Improved 24-bit Binary Multiplier Architecture for use in Single Precision Floating Point Multiplication

Marcus Lloyde George
Dept of Electrical and Computer Engineering
University of West Indies, St Augustine, Trinidad and Tobago
marcus.george99@yahoo.com

Abstract- Arithmetic Logic Units (ALUs) are very important components of processors performing arithmetic operations such as multiplication, division, addition, subtraction, cubing and squaring. Of all operations, multiplication is most frequently used in ALUs. Floating point multiplication is a very important component of many engineering applications such as signal processing, video processing and image processing. The dynamic range of numbers represented by floating point arithmetic is very large when compared to fixed point numbers of the same bit-width. This paper presents the development of a novel 24-bit binary multiplier architecture for use in implementation of a single precision floating point multiplier for use in data intensive applications, requiring low delay benefits. The system was synthesized for a variety of FPGA targets using Xilinx ISE Design Suite 14.7 Commercial Edition and performance was simulated with ISIM which was available from Xilinx ISE Design Suite 14.7. The implemented system was compared with existing implementations of 8-bit, 16-bit and 24-bit binary multipliers in terms of path delay. The novel 24-bit binary multiplier architecture achieved shorter path delay than its existing counterpaths.

Keywords- Arithmetic Logic Unit; Arithmetic Circuits; Binary Multiplier; Floating-Point Multiplier; Arithmetic Logic; FPGAs in Arithmetic.

I. INTRODUCTION

Arithmetic Logic Units (ALUs) are very important components of processors that perform various arithmetic operations such as multiplication, division, addition, subtraction, cubing, squaring, etc. Off all operations the operation of multiplication is most elementary and most frequently used in ALUs. It allows one number to be scaled by another number. The operation of multiplication also forms the basis of many other complex arithmetic operations such as cubing, squaring, convolution, etc.

The binary multiplication methodology is quite similar to the methodology utilized in the decimal system where multiplication is concerned. The methodology involves the multiplication of the multiplicand by the multiplier, one bit at a time, in each event a partial product is generated. After the multiplicand is multiplied by each bit of the multiplier all partial products would have been generated. The final step involves the summation of all generated partial products to give the complete product [1].

If the two binary numbers to be multiplied were in fractional representation then the placement of the binary point in the final product is determined in the same way in which the product of decimal numbers in fractional form were determined. An approach to multiplier fractional numbers is by first counting the number of decimal places after the binary point for both multiplicand and multiplier, then finding the sum of these two numbers to give a total that can be denoted by number *P* [1].

The next step involves multiplication of both numbers as if they were integers to find the product, then placing the binary point such that *P* binary numbers are behind the binary point. In essence the binary point in the product is placed before total number of places *P* counted from right. The multiplication by zero will produce a partial product of zero. A multiplication by 1 resulted in the bits of multiplicand shifted towards the left by one bit position [1].

II. LITERATURE REVIEW

[2] presented a study of five (5) high speed binary multipliers: Booth Multiplier, Modified Booth Multiplier, Vedic Multiplier, Wallace Multiplier and Dadda Multiplier. The Booth Multiplier considers a set of two bits starting from the LSB of the multiplier and uses it to perform operations to generate the partial products. The number of partial products generated is equivalent to the number of bits of the multiplier, *n*. As the number of bits of the multiplier increases the cost increases, and its speed worsens as more partial products are involved. In the Modified Booth multiplier n/2 partial products are generated if *n* is even and (n+1)/2 if odd. The generation of partial products and the corresponding computation of sums are done in parallel. The partial products are obtained as presented in [3]. The Wallace multiplier operates using two: first the numbers are converted to binary after which the partial products are generated. Thereafter the matrix produced

DOI- 10.18486/ijcsnt.2018.7.2.03
ISSN-2053-6283

is compressed to 6, 4, 3 and 2 rows using (2,2) or (3,2) counters [3]. The Dadda multiplier operates with two stages: first the formation of the partial product matrix, then secondly the matrix must be broken down into rows which are added using carry-propagating adders. The advantage of the Dadda multiplier is that it does the minimum reduction required at each level [3].

According to [2] the Modified Booth multiplier reduces the number of partial products generated compared to other multipliers while the Dadda multiplier minimizes the number of adders used when compared to the Wallace multiplier. [2] therefore proposed a new multiplier architecture called the Booth Dadda Algorithm which combined the benefits of the Modified Booth Multiplier and Dadda Multiplier. As such [2] indicated that this proposed architecture will reduce the hardware utilization because of the reduction of the number of adders used, and also increased its speed because of the reduction in the number of partial products formed.

