosen because of the simple task of generating the multiplicand and twice the multiplicand for the partial products. The main advantage of the vedic multiplication algorithm (Urdhva-Tiryak Sutra) stems from the fact that it can be easily realized in hardware. The hardware realization of a 4-bit multiplier using this Sutra is shown in below figure. This hardware design is very similar to that of the famous array multiplier where an array of adders is required to arrive at the final product. All the partial products are calculated in parallel and the delay associated is mainly the time taken by the carry to propagate through the adders which form the multiplication array.

The proposed Vedic multiplier is based on the Vedic multiplication formulae (Sutras). These Sutras have been traditionally used for the multiplication of two numbers in the decimal number system. In this work, we apply the same ideas to the binary number system to make the proposed algorithm compatible with the digital hardware. Vedic multiplication based on some algorithm, used has been discussed below:

This work proposes a Approximate multiplier providing the solution of the aforesaid problems adopting the sutra of Vedic Mathematics called Urdhva Tiryakbhyam (Vertically and Cross wise). It can be shown that the design MAC unit is highly efficient in terms silicon area/speed.

1.3 Accumulation
After the partial products are generated using urdhav technique, the next step is to accumulate them. A 8 bit ripple carry is used to accumulate the partial products.

In our work a new approximate accumulation approach has been proposed. The accumulation of 8 bits is divided in to two sets, in first set the 4 bits (MSB to LSB) are added accurate and the other four bits are approximated i.e., only nanding those bits and leaving the carry behind there. By this approach the accuracy level is maintain up to 96% which is processed for a One lakh data.

There is an Example of two binary numbers from which we can explain how binary are added.

Let’s consider an example binary number

\[
\begin{align*}
A &= 01101101 \\
B &= 01101100
\end{align*}
\]

The original result is 220 where the approximate result is 195 whose accuracy is near about 90% which
is tolerable. After applying random data of one lakh and processed through this adder and the error results shows that only 4 % accuracy level is decreased and remaining 96% of data is accurate. In DSP system based ITRS report up to 10% error is tolerable which is stated in previous chapters.

1.4 Final stage for the MAC unit (Proposed addition arithmetic):
In a conventional adder circuit, the delay is mainly attributed to the carry propagation chain along the critical path, from the Least Significant Bit (LSB) to the Most Significant Bit (MSB). Meanwhile, a significant proportion of the power consumption of an adder is due to the glitches that are caused by the carry propagation. Therefore, if the carry propagation can be eliminated or curtailed, a great improvement in speed performance and power consumption can be achieved. In this study, we propose, an innovative and novel addition arithmetic that can attain great saving in speed and power consumption. This new addition arithmetic can be illustrated via an example shown in Fig. 3. We first split the input operands into two parts: an accurate part that includes several higher order bits and the inaccurate part that is made up of the remaining lower order bits. The length of each part need not necessary be equal. The addition process starts from the middle (joining point of the two parts) toward the two opposite directions simultaneously. In the example of Fig, the two 16-bit input operands, “1011001110011010” (45978) and “0110100100010011” (26899), are divided equally into 12 bits each for the accurate and inaccurate parts. The addition of the higher order bits (accurate part) of the input operands is performed from right to left (LSB to MSB) and normal addition method.

Hence using this approach fast adder algorithms and multiplication schemes has been used, and utilizes them in the design and implementation of a MAC unit. Using Vedic mathematics a new multiplier technique is proposed which reduces the delay and hardware complexity. A new approximate addition technique is proposed whose the lsb four bits are approximated and the proposed adder provides up to an average of 96 % accuracy. The overall area and Delay and Frequency analysis are presented and compared. From the results we can depict that up to 35% of reduction at all levels are achieved.

VII. IMPLEMENTATION

The figure1 below shows the schematic view of 8 bit proposed vedic multiplier i.e 8x8-bit Urddhav multiplier. For this implementation, Urddhav multiplication approach is chosen. The proposed multiplier consists of 16 bit adder.

7.1 IMPLEMENTATION OF 2X2 BITS VEDIC MULTIPLIER
It is clear that the basic building blocks of this multiplier are one bit multipliers and adders. One bit multiplication can be performed through two input AND gate and for addition, full adder can be utilized. The 2 x 2 bit multiplier is shown in figure.

