International Journal of Research Publication and Reviews

Journal homepage: www.ijrpr.com ISSN 2582-7421

Design of Vedic Multiplier Using a for Adaptive Traversal Filters Low-Power Distributed Arithmetic

*Komal Rai *[a] and Chandrahas Sahu [b]*

[a] *M. Tech Research Scholar in Electronics and Telecommunication (VLSI Design), Shri Shankaracharya Technical Campus, Bhilai, India*
[b]*Asst. Professor, Electronics and Telecommunication (VLSI Design) Shri Shankaracharya Technical Campus, Bhilai, India*
[a]*komalrai2016@gmail.com,* [b]
Email: *chandrahassscet@gmail.com*

A B S T R A C T

A project concerning adaptive filters, with a particular emphasis on the LMS (Least Mean Squares) method for adaptive traversal filters. The project entails developing this method in Verilog HDL, synthesising it in Xilinx XST, and simulating it in Modelsim 6.5. The application side of the research entails recognising an unknown system in communication and digital signal processing settings such as Channel Equalisation, Adaptive Noise Cancellation, and Adaptive Echo Cancellation. If you have any particular queries or want further information about this project, please contact us. Filters are used to extract usable information from noisy signals. Adaptive filters change their settings dependent on the input data, making them appropriate for scenarios with changing surroundings and unknown signal statistics. The project's overall goal is to demonstrate the installation and application of The LMS method is used in signal processing for adaptive filters, with an emphasis on communication and digital signal processing applications. This sort of work is necessary to improve the quality and dependability of diverse systems in the presence of noise and changing environments.

Keywords: Design of Vedic Multiplier, FIR Filter, Adaptive Traversal Filters, LMS (least mean squares) Algorithm.

1. INTRODUCTION

the notion of adaptive filters, as well as the least-mean-square (LMS) algorithm, which is a popular way for creating adaptive filters. Please contact us if you have any queries or would want to discuss specific features of adaptive filters or the LMS algorithm. Whether it's about the fundamental mathematics, practical applications, algorithm changes, or any other connected issue; Discrete-time (or digital) filters are widely used in signal processing applications nowadays. Filters are used to generate desired spectral features of a signal, reject unwanted signals such as noise or interferers, reduce bit rate in signal transmission, and so on. The idea of making filters adaptable, i.e., modifying the parameters (coefficients) of a filter based on some algorithm, solves issues that we face. may not be known in advance, such as the properties of the signal, the undesirable signal, or the effect of a system on the signal that we wish to compensate for. Adaptive filters can adapt to a new environment and even track changes in signal or system parameters over time. The initial work on adaptive filters may be traced back to the late 1950s, when several researchers were independently working on theories and implementations of such filters. The least-mean-square (LMS) method sprang from this early work as a simple but effective strategy for designing adaptive transversal (tapped-delay-line) filters.

Widrow and Hoff created the LMS algorithm in 1959 while working on the adaptive linear element, a pattern-recognition machine. Adaline is another name for the Adaline [1, 2]. In the sense that it iterates each transversal filter tap weight in the direction of the instantaneous gradient of the squared error signal with respect to the tap weight in question, the LMS method is a stochastic gradient algorithm. Let w(n) be the tap-weight vector created by the LMS filter at each iteration (time). n. The recursive equation defines the filter's adaptive behaviour in detail (assuming difficult data).

w(n+1) = w(n) multiplied by x(n)*e(n)*m

Where x (n) denotes the tap-input vector, d(n) the anticipated response, e(n)=d(n)-x(n), and m the step-size parameter. The number included in square brackets represents the error signal. The asterisk represents complex conjugation, whereas the superscript H represents Hermitian transposition (ordinary transposition combined with complex conjugation).

1.1 URDHVA TIRYAGBHYAM

It's interesting to see the connection you've drawn between the Urdhva Tiryakbhyam Sutra from Vedic Mathematics and the LMS algorithm used in adaptive filters. The Urdhva Tiryakbhyam Sutra is indeed a multiplication formula from Vedic Mathematics that involves both vertical and crosswise calculations for efficient multiplication. Vedic Mathematics is a system of mathematical calculations that has its roots in ancient Indian texts and offers alternative techniques for arithmetic operations.In the context of your mention of the LMS algorithm, it's important to note that while there might be parallels or conceptual connections between ancient mathematical techniques and modern signal processing algorithms, the specifics and purposes of these two are quite different. The LMS algorithm is a method used in adaptive filtering for updating the coefficients of a filter to minimize the error between the filter's output and a desired signal. It's based on optimization principles and gradient descent, and its application is within the realm of signal processing and machine learning.The Vedic Multiplier you mentioned seems to be an application of the Urdhva Tiryakbhyam Sutra for digital multiplication, possibly for hardware implementations like digital circuits. While the naming might be inspired by the Vedic Mathematics concept, the actual techniques and algorithms used in digital multipliers are designed to suit the requirements and constraints of modern digital systems.

