NOVEL MANTISSA SIMILARITY INVESTIGATOR FOR PATH-DELAY REDUCTION OF PRODUCT MANTISSA CALCULATION

MARCUS LLOYDE GEORGE

Dept. Electrical and Computer Engineering, University of the West Indies, St. Augustine, Trinidad and Tobago
E-mail: marcus.george99@yahoo.com

Abstract - Floating point multiplication is a very important component of many engineering applications such as signal processing, video processing and image processing. In floating point multiplication, the mantissa calculation operation caters for the majority of time for the process. Because of this it is important to consider the speed up of the mantissa multiplication process in order to speed up systems that utilize floating point multiplication. This paper presents the development of a novel Mantissa Similarity Investigator (MSI) which can be interfaced to any product mantissa calculator to reduce the path delay of the multiplication operation. The system was synthesized for a variety of FPGA targets using Xilinx ISE Design Suite 14.7 Commercial Edition. The Mantissa Similarity Investigator (MSI) was interfaced to a Mantissa Calculator developed for this project, to form a complete novel MSI-Interfaced Mantissa Calculator. The path delay of this system was compared with existing implementations of 24-bit, 53-bit, 113-bit and 237-bit binary multipliers which represent mantissa multiplication at various precision levels. The novel MSI-Interfaced Mantissa Calculator achieved shorter path delay than its existing counterpaths reviewed.

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. The operation of multiplication also forms the basis of many other complex arithmetic operations such as cubing, squaring, convolution, etc.

According to [3] multiplication is the most elementary and most frequently used operation in ALUs. It allows one number to be scaled by another number. Floating-point multiplication is the arithmetic operation most frequently utilized and is a very important component of many engineering applications such as signal processing, video processing, image processing, etc. [12].

Floating-point format can represent very small and large numbers when compared to fixed-point numbers, therefore the dynamic range of numbers that can be represented is greater [3]. Many processes in science utilize floating-point arithmetic and therefore there is a need to develop units with shorter path delay, smaller hardware utilization and less power consumption [3]. Multiplication consumes significant delay compared to other arithmetic units used in basic mathematical computations [14]. As a result it is beneficial in the area of mathematical computation to present faster and more efficient mechanisms for implementing mathematical operations which also can utilize less power.

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.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. [6] indicated that the both the Urdhava and Nikhilam Sutra algorithms showed significant improvements in delay over the conventional binary multiplier at 8, 16 and 32-bit modes.

[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.

[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 of[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 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.

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 [2], [3] 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 an 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. 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. None of the existing binary multiplication systems analyzed past multiplication operations to further reduce path delay 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 path delay of existing binary multipliers are long and should be reduced to ensure that the systems they are utilized in have shorter path delays themselves.

The contribution of this paper will be the development of a novel Mantissa Similarity Investigator (MSI) which can be connected as the front end to any existing Product Mantissa Calculator for any floating point multiplier system to the reduce the path delay of the mantissa multiplication process. The MSI unit will result in the mantissa calculator having shorter path delay than all existing implementations of 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 MANTISSA SIMILARITY INVESTIGATOR

The Mantissa Similarity Investigator (MSI) (Fig. 1.) consisted of eight (8) inputs and three (3) outputs. The input clk is the periodic clock waveform to the system. When reset is asserted HIGH the entire system is initialized and the default state is enforced. precision_mode is used for selection of the floating-point precision mode (single, double, quadruple, octuple).