[4] presented an efficient method for partial product reduction for binary multiplier. This system was designed for the 16nm TSMH CMOS technology and was done using the Tanner EDA 14.1 development tool. [4] also presented a study of several partial product techniques such as Wallace and Dadda schemes. According to [4] the Dadda multiplier performed less reductions than the Wallace multiplier. [4] also claims that the Dadda multiplier consumed less power and area than the Wallace multiplier. [4] also presented several compressors, eg. 4 to 2 compressor which introduced a horizontal path as a result of limited propagation of the carry of the multiplier unit. [4] produced a gate level redesign of this compressor for maximizing performance. Two operating modes were considered: active mode and sleep mode. [4] examined 3 to 2, 4 to 2, 5 to 2 and 7 to 2 compressors and their performance. According to [4] the compressors with sleep transistors consumed on average 47.35% less power than the same architecture of compressor without sleep transistors. [4] also claims that the compressors with sleep transistors have less delay than the same architecture containing compressors without sleep transistors.

[5] proposed an 8x8 hybrid tree multiplier system by combination of the Dadda and Wallace strategies. The system was implemented on the DSCH2 tool and simulated on MICROWIND with 0.25um technology. [5] indicated that the conventional 8x8 Dadda multiplier executes more addition operations and therefore overheads due to wiring are greater. The decomposition logic type Dadda multiplier has partial products which are divided into four (4) parts and partial product addition (PPA) reduction is performed on each part and these results in the reduction of the path delay [5]. The proposed approach includes the assignment of the name group1, group2, group3 and group4 to the four decomposition blocks and each group is assigned either a 4x4 Dadda or 4x4 Wallace algorithms to be used for compressing the partial products. The preliminary results of the [5] indicate a 40% reduction in power (via analysis of Power Delay Product PDP) was achieved for the proposed system over existing 8x8 Dadda, Wallace, Decomposed Dadda and Partitioned-type multiplier without compressors.

[6] presented a high speed multiplier system which was based on Vedic mathematics. [6] also does a comparison of the implemented multiplier with the conventional binary multiplier in 8-bit, 16-bit and 32-bit modes. The multipliers were designed and implemented using VHDL for the target device Spartan 3 XC3S50a-4tq144 using Xilinx 14.7 ISE.

[7] presented a comparison of 32-bit Vedic multiplication with the conventional binary multiplier. In [7] both systems were implemented on Xilinx Nexys 3 Spartan 6 FPGA using Xilinx ISE 13.4.

[8] presented the implementation of a new and efficient reduction scheme for implementation of tree multipliers on FPGAs. The system implemented was not a binary multiplier system but rather a reduction scheme for partial product reduction. [8] proposed using a library of *m:n* counters of varying sizes in order to maximize the partial product reduction operation of the system, hence reducing the number of reduction steps hence minimizing latency and hardware utilization of the multiplier. The 32-bit multiplier scheme was implemented in Verilog on Xilinx ISE suite and targeted the Xilinx Spartan-6 platform.

[9] presented the implementation of a low power, high speed 16-bit binary multiplier using Vedic mathematics. The design started with the construction of a 2x2 multiplier block which is used in the construction of a 4x4 multiplier block, after which a 8x8 multiplier block is constructed. The required 16x16 multiplier block was constructed using the 8x8 multiplier blocks. [9] reported that the system had a path delay of 27.15 ns and utilized 14,382 transistors in its implementation.

[10] presented the implementation of a high speed, area efficient 16-bit Vedic Multiplier and 32-bit Booth Recoded Wallace Tree multiplier for use in implementation of arithmetic circuits. The system was implemented in Verilog HDL and synthesized for Xilinx Virtex 6 FPGA device. [10] reported that the multiplier systems implemented had path delays of 13.45ns and 11.57ns respectively. The hardware utilization was not stated. [11] presented the design of a 24-bit binary multiplier for use in the implementation of a 32-bit floating point multiplier. Vedic Mathematics was utilized in the implementation.