If you have any specific questions about these concepts or their connections,

2. VERILOG HDL

It seems like you've provided a detailed overview of Verilog and its history, as well as some of its features and benefits. Verilog is indeed a hardware description language (HDL) that is widely used in the field of electronic design for modeling and describing digital systems. It's used for various purposes, including design, verification, and implementation of electronic circuits. Here's a summary of the key points you've mentioned:

Origin and Development: Verilog was originally developed by Gateway Design Automation in 1984. Later, Cadence Design Systems acquired Gateway and continued developing Verilog. It gained popularity as a hardware description language for modeling digital systems.

Standardization: The specifications for Verilog-HDL were released by Cadence and were accepted as an IEEE standard (IEEE 1364) in 1995. The standardization included support for the Programming Language Interface (PLI) version 1.0 and later PLI 2.0 (VPI).

Verilog 2001: In 2001, the IEEE updated the Verilog standard, often referred to as Verilog 2001. This version introduced enhancements to the language and its capabilities.

Syntax and Abstraction Levels: Verilog's syntax is often compared to the C programming language, making it relatively easy for those with C programming experience to learn. Verilog allows different levels of abstraction to be mixed within the same model, enabling designers to describe circuits at various levels, from gates to high-level behavioral code.

Abstraction Flexibility: Designers can model hardware using different abstraction levels such as switches, gates, Register Transfer Level (RTL), or behavioral descriptions. This flexibility allows designers to choose the appropriate level of abstraction for their specific needs.

Conversion from VHDL: You mentioned converting an IOP (Input-Output Processor) VHDL design to Verilog. This could be for various reasons, such as compatibility with specific tools or platforms that prefer Verilog, or the familiarity of the design team with Verilog.

Comparison with VHDL: You briefly compared Verilog with VHDL. VHDL is another widely used HDL for describing hardware systems. While VHDL offers more programming constructs and supports 9-value logic, Verilog is often noted for its C-like programming style and closeness to hardware representation.

Synthesis and Tool Support: Verilog is well-supported by logic synthesis tools, which convert high-level descriptions into gate-level representations suitable for hardware implementation.

Overall, your description provides a comprehensive overview of Verilog, its history, features, and advantages, as well as its role in the field of electronic design. It's worth noting that as of my last update in September 2021, these details are accurate, but there might have been further developments or changes in the field since then.

3. FILTER DESCRIPTION

It seems like you've provided a passage discussing different types of digital filters, specifically finite impulse response (FIR) filters and infinite impulse response (IIR) filters. FIR filters are highlighted for their linear phase characteristics, stability, and ease of implementation, even though they might require more taps compared to IIR filters. The text mentions that in applications where a fixed number of taps are used in commercial filter chips, unused taps are filled with zero coefficients, leading to unnecessary calculations.

To address this issue, the paper proposes two special features: a data reuse structure and a recurrent-coefficient scheme. These features aim to efficiently implement variable-length taps in FIR filter chips. The proposed architecture, utilizing multiplexers (MUXs), registers, and a feedback loop, can reduce the number of gates by over 20% compared to existing chips that use memory, an address generation unit, and a modulo unit for circular memory access.In essence, the passage emphasizes the advantages of FIR filters and discusses a novel approach to efficiently implement variable-length taps in FIR filter chips, ultimately reducing gate count and improving performance.

In general, FIR filtering is defined by a simple convolution process, as shown in Equation (1).