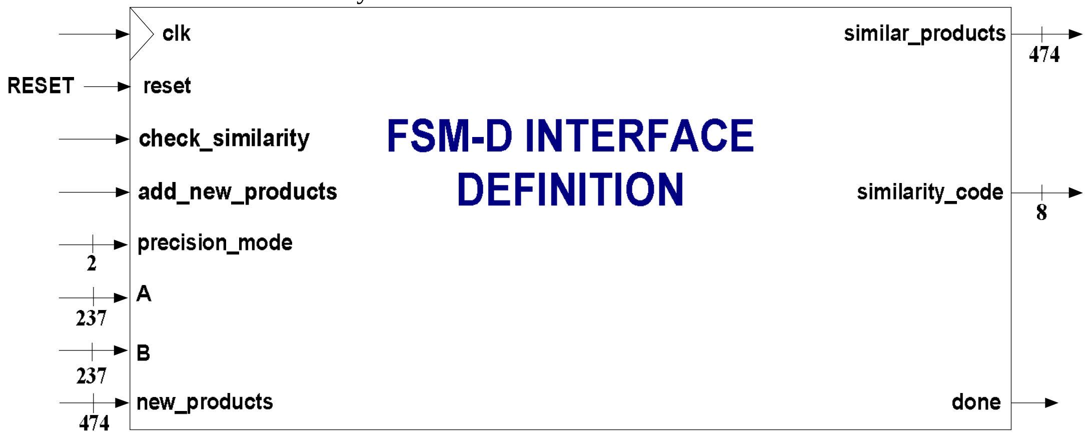

Fig. 1. FSM-D Interface Definition for Mantissa Similarity Investigator (MSI)

Before any similarity check (check_similarity) process is conducted, all inputs are analyzed to ensure that there are no redundancies. This provides several benefits to the system all of which reduce unnecessary interaction with the memory storage element of the MSI unit (Self-Ordering RAM block), hence possibly reducing the path delay. The first benefit stems from avoiding the system performing duplicate searches. For instance, if the requirement is to perform eight (8) single precision floating point multiplication operations and say two of the batches have the same inputs A (multiplicand) with B (multiplier), then the system performs only one similarity check and uses the result for both input batches. If all eight (8) input batches are identical, then only one similarity check will be done. This is expected to reduce the path delay. The second benefit stems from avoiding storage of duplicates. The instance the system discovers the occurrence of duplicate batches of inputs A (multiplicand) with B (multiplier), it makes a record of it in the form of a "input_similarity_code" which utilizes digits 0-7 to indicate how similar inputs are to each other. When adding new products, the system refers to this code to determine which new products should be added to the Self-Ordering RAM block. When check_similarity is asserted HIGH and add_new_products is asserted LOW the system utilizes inputs A (multiplicand) with B (multiplier) to determine if such inputs were multiplied in the past by conducting a search of the Self-Ordering RAM block. Before performing a search, the system checks for redundancies in the input A (multiplicand) with B (multiplier) batches to avoid unnecessarily searching the Self-Ordering RAM block. This process results in the production of an 8-bit "input_similar_code". Once this is done the similarity check process can proceed. If a match is found then the product corresponding to those inputs is assigned as a batch of the 474-bit similar_products output, and a logic 1 is assigned to the corresponding bit of the 8-bit similarity_code which generally indicates which of the batches (or segments) of similar_products holds a product computed in the past. When all batches of inputs are processed, the done signal will be asserted HIGH to indicate the end of the process in both cases (checking similarity and adding new products). When check_similarity is asserted LOW add_new_products is asserted HIGH the system adds the new products generated by the novel MSI-prepared binary multiplier system (found on the 474-bit port new_products) to the memory storage element of the MSI unit (Self-Ordering RAM block) for future similarity checks. Before adding new products, the system checks the "input_similarity_code" created prior to the similarity check process. Depending on this the system will know which of the products is expected to be a redundancy and which should not be added to the Self-Ordering RAM block. Once this is done then the system can proceed with the addition of new products.