[12] proposed an efficient strategy for unsigned binary multiplication which was expected to improve the implementation in terms of path delay and area. [12] utilized a combination of Karatsuba algorithm and Urdhva-Tiryagbhyam algorithm in implementing the required system. The Karatsuba algorithm was implemented such that the two inputs were multiplied using vertical and crosswise multiplication method, the partial products are generated and summed up. The Urdhva-Tiryagbhyam algorithm on the other hand is best suited for multiplication of large numbers and the strategy is a divide and conquer one in which the numbers are divided into their most significant and least significant half after which multiplication is performed. The system implemented in [12] was implemented using Verilog HDL using a target Spartan-3E and Virtex-4 FPGA. 8-bit, 16-bit, 24-bit and 32-bit versions of the multiplier were implemented. The delay of each was obtained and compared with existing systems of same bit sizes. [12] indicated that the proposed 8-bit, 16-bit

DOI- 10.18486/ijcsnt.2018.7.2.03
ISSN-2053-6283

and 24-bit versions outperform their counterparts when it came to path delay while the 32-bit did not perform better than its 32-bit counterparts.

[13] designed an area-efficient multiplier using modified carry select adders (CSLAs) based on crosswise and vertical Vedic multiplier algorithms. The conventional BEC-based CSLAs utilized one ripple carry adder (RCA) and one binary to excess one converter (BEC) instead of dual ripple carry adders (RCAs) in its implementation. The modified CSLA consisted of three stages – half sum generation, final sum generation and carry generation [13]. [13] claimed that the 8-bit modified CSLA has shorter latency than the conventional 8-bit Vedic multiplier. This modified CSLA was then used in implementation of proposed 8-bit Vedic Multiplier. [13] reported that the path delay of the proposed Vedic Multiplier was 45.68ns while the hardware utilization was 1380 gates.

[14] presented the design of a high speed 32-bit multiplier architecture based on Vedic mathematics. [14] implemented this system by adjustment of the partial products using concatenation approach. The partial products are also added using carry-save adders instead of two adders at each stage of partial product reduction. The system of [14] was implemented on the Xilinx Spartan-3E device XC3S500e-fg320-5. [14] reported that the 8-bit, 16-bit and 32-bit Vedic multiplier implementations had path delays of 13.43ns, 17.62ns and 22.47ns respectively.

[15] presented area efficient 8x8 and 16x16 multiplier systems using Vedic mathematics to improve performance. The system was implemented in Verilog HDL and synthesized on Xilinx ISE 12.2 for the target device Spartan 3E, XC3S500-5FG320. The systems were implemented with BEC adders [15].

III. Contribution of the Research

Most of the multiplier systems reviewed in this paper carried out the processes of partial product generation, partial product storage and partial product reduction. For example, multipliers developed in [3], [2] and [4] perform partial product reduction using Wallace or Dadda multipliers, thereafter the results are compressed using compressors. Others like [5] use a combination of multiplier and compressor techniques to perform the partial product reduction segment.

Most of the existing systems reviewed utilized Vedic mathematics for partial product generation. Multiplier systems in [14], [13] and [16] for instance developed Vedic multipliers by utilizing smaller multipliers as building blocks to developing bigger multipliers. For instance the construction of a 2x2 multiplier block which is used in the construction of a 4x4 multiplier block, after which a 8x8 multiplier block is constructed.

Some multiplier systems such as that in [10] concurrently added the partial products during the multiplication operation, hence reducing the delay at the expense of hardware utilization. Others like [17] added the partial products as they were generated to reduce demands for memory for storage of partial products. Some systems such as in [18] broke up the input mantissa bits into ten (10) parts, partial products were generated, after which partial products were summed together and reduced.

Others like [19] proposed a technique for low power operation which utilized both Sleep and BIVOS techniques. When starting from the columns of least significance, some columns are switched to sleeping mode while the remaining is supplied with a biased voltage. This method resulted in a loss in accuracy.

There is no implementation of the binary multiplier that can be utilized for all four floating point multiplier modes – single, double, quadruple and octuple precisions modes. Only a few such as the multiplier of [20] and others catered for multi-precision and at best processed 2 batches of input single precision floating point numbers or 1 in double precision mode.

Some systems such as the multiplier architecture of [21] split the mantissa multiplier into the upper and lower components, and predicted the, sticky bit, carry bit, and mantissa product from the upper part. In the event that the prediction was correct the computation of the lower part was disabled and the rounding operation was simplified, hence allowing the system to consume less power.

Finally none of the existing binary multiplication systems analyzed past multiplication operations to further reduce latency of the multiplication operation. Focusing on previous multiplication operations could benefit future multiplications, hence preventing the system from having to undergo lengthy partial product generation operations especially in the case of quadruple and octuple precision modes where the number of partial products can become very large. The contribution of this paper will be the development of a novel 24-bit binary multiplier architecture for single precision floating point multiplier which does not utilize partial product storage and reduction process, hence reducing latency, path delay and hardware utilization of the system. The system will have shorter path delay than all existing implementations of 24-bit binary multipliers. This contribution will likely be extremely useful to arithmetic operations in digital and computer systems presently and in the future.