$$y[n] = \sum_{k=0}^{N-1} h[k] \cdot x[n-k] \qquad (1)$$

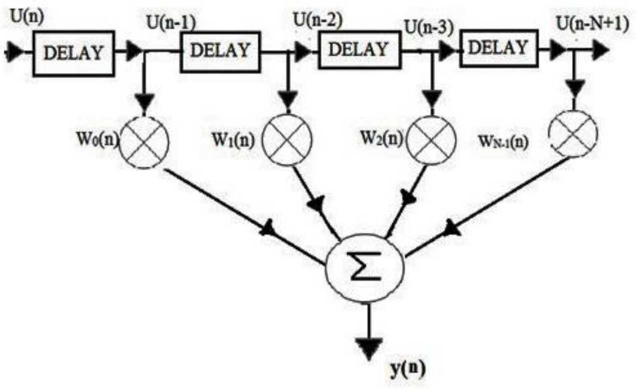


Figure3.1: Diagram of a Transverse Filter

4. THE LMS ADAPTATION ALGORITHM

Widrow and Hoff developed the LMS algorithm in 1960 to be utilised in neural network training. It seeks the desired weight vector by using a crude gradient approximation. This procedure is used to determine the weight vectors for training the ALC (Adaline). The learning principles may be included into the same device, allowing it to automatically adjust when the desired inputs and outputs are supplied. The weight vectors' values vary as each input-output combination is handled. This process is repeated until the ALC produces the right outputs. This is a true training procedure because the weight vector value does not need to be calculated explicitly. Where is a convergence factor that dictates how large or tiny the signal is? Going through the filter; its value is critical for convergence speed and LMS algorithm reliability. It accepts values between 0 and 2/MSmáx, where M is the number of filter steps and Smáx is the greatest value of the input u step's power spectral density.The LMS (least mean squares) approach is a steepest descent approximation that employs an immediate estimate of the gradient vector of a cost function. The gradient is estimated using sample values from the tap-input vector and an error signal. The method iterates over the filter, shifting each coefficient in the direction of the calculated gradient [1]. A reference is required for the LMS algorithm. The LMS method requires a reference signal d[n] that represents the intended filter output. The error signal is the difference between the reference signal and the actual output of the transversal filter.

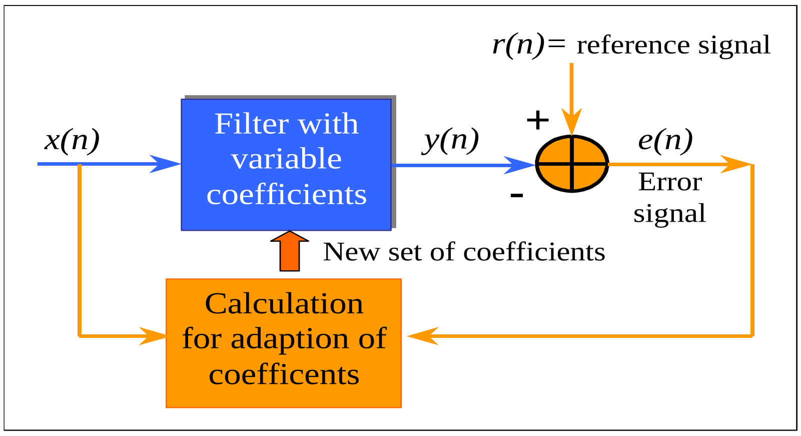
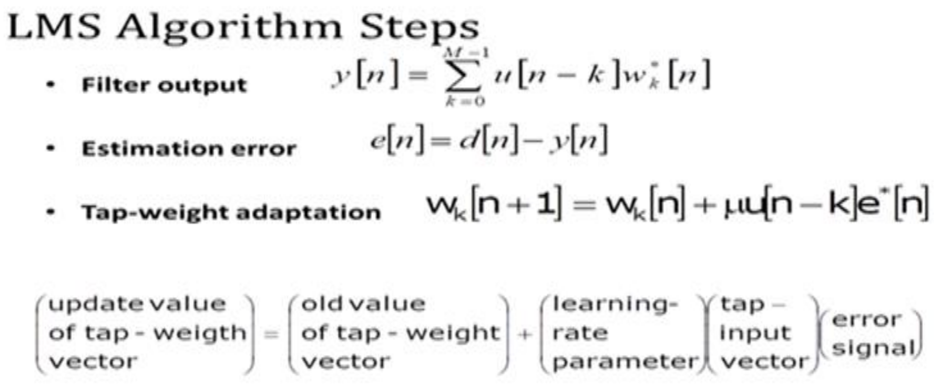

r(n)= reference signal
x(n)
Filter with variable coefficients
y(n)
+
-
e(n)
Error signal
New set of coefficients
Calculation for adaption of coefficents

Figure 4.1: Block diagram of LMS filter



LMS Algorithm Steps

- **Filter output** $y[n] = \sum_{k=0}^{M-1} u[n-k]w_k^*[n]$
- **Estimation error** $e[n] = d[n] - y[n]$
- **Tap-weight adaptation** $w_k[n+1] = w_k[n] + \mu u[n-k]e^*[n]$

$$\begin{pmatrix}\text{update value}\\ \text{of tap - weigth}\\ \text{vector}\end{pmatrix} = \begin{pmatrix}\text{old value}\\ \text{of tap - weight}\\ \text{vector}\end{pmatrix} + \begin{pmatrix}\text{learning-}\\ \text{rate}\\ \text{parameter}\end{pmatrix}\begin{pmatrix}\text{tap -}\\ \text{input}\\ \text{vector}\end{pmatrix}\begin{pmatrix}\text{error}\\ \text{signal}\end{pmatrix}$$

Figure 4.1: LMS Filter Block Diagram

4.1 THEMULTIPLIER ARCHITECTURE

Vedic Multiplication Technique and Its Application To Designing Multiplier Architecture. The Vedic multiplication technique is an ancient Indian method of multiplication that involves breaking down numbers into smaller parts and performing parallel calculations to arrive at the final product. This technique is based on various sutras (aphorisms) from Vedic mathematics. In the context of multiplier architecture, the Vedic multiplication technique can be utilized to design a parallel multiplier that computes partial products and their sums simultaneously, thus potentially improving the speed of multiplication operations. The advantage of this approach is that it can lead to a more efficient and streamlined design compared to other multiplication algorithms.