Let’s take two inputs, each of 2 bits; say A1A0 and B1B0. Since output can be of four digits, say Q3Q2Q1Q0. As per basic method of multiplication, result is obtained after getting partial product and doing addition. In Vedic method, Q0 is vertical product of bit A0 and B0, Q1 is addition of crosswise bit multiplication i.e. A1 & B0 and A0 and B1, and Q2 is again vertical product of bits A1 and B1 with the carry generated, if any, from the previous addition during Q1. Q3 output is nothing but carry generated during Q2 calculation. This module is known as 2x2 multiplier block [5].
7.2 IMPLEMENTATION OF 4X4 BITS VEDIC MULTIPLIER

Let's analyze 4x4 multiplications, say A3A2A1A0 and B3B2B1B0. Following are the output line for the multiplication result, Q7Q6Q5Q4Q3Q2Q1Q0. Let's divide A and B into two parts, say A3 A2 & A1 A0 for A and B3 B2 & B1 B0 for B. Using the fundamental of Vedic multiplication, taking two bit at a time and using 2 bit multiplier block,

Fig8.3. RTL view of 4*4

7.3 IMPLEMENTATION OF 8X8 BITS VEDIC MULTIPLIER

The 8x 8 bit multiplier is structured using 4X4 bit blocks as shown in figure. In this figure the 8 bit multiplicand A can be decomposed into pair of 4 bits AH-AL. Similarly multiplicand B can be decomposed into BH-BL. The 16 bit product can be written as:

\[ P = A \times B = (AH-AL) \times (BH-BL) = AH \times BH + AH \times BL + AL \times BH + AL \times BL \]

The outputs of 4X4 bit multipliers are added accordingly to obtain the final product. Thus, in the final stage two adders are also required. Now the basic building block of 8x8 bits Vedic multiplier is 4x4 bits multiplier which implemented in its structural model. For bigger multiplier implementation like 8x8 bits multiplier the 4x4 bits multiplier units has been used as components.

The figure8.4 below shows a 16 bit Approximate Adder. The 16 bits are divided equally into 12 bits each for the accurate and 4 bits are the inaccurate parts. The higher order are the 12 bits (accurate part) are added by normal addition and the lower order 4 bits (inaccurate part) are nanded and the addition is performed from LSB to MSB.

Fig8.4. Schematic View of Approximate Adder

The proposed approximate and Accurate Multiply and accumulate unit are designed on Xilinx FPGA Virtex 6 device and analyzed the results. The goal of this project is to design and implement a MAC [9] unit. The MAC[10] unit is a 8x8-bit Urdhav multiplier with a 16-bit accumulator.

The below figure8.5 and figure 8.6 shows the schematic view of approximate and accurate MAC unit.

Fig8.5. Schematic view of Accurate Multiply and accumulate Unit

Fig8.6. Schematic view of Approximate Multiply and accumulate Unit

VIII. RESULT ANALYSIS & CONCLUSION

The proposed Approximate and Accurate multiplier and adder and Multiply and accumulate unit are designed on Xilinx FPGA Virtex 6 device and analyzed the results.

- Approximate Adder Accuracy Level=99%
- Approximate Multiplier Accuracy Level=85%
- Approximate MAC Unit Accuracy Level=95%

The FPGA comparison analysis of proposed and accurate are shown below:
1.5 **LUT** (Logic Blocks):

![Fig 9.1](image1)

**Fig 9.1** Comparison analysis of Proposed and Accurate ripple carry Adder

From the above graph it is clear that 8.3% reduction in logic blocks is achieved.

![Fig 9.2](image2)

**Fig 9.2** Comparison analysis of Proposed and Accurate Multiplier

From the above graph it is clear that 31.25% reduction in logic blocks is achieved.

![Fig 9.3](image3)

**Fig 9.3** Comparison analysis of Accurate and Proposed MAC unit

From the above graphs we can see that the reduction in logic block 35% reduction in logic blocks is achieved.

1.6 **Delay**:

![Fig 9.4](image4)

**Fig 9.4** Comparison analysis of Proposed and Accurate Multiplier

![Fig 9.5](image5)

**Fig 9.5** Comparison analysis of Proposed and Accurate Adder

![Fig 9.6](image6)

**Fig 9.6** Comparison analysis of Proposed and Accurate Multiplier and accumulate

1.7 **Frequency**:

![Fig 9.7](image7)

**Fig 9.7** Comparison Analysis of Multiplier

![Fig 9.8](image8)

**Fig 9.8** Comparison analysis of Adder

![Fig 9.9](image9)

**Fig 9.9** Comparison analysis of MAC unit
CONCLUSION

This thesis explores and analyzes fast adder algorithms and multiplication schemes, and utilizes them in the design and implementation of a MAC unit. Using Vedic mathematics a new multiplier technique is proposed which reduces the delay and hardware complexity.

A new approximate addition technique is proposed whose the lsb four bits are approximated and the proposed adder provides upto an average of 96 % accuracy. The overall area and Delay and Frequency analysis are presented and compared. From the results we can depict that upto 35% of reduction at all levels are achieved.

REFERENCES


[6]. Rudagi, J M; Ambli; Vishwanath; Munavalli; Vishwanath; Patil; Ravindra; Sajan; Vinaykumar, "Design and implementation of efficient multiplier using Vedic Mathematics," Advances in Recent Technologies in Communication and Computing (ARTCom 2011), 3rd International Conference on, vol., no., pp.162,166, 14-15 Nov, 2011