IV. Design of Novel 24-bit Binary Multiplier Architecture

The novel 24-bit binary multiplier architecture was designed using the structured design approach of [22]. The proposed architecture can be configured to operate in at 1-bit to 24-bit mode. The system also performs most operations only when a non-zero bit of the multiplier input A. This implementation's aim was to eliminate the partial product storage and reduction components. The multiplier system of this section aimed at eliminating the entire partial product reduction component by accumulating the partial products as they were computed. This decision was expected to reduce the path delay and hardware utilization when compared to the system.

The datapath design (Fig. 1.) of the modified multiplier system consisted of nine (9) blocks. Block 2A and 2B are hold-shift registers which are used to hold the multiplicand and multiplier respectively. In this multiplier system the

DOI- 10.18486/ijcsnt.2018.7.2.03
ISSN-2053-6283

multiplicand will be shifted left by Block 2A while Block 1 is an up-counter which will keep track of the number of bit-shifts made.

At the same time Block 2B which held the multiplier is shifted right and the least significant bit was analyzed with the aid of a comparator (Block 3) to determine if its non-zero. If it's non-zero then the shifted multiplicand from block 2A is stored in the hold register of block 4B. At the same time the previously accumulated sum from the 48-bit binary adder (Block 5) is stored in the hold register of block 4A. The sum of the new partial product generated by the hold-shift register Block 2A and stored in the hold register of Block 4B, and the previously accumulated sum of Block 5 which was stored in the hold register of Block 4A was computed by the 48-bit binary adder (Block 5). If on the other hand the least significant bit of the multiplier was zero then the up-counter (Block 1) was still incremented and the shifting and checking operations are repeated. This process is done when all multiplier bits were examined.

If either the multiplicand or multiplier were zero (0) the product was set to zero. If the multiplier was one (1) then the product would have been set to the value of the multiplicand. The three possible cases for product are input to a 3-line to 1-line multiplexer (Block 6) and selected depending on which situation arose. The selected result was then stored in a hold register (Block 7).

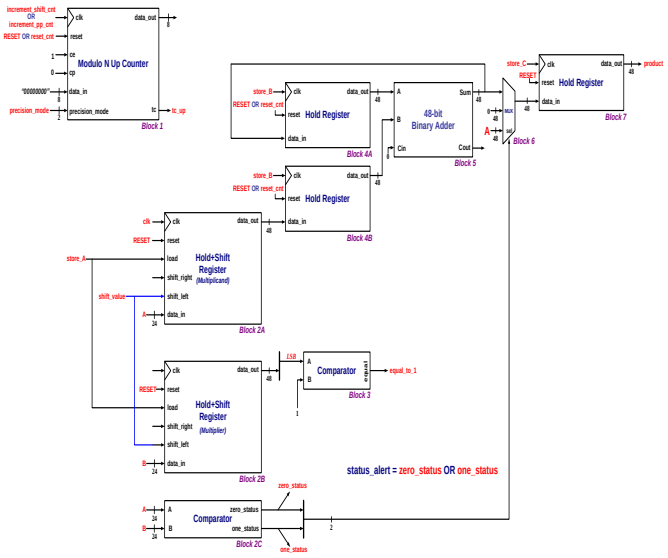

Fig. 1. Datapath Design for Novel 24-bit Binary Multiplier for Single Precision Floating Point Multiplication

The datapath interface definition for the system was constructed by simply extracting the inputs and outputs of the datapath design, clearly indicating which ports were connected to the control path. The control interface definition was constructed similarly. The FSM-D model (Fig. 2.) was developed by interfacing the datapath interface definition to the control interface definition.