In the specific example you provided, where two 64-bit numbers are being multiplied, the multiplier and multiplicand are divided into smaller blocks to enable parallel processing. The numbers are broken down into 32-bit blocks, and then further divided into 16-bit blocks, and so on, until you reach the smallest unit of calculation, which in this case seems to be two bits. Each of these blocks is then multiplied using the Vedic multiplication technique, and the results are combined in a parallel manner to obtain the final product of the full 64-bit multiplication. The key advantage of this approach is that it can lead to a multiplier design that requires fewer logic gates compared to some traditional multiplication algorithms. This reduction in the number of gates can translate to faster multiplication operations and potentially more efficient hardware implementations. It's important to note that the details of the architecture would depend on the specific Vedic multiplication technique being used and the design considerations for the hardware implementation. Additionally, while the Vedic multiplication technique can offer advantages in certain cases, modern processors also employ a variety of multiplication algorithms optimized for speed and efficiency.

64 BIT VEDIC MULTIPLIER

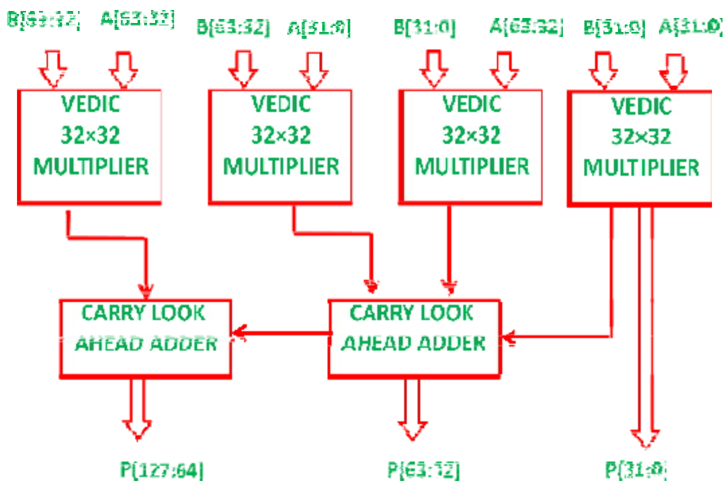


Figure 4.2: Vedic Multiplier 64x64 Bit Architecture Proposed (P127-P64) & (P63-P32) & (P31-P0) Result

32 BIT VEDIC MULTIPLIER

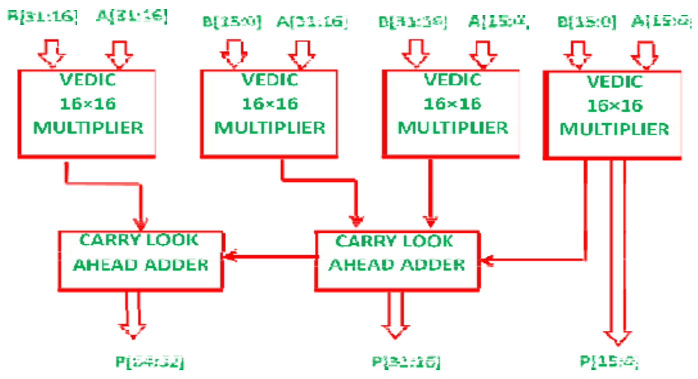


Figure 4.3: Vedic Multiplier Suggested in 32X32 Bits.

(P63-P32) & (P31-P16) & (P15-P0) = Result

16 BIT VEDIC MULTIPLIER

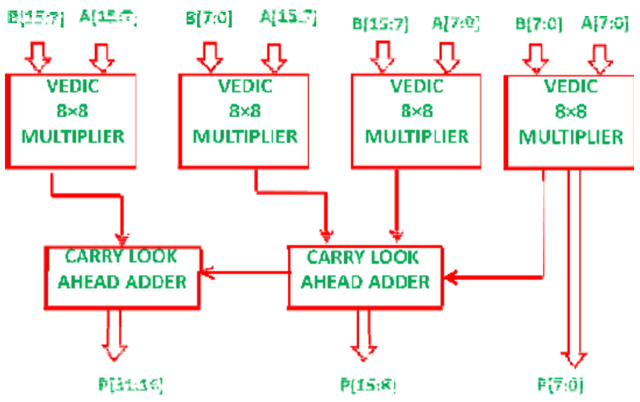


Figure 4.4: Decomposed 16x16 Bits Vedic Multiplier

(P31- P16) & (P15- P8) & (P7- P0) = Result

5. SIMULATION RESULT

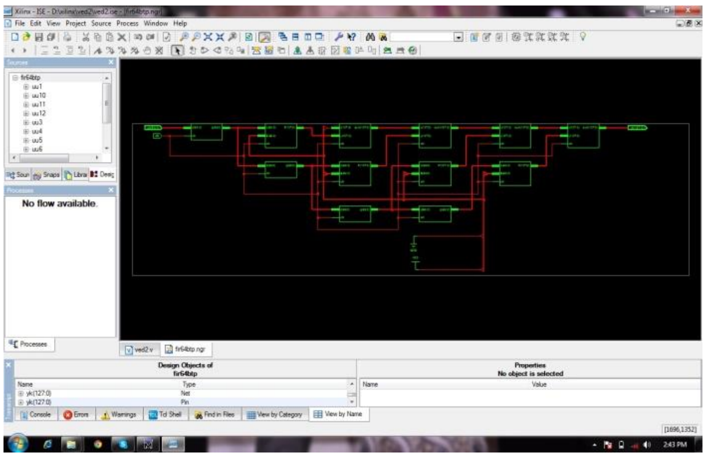


Figure 5.1: Diagram of an LMS filter

The Macro Statistics from the HDL synthesis report for a specific design, presumably a 64x64-bit Vedic Multiplier implementation. These statistics give insight into the resources used and the structure of the synthesized design. Let's break down the information you've provided:

Adders/Subtractors: There is a total of 1 adder/subtractor used in the design.

128-bit adder: There's one 128-bit adder in the design.

Registers: The design uses a total of 8199 registers.

128-bit register: There's one 128-bit register.

4-bit registers: There are 8192 4-bit registers.

64-bit registers: There are 6 64-bit registers.

XOR Gates: The design uses a total of 24582 XOR gates.

1-bit XOR2: There are 16384 1-bit XOR gates.

128-bit XOR2: There are 12 128-bit XOR gates.

16-bit XOR2: There are 768 16-bit XOR gates.

32-bit XOR2: There are 192 32-bit XOR gates.

4-bit XOR2: There are 4096 4-bit XOR gates.

64-bit XOR2: There are 58 64-bit XOR gates.

These statistics provide insight into the utilization of various resources in the design, such as adders, registers, and XOR gates. The 128-bit adder is likely used for the final accumulation stage, where the partial products are added to produce the final result. The registers are used for storing intermediate results and operands during various stages of computation. XOR gates are likely used in the Vedic multiplier architecture for various computations involving partial products and accumulations.

Overall, this information gives you an idea of the complexity and resource utilization of the synthesized Vedic multiplier design. It seems like a relatively efficient and compact design, considering the number of resources used for a 64x64-bit multiplication operation.8-bit xor2 : 3072