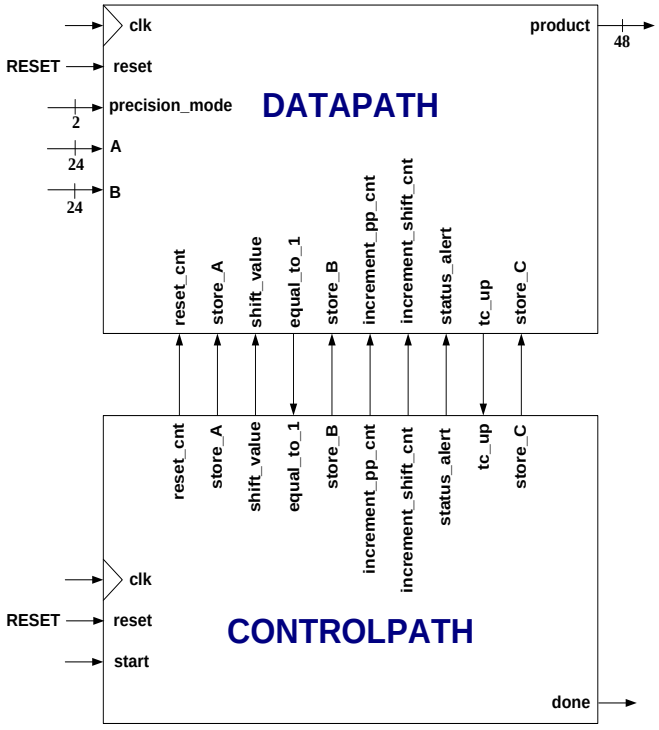

Fig. 2. FSM-D Model for Novel 24-bit Binary Multiplier for Single Precision Floating Point Multiplication

The operation of the datapath was then reviewed to ensure correctness and then the state transition table of the system was developed to meet the functional and timing requirements of the datapath. The state transition table was then used in the construction of the state diagram for the controlpath.

V. Hardware Implementation of the of Novel 24-bit Novel Binary Multiplier Architecture

The hardware implementation of the novel 24-bit binary multiplier for single precision floating point multiplier was entirely using VHDL in the Xilinx ISE Design Suite 14.7 The system consisted of several sub-modules and as such a structural approach was used in the implementation of the system. Sub-modules were port-mapped together using knowledge gained in [23] and [24] to implement the datapath. The controlpath was implemented as a finite state machine using knowledge gained from knowledge gained in [23] and [24]. Both datapath and controlpath were then port-mapped together to produce the overall system. After implementation they were synthesized in the Xilinx ISE Design Suite 14.7 in preparation for verification and validation stages.

DOI- 10.18486/ijcsnt.2018.7.2.03
ISSN-2053-6283

VI. Verification of Novel 24-bit Binary Multiplier Architecture

Timing simulation was performed on all components of the multiplier system to determine if they were operating as expected. Simulation was done using ISim simulator on Xilinx ISE Design Suite 14.7. The novel 24-bit binary multiplier architecture for single-precision floating point multiplier was verified using a wide range of input multiplicand and multiplier values.

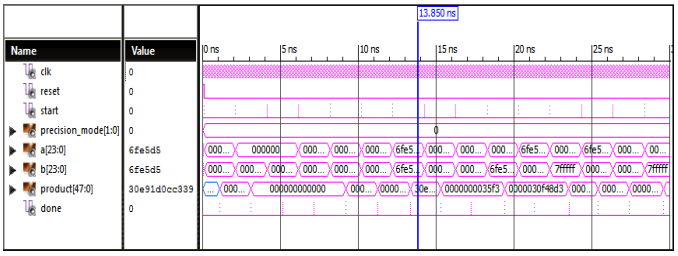

Fig.3. I Sim Simulation for Novel 24-Bit Binary Multiplier Architecture

The actual outputs were compared to the expected outputs to verify that the novel 24-bit binary multiplier architecture in 8-bit, 16-bit and 24-bit modes correctly multiplied the inputs. The comparison confirmed that the actual outputs of the novel 24-bit binary multiplier architecture in 8-bit, 16-bit and 24-bit modes corresponded to their expected results.

VII. Validation of Novel 24-bit Binary Multiplier Architecture

The novel 24-bit binary multiplier in 24-bit mode was simulated using ISim and path delay was extracted from the simulation screen. The hardware utilization was obtained using the synthesis report from Xilinx ISE Design Suite 14.7. The path delay of the system for several FPGA targets was determined via Post Place and Route Static Timing Report. Xilinx ISE Design Suite 10.1 was utilized for obtaining the post place and route static timing report for the Virtex 2 FPGA target because that target was not supported by Xilinx ISE Design Suite 14.7. The novel 24-bit binary multiplier was analyzed and performance parameters extracted (see Table I). The path delay of the novel multiplier system increased as the number of non-zero bits of the multiplicand and multiplier inputs A and B increased.

TABLE I

Performance Summary Of Novel 24-Bit Binary Multiplier Architecture For Various FPGA Targets

Bit Mode	Path Delay / ns
8 16 24	*Spartan 3 (XC3S1000-4FG256):* 8.023ns (1.873ns logic, 6.150ns route) *Spartan 3A (XC350A-4TQ144):* 7.114ns (2.197ns logic, 4.917ns route) *Spartan 3E (XC3S100E-5TQ144):* 6.563ns (1.109ns logic, 5.454ns route) *Spartan 6 (XC6SLX4-3TQG144):* 4.175ns (0.925ns logic, 3.250ns route) *Virtex 2 (XC2V1000-5FG456C):* 4.108ns (0.879ns logic, 3.229ns route) *Virtex 4 (XC4VFX12-10FF668):* 4.829ns (0.822ns logic, 4.007ns route) *Virtex 5 (XC5VFX30T-2FF665):* 2.780ns (0.424ns logic, 2.356ns route) *Virtex 6 (XC6VLX75T-3FF484):* 2.300ns (0.500ns logic, 1.800ns route)

It must be noted that only a few authors of documentation for existing 8-bit, 16-bit and 24-bit binary multiplier systems explicitly stated that the figures stated for delays of their multiplier systems was actually maximum path delays obtained after post place and route static timing analysis (consisting of logic delays, routing delays and clock skew), and not path delay obtained from the synthesis report (less accurate). Many of them indicated that their path delays were obtained after synthesis and as a result the delays stated may not include routing delays and clock skew.

The delays stated for the novel 24-bit binary multiplier implemented in this paper are maximum path delays after post place and route static timing analysis. In this paper it was assumed that the authors of documentation for existing implementations of 8-bit, 16-bit and 24-bit binary multiplier systems provided maximum path delay after post place and route static timing analysis. The novel 24-bit binary multiplier implemented in this paper still had shorter delays than these existing multiplier systems. However if the delays stated by authors of existing multiplier systems were indeed excluding routing delays and clock skew, it would mean that the novel 24-bit binary multiplier implemented in this paper performed much better than its existing counterparts.

TABLE II

Performance Comparison Between Novel 24-Bit Binary Multiplier Architecture And Various 8-Bit Binary Multipliers Reviewed

Source	Multiplier	Path Delay / ns
This Paper	Novel 24-bit Binary Multiplier Architecture	See Table I
[5] – *DSCH2 tool*	Regular Dadda Multiplier	4.40
	Decomposed Dadda Multiplier	4.10
	Partitioned Dadda Multiplier	5.50
	Wallace Tree Multiplier based on 3:2, 4:2 & 5:2 compressor	9.40
	Dadda Multiplier based on Higher Order Compressors	6.40

DOI- 10.18486/ijcsnt.2018.7.2.03
ISSN-2053-6283

	Proposed Hybrid Multiplier Combination (Dadda/Wallace/Wallace/Dadda)	7.50
[6] – *Spartan 3 XC350A-4TQ144*	Conventional Multiplier	11.00
	Urdhava Vedic Multiplier	5.50
	Nikhilam Sutra Vedic Multiplier	6.25
[12] – *Virtex 4 (XC4VFX12-10FF668)*	Vedic Multiplier	9.40
[13] – *Spartan 3 (XC3S1000-4FG256)*	Vedic Multiplier	45.68
[14] – *Spartan 3E (XC3S100E-5TQ144)*	Vedic Multiplier	13.43
[15] – *Spartan 3 (XC3S500-5FG320)*	Vedic Multiplier	23.18

TABLE III
PERFORMANCE COMPARISON BETWEEN NOVEL 24-BIT BINARY MULTIPLIER ARCHITECTURE AND VARIOUS 16-BIT BINARY MULTIPLIERS REVIEWED

Source	Multiplier	Path Delay / ns
This Paper	Novel 24-bit Binary Multiplier Architecture	See Table I
[9]	Vedic Multiplier	27.15
[10] – *Virtex 6 (XC6VLX75T-3FF484)*	Vedic Multiplier	13.45
[6] – *Spartan 3 (XC3S1000-4FG256)*	Conventional Multiplier	11.00
	Urdhava Vedic Multiplier	6.00
	Nikhilam Sutra Vedic Multiplier	6.00
[12] – *Virtex 4 (XC4VFX12-10FF668)*	Vedic Multiplier	11.51
[14] – *Spartan 3E (XC3S100E-5TQ144)*	Vedic Multiplier	17.62
[15] – *Spartan 3 (XC3S500-5FG320)*	Vedic Multiplier	38.82

TABLE. IV
PERFORMANCE COMPARISON BETWEEN NOVEL 24-BIT BINARY MULTIPLIER ARCHITECTURE AND VARIOUS 24-BIT BINARY MULTIPLIERS REVIEWED