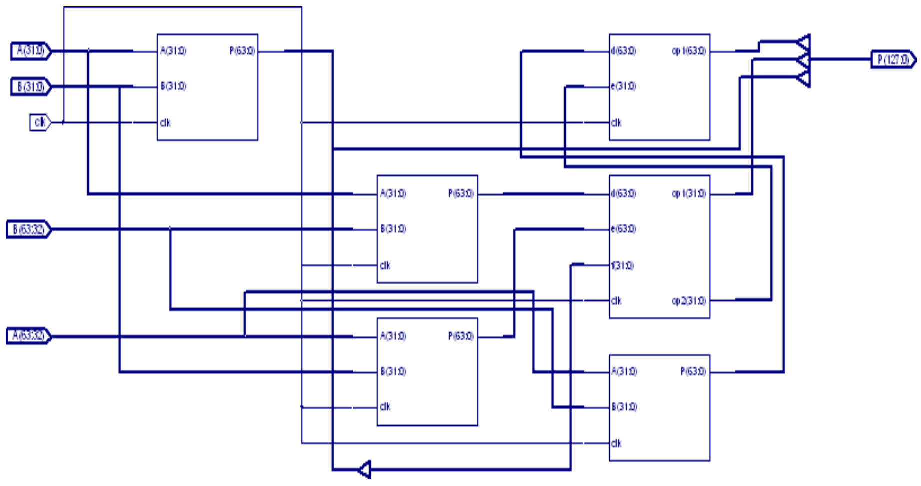


Figure 5.2 64x64 Bit Vedic Multiplier Synthesis Result

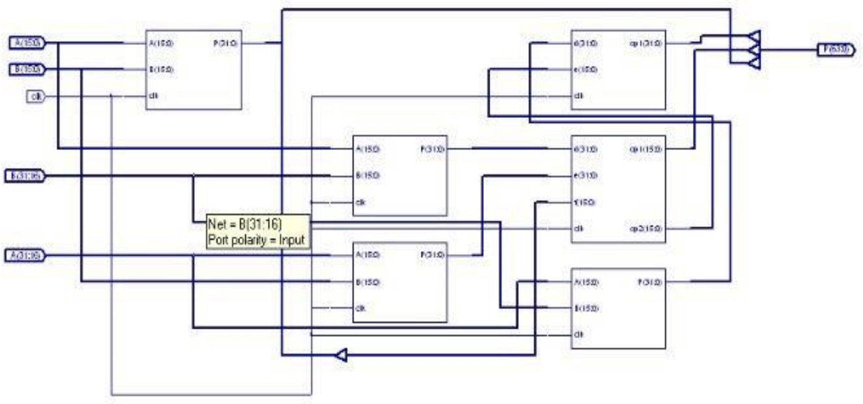


Figure 5.3 32x32 Bit Vedic Multiplier Synthesis Result

An HDL synthesis report for a 64-bit multiplier design. This report provides information about various aspects of the synthesized design, such as the number of registers, flip-flops, and XOR gates used. Let me break down the information for you:

# Registers: This indicates the total number of registers used in the design. In this case, there are 4096 registers.

Flip-Flops: Flip-flops are a type of digital circuit element used for storing binary information (0 or 1). In your design, there are also 4096 flip-flops, which is consistent with the number of registers.

# Xors: This indicates the total number of XOR gates used in the design. In your design, there are 1022 XOR gates in total.

16-bit xor2: This likely refers to 16-bit XOR gates, and there are 96 of them in your design. These gates could be used for performing XOR operations on 16-bit data.

32-bit xor2: Similarly, there are 24 instances of 32-bit XOR gates, which could be used for XOR operations on 32-bit data.

4-bit xor2: There are 512 4-bit XOR gates. These gates might be used for performing XOR operations on 4-bit data.

64-bit xor2: There are 6 instances of 64-bit XOR gates. These gates could be used for XOR operations on 64-bit data.

8-bit xor2: Lastly, there are 384 8-bit XOR gates, likely used for XOR operations on 8-bit data.

The information you've provided gives insights into the structure of the synthesized multiplier design, particularly in terms of registers, flip-flops, and XOR gates of various sizes. This type of information is essential for assessing the efficiency and complexity of a digital design, and it's commonly used by hardware engineers during the design and optimization process.

Output. The error signal is the difference between the reference signal and the actual output of the transversal filter.

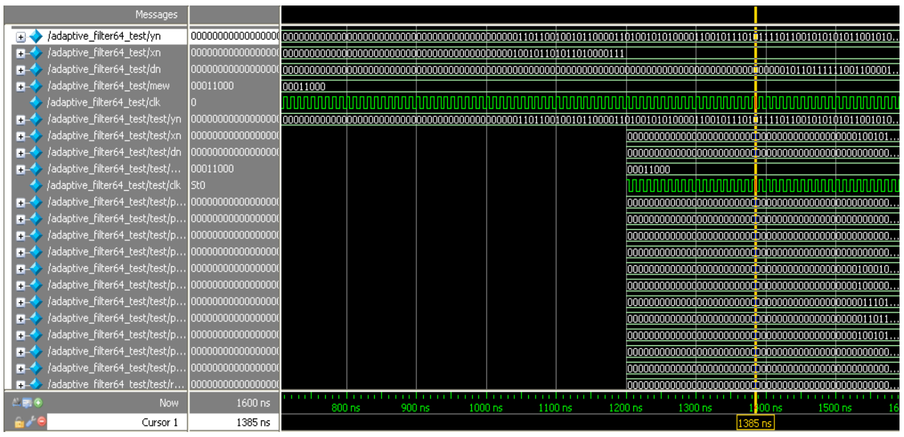
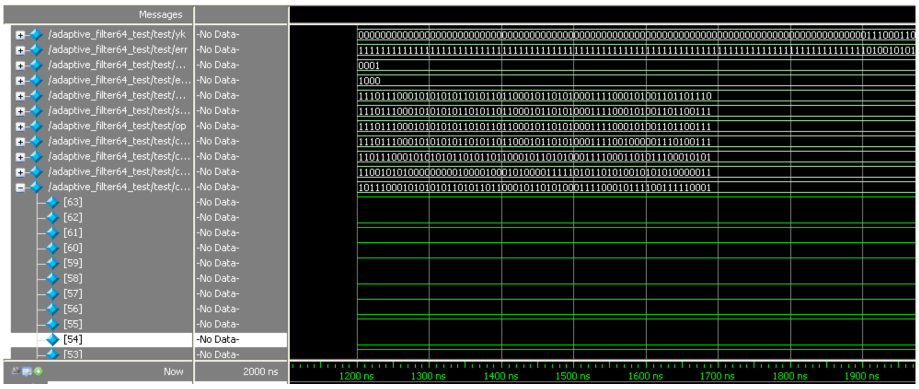

Messages
/adaptive_filter64_test/yn
/adaptive_filter64_test/xn
/adaptive_filter64_test/dn
/adaptive_filter64_test/mew
/adaptive_filter64_test/clk
/adaptive_filter64_test/test/yn
/adaptive_filter64_test/test/xn
/adaptive_filter64_test/test/dn
/adaptive_filter64_test/test/...
/adaptive_filter64_test/test/clk
Now 1600 ns
Cursor 1 1385 ns
800 ns
900 ns
1000 ns
1100 ns
1200 ns
1300 ns
1500 ns
1385 ns

Figure 5.4: LMS filter Simulation Result.


Messages
/adaptive_filter64_test/test/yk
/adaptive_filter64_test/test/err
/adaptive_filter64_test/test/op
-No Data-
Now
2000 ns
1200 ns
1300 ns
1400 ns
1500 ns
1600 ns
1700 ns
1800 ns
1900 ns
Figure 5.5: FIR Filter Simulation Result.

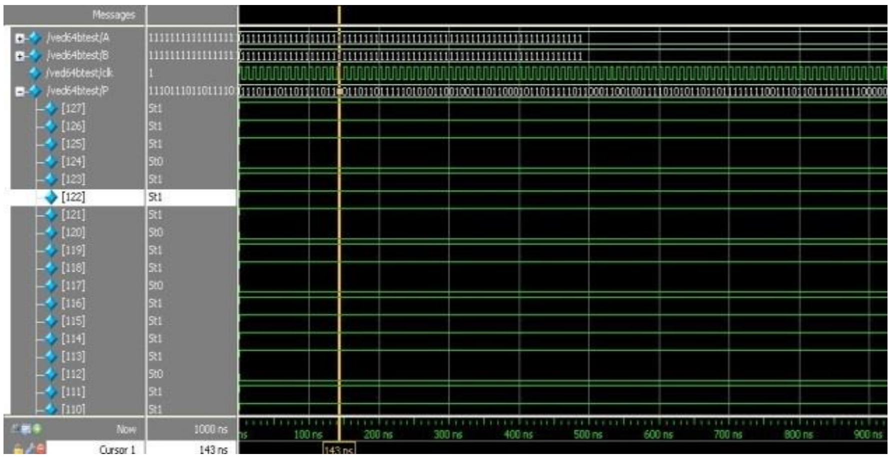
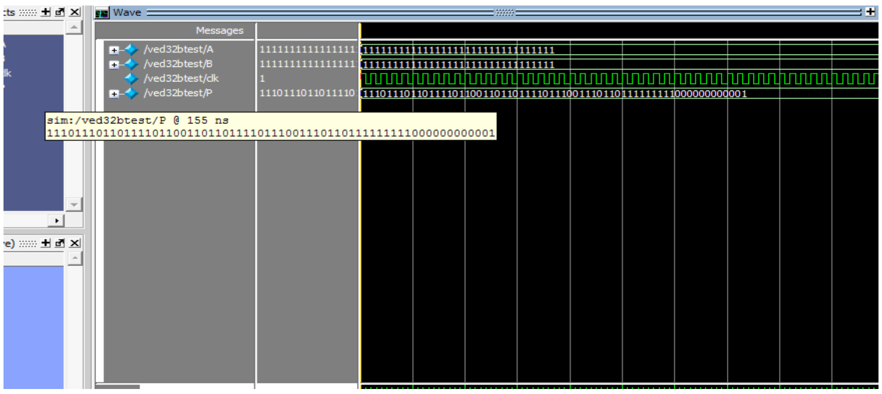


Figure 5.6: 64x64 Bit Multiplier Simulation Result.



```
sim:/ved32btest/P @ 155 ns
1110111011011110110011011011110111001110110111111111000000000001
```

Figure 5.7: 32x32 Bit Multiplier Simulation Result.