Source	Multiplier	Path Delay / ns
This Paper	Novel 24-bit Binary Multiplier Architecture	See Table I
[11]	Vedic Multiplier	16.32
[12] – *Virtex 4 (XC4VFX12-10FF668)*	Vedic Multiplier	13.00

VIII. CONCLUSION

This paper presented the design, implementation, verification and validation of a novel 24-bit binary multiplier architecture for use in implementation of a single precision floating point multiplier. Most existing multiplier systems conducted their multiplier operations using the partial product generation, storage and reduction processes. This chapter produced an alternative system which excluded the partial product storage and reduction processes, hence reducing the latency of the system and hardware utilization in the process. The system also performed most operations only when a non-zero bit of the multiplier is found and as such the more non-zero bits found, the greater the number of cycles required to complete the multiplication process. At the worst case scenario the novel 24-bit binary multiplier architecture developed has shorter path delay than existing systems reviewed.

REFERENCES

[1] Sunesh, N.V and P Sathishkumar. 2015. Design and implementation of fast floating point multiplier unit. 2015 International Conference on VLSI Systems, Architecture, Technology and Applications (VLSI-SATA). pp 1 - 5, DOI: 10.1109/VLSI-SATA.2015.7050478

[2] Abraham, Sumod, Sukhmeet Kaur and Shivani Singh. 2015. Study of Various High Speed Multipliers. 2015 International Conference on Computer Communication and Informatics (ICCCI). pp 1 - 5, DOI: 10.1109/ICCCI.2015.7218139

[3] Kodali, Ravi Kishore, Lakshmi Boppana, Sai Sourabh Yenamachintala. 2015. FPGA implementation of vedic floating point multiplier. 2015 IEEE International Conference on Signal Processing, Informatics, Communication and Energy Systems (SPICES). Pp. 1 - 4, DOI: 10.1109/SPICES.2015.7091534

[4] Vyas, Keerti, Ginni Jain, Vijendra K. Maurya and Anu Mehra. 2015. Analysis of an Efficient Partial Product Reduction Technique. 2015 International Conference on Green Computing and Internet of Things (ICGCIoT). pp 1 - 6, DOI: 10.1109/ICGCIoT.2015.7380417

[5] Anitha, P., P. Ramanathan. 2014. A new hybrid Multiplieusing Dadda and Wallace Method. 2014 International Conference on Electronics and Communication Systems (ICECS). pp 1 - 4, DOI: 10.1109/ECS.2014.6892623

[6] Chopade, S.S. and Rama Mehta. 2015. Performance Analysis of Vedic Multiplication Technique using FPGA. 2015 IEEE Bombay Section Symposium (IBSS), September 2015, Mumbai, pp. 1 – 6. DOI: 10.1109/IBSS.2015.7456657

[7] Bisoyi, Abhyarthana,Mitu Baral, Manoja Kumar Senapati. 2014. Comparison of a 32-bit Vedic Multiplier with a Conventional Binary Multiplier. 2014 International Conference on Advanced Communication Control and Computing Technologies (ICACCCT). pp 1757 - 1760, DOI: 10.1109/ICACCCT.2014.7019410

[8] Mhaidat, Khaldoon M., Abdulmughni Y. Hamzah. 2014. A New Efficient Reduction Scheme to Implement Tree Multipliers on FPGAs. 2014 9th International Design and Test Symposium. pp 180-184, DOI: 10.1109/IDT.2014.7038609

[9] Bathija, R.K., R.S. Meena, S. Sarkar, Rajesh Sahu, "Low Power High Speed 16x16 bit Multiplier using Vedic Mathematics", International Journal of Computer Applications (0975 – 8887), Volume 59– No.6, pp. 41-44, December 2012

[10] Rao, Jagadeshwar M, Sanjay Dubey, "A High Speed and Area Efficient Booth Recoded Wallace Tree Multiplier for fast Arithmetic Circuits", 2012 Asia Pacific Conference on Postgraduate Research in Microelectronics & Electronics (PRIMEASIA), pp. 220-223, 2012.

[11] Jain, Anna, Baisakhy Dash, Ajit Kumar Panda, Muchharla Suresh. 2012. FPGA Design of a Fast 32-bit Floating Point Multiplier Unit. International Conference on Devices, Circuits and Systems (ICDCS), pp. 545-547.