6. CONCLUSION

Research paper or project that presents an Adaptive Finite Impulse Response (FIR) filter system using the Least Mean Square (LMS) Algorithm. This system offers advantages in terms of flexibility, power efficiency, and configuration time. The project also incorporates a newly proposed Vedic multiplier concept from IETE journals, which is highly efficient in terms of speed. This Vedic multiplier's regular and parallel structure makes it suitable for implementation on silicon, leading to reduced hardware costs. The implementation of the Adaptive LMS FIR filter is done using Verilog Hardware Description Language (HDL), and synthesis is performed using the Xilinx 7.1i tool. Simulation of the system is carried out using Modelsim 6.3f. The presented Adaptive LMS FIR filter system is designed to autonomously adapt and provide optimal solutions for FIR filter realization, making it suitable for various communication and digital signal processing applications such as Channel Equalization, Adaptive Noise Cancellation, Adaptive Echo Cancellation, and System Identification. The proposed Vedic multiplier architecture demonstrates speed improvements compared to existing multiplier architectures, including the 64x64 Vedic multiplier and other lower-bit multipliers using the "Urdhva Tiryakbhyam" Sutra. The architecture not only offers speed improvements but also showcases simplicity in gate-level design complexity. This architecture is particularly well-suited for multiplying numbers with sizes greater than 64 bits. If you have any specific questions or if you'd like to discuss any aspects of this project further.

Future scope:

Explore ways to modify or enhance the Vedic multiplier concept to accommodate different multiplication scenarios and potentially further improve speed and efficiency.Remember that the future scope of this project will depend on the evolving needs and trends in communication, signal processing, and hardware design. Continuously keeping up with advancements in technology and adapting the project accordingly will ensure its relevance and impact in the long run.

REFERENCES

[1]. Haykin, Simon, Adaptive Filter Theory, Prentice Hall, Upper Saddle River, New Jersey, 1996.

[2]. Bernard Widrow and Samuel D. Stearns: "Adaptive Signal Processing", Prentice-Hall, Inc., UpperSaddle River, NJ, 1985.

[3]. Honig, Michael L., Messerschmitt, David G., Adaptive Filters, Structures, Algorithms and Applications, Kluwer Academic Publishers, Boston, 1984.

[4]. Jenkins, W. Kenneth, Hull, Andrew W.,Strait, Jeffrey C., Schnaufer, Bernard A., Li, Xiaohui, Advanced Concepts in Adaptive Signal Processing, Kluwer Academic Publishers, Boston, 1996.

[5]. Purushottam D. Chidgupkar and Mangesh T. Karad, "The Implementation of Vedic Algorithms in Digital Signal Processing", Global J. of Engng. Educ., Vol.8, No.2 © 2004 UICEE Published in Australia.

[6]. Himanshu Thapliyal and Hamid R. Arabnia, "A Time-Area- Power Efficient Multiplier and Square Architecture Based On Ancient Indian Vedic Mathematics", Department of Computer Science, The University of Georgia, 415 Graduate Studies Research Center Athens, Georgia 30602-7404, U.S.A.

[7]. E. Abu-Shama, M. B. Maaz, M. A. Bayoumi, "A Fast and Low Power Multiplier Architecture", The Center for Advanced Computer Studies, The University of Southwestern Louisiana Lafayette, LA 70504.

[8]. Implementation of Multiplier using Vedic Algorithm ;Poornima M, Shivaraj Kumar Patil, Shivukumar , Shridhar K P , Sanjay H International Journal of Innovative Technology and Exploring Engineering (IJITEE) ISSN: 2278-3075, Volume-2, Issue-6, May 2013.

[9].Design of High Speed Vedic Multiplier using Vedic Mathematics Techniques; G.Ganesh Kumar, V.Charishma;International Journal of Innovative Technology and Exploring Engineering (IJITEE) ISSN: 2278-3075, Volume-2, Issue-6, May 2013

[10]. Purushottam D. Chidgupkar and Mangesh T. Karad, "The Implementation of Vedic Algorithms in Digital Signal Processing", Global J. of Engng. Educ., Vol.8, No.2 © 2004 UICEE Published in Australia.

[11]. E. Abu-Shama, M. B. Maaz, M. A. Bayoumi, "A Fast and Low Power Multiplier Architecture", The Center for Advanced Computer Studies, The University of Southwestern Louisiana Lafayette, LA 70504.

[12]. Jenkins, W. Kenneth, Hull, Andrew W ,Strait, Jeffrey C., Schnaufer, Bernard A., Li, Xiaohui, Advanced Concepts in Adaptive Signal Processing, Kluwer Academic Publishers, Boston, 1996.

[13]. Himanshu Thapliyal and Hamid R. Arabnia, "A Time-Area- Power Efficient Multiplier and Square Architecture Based On Ancient Indian Vedic Mathematics", Department of Computer Science, The University of Georgia, 415 Graduate Studies Research Center Athens, Georgia 30602-7404, U.S.A.