[12] Su, Arish and R. K. Sharma. 2015. An Efficient Binary Multiplier Design for High Speed Applications using Karatsuba Algorithm and Urdhva-Tiryagbhyam Algorithm. 2015 Global Conference on Communication Technologies (GCCT), pp 192 - 196, DOI: 10.1109/GCCT.2015.7342650

[13] Gokhale, G. R., and P. D. Bahirgonde. 2015. Design of Vedic-Multiplier using Area-Efficient Carry Select Adder. 2015 International Conference on Advances in Computing, Communications and Informatics (ICACCI). pp. 576-581, DOI: 10.1109/ICACCI.2015.7275671

[14] Sharma, Richa, Manjit Kaur and Gurmohan Singh. 2015. Design and FPGA Implementation of Optimized 32-Bit Vedic Multiplier and Square Architectures. 2015 International Conference on Industrial Instrumentation and Control (ICIC) College of Engineering Pune, India. May 28-30, 2015. pp. 960-964, DOI: 10.1109/IIC.2015.7150883

[15] Ram, G. Challa, Y. Rama Lakshmanna, D. Sudha Rani and K.Bala Sindhuri. 2016. Area Efficient Modified Vedic Multiplier. 2016 Internbational Conference on Crircuit and Computing Technologies (ICCPCT), pp 276 - 279, DOI: 10.1109/ICCPCT.2016.7530294

[16] Thapliyal, Himanshu and M. B. Srinavas. 2005. A Novel Time-Area-Power Efficient Single Precision Floating Multiplier. Proceedings of MAPLD 2005, pp. 1 - 3, DOI: 10.1.1.97.1539

[17] Anane, N., H. Bessalah, M. Issad and M. Anane. 2009. Hardware Implementation of Variable Precision Multiplication on FPGA. 4th International Conference Design & Technology of Integrated Systems in Nanoscal Era, 2009 (DTIS '09). on pp 77-81. DOI: 10.1109/DTIS.2009.4938028

[18] Ramesh, Addanki Purna, A. V. N. Tilak and A. M. Prasad. 2013. An FPGA based high speed IEEE-754 double precision floating point multiplier using Verilog. 2013 International Conference on Emerging Trends in VLSI, Embedded System, Nano Electronics and Telecommunication System (ICEVENT), pp. 1 - 5, DOI: 10.1109/ICEVENT.2013.6496575.

[19] Gupta, Aman, Satyam Mandavalli, Vincent J. Mooney, Keck-Voon Ling, Arindam Basu, Henry Johan, Budianto Tandianus. 2011. Low Power Probabilistic Floating Point Multiplier Design. 2011 IEEE Computer Society Annual Symposium on VLSI. Volume 24 (3), pp. 182 - 187, DOI: 10.1109/ISVLSI.2011.54

[20] Havaldar, Soumya and K S Gurumurthy. 2016. Design of Vedic IEEE 754 floating point multiplier. 2016 IEEE International Conference on Recent Trends in Electronics, Information & Communication Technology (RTEICT). Volume 24 (3), pp 1131 - 1135, DOI: 10.1109/RTEICT.2016.7808008

[21] Kuang, Shiann-Rong, Jiun-Ping Wang, and Hua-Yi Hong. 2010. Variable-Latency Floating-Point Multipliers for Low-Power Applications. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, Vol 18 (10). pp. 1493-1497. DOI: 10.1109/TVLSI.2009.2025167

[22] George, Marcus and Geetam Singh Tomar. 2015. Hardware Design Procedure: Principles and Practices. 2015 Fifth International Conference on Communication Systems and Network Technologies. pp. 834 - 838.

[23] Institute of Electrical and Electronic Engineers (IEEE). 1993. IEEE Standard VHDL Language Reference Manual. IEEE 1076.3.

[24] G S Tomar, M L George, "Hardware Implementation of Pulse Code Modulation Speech Compression Algorithm", Asia-pacifc Journal of Multimedia Services Convergence with Art, Humanities and Sociology, Vol.2, No.1, pp.17-24, 2012.

Biography: Author

Marcus Lloyde George received the Bsc degree in Electrical and Computer Engineering from the University of the West Indies, St. Augustine in 2007, his MPhil degree in Electrical and Computer Engineering from the University of the West Indies, St. Augustine in 2011 and his PhD degree in Electrical and Computer Engineering from the University of the West Indies, St. Augustine in 2019. He is the author of several academic books. His research engineering interest include the business administration, strategic planning and management, engineering education, formal specification, modelling and verification, field programmable architectures, intelligent electronic instrumentation and biomedical engineering.

DOI- 10.18486/ijcsnt.2018.7.2.03
ISSN-2053-6283