To minimize the time taken for future similarity checks two considerations were taken into account in the design of the RAM block. First the RAMblock was divided into sectors. Each sector represents a portion of the range of numbers that can be represented at the selected precision level. Hence, prior to adding data, the MSI unit determines which of the sectors of the RAMblock the data must be added to. Therefore, when a similarity check for a particular input pair of A (multiplicand) with B (multiplier), the search is conducted on only that sector of the RAMblock. The second design consideration was that the RAMblock is made a Self-Ordering RAMblock, meaning as data is added to its sectors, all data in the sector will be arranged in ascending order. As such when a similarity check is done the search can begin within the vicinity of numbers in which the inputs A (multiplicand) with B (multiplier) are likely to be found. The datapath design of the mantissa similarity investigator comprised of sixteen (16) blocks. A data organizer (Block 1D) organizes inputs A (multiplicand) with B (multiplier) such that batches of multiplicand and multiplicands can be easily used for similarity checks or for adding new products. A Mantissa Calculator was developed for the purpose of validation of the effect of MSI on mantissa calculation. The Mantissa Calculator was then combined with the Mantissa Similarity Investigator (MSI) to form the complete modified novel MSI-Interfaced Mantissa Calculator which was expected to result in significant reduction in path delay in all precision modes. The datapath design (Fig. 2.) of the novel MSI-Interfaced Mantissa Calculator.

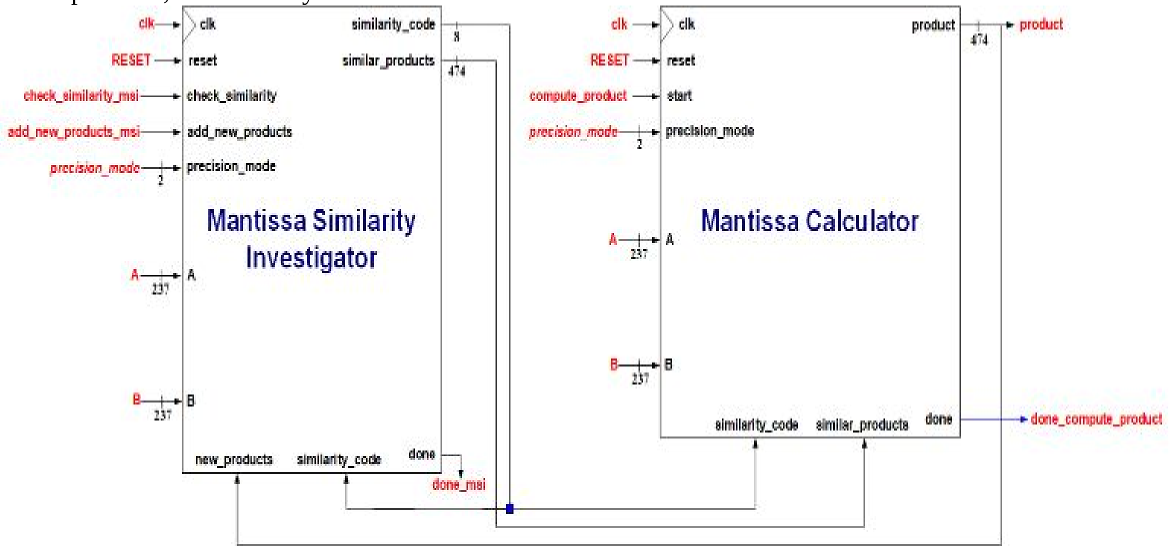

Fig. 2. Datapath Design for the Novel MSI-Interfaced Multi-Precision Binary Multiplier Architecture

V. HARDWARE IMPLEMENTATION OF THE OF NOVEL MANTISSA SIMILARITY INVESTIGATOR

The hardware implementation of the novel MSI-interfaced mantissa calculator was implemented 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 to implement the datapath using knowledge gained in [23] and [24]. After implementation they were synthesized in the Xilinx ISE Design Suite 14.7 in preparation for verification and validation stages.

VI. VERIFICATION OF NOVEL MANTISSA SIMILARITY INVESTIGATOR

The novel MSI-Interfaced Mantissa Calculator in single, double, quadruple and octuple precision modes was verified using a wide range of input multiplicand and multiplier values. 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 system was implemented for all four (4) precision modes to verify that it correctly multiplied multiplicand and multipliers for a range of values at all four precision modes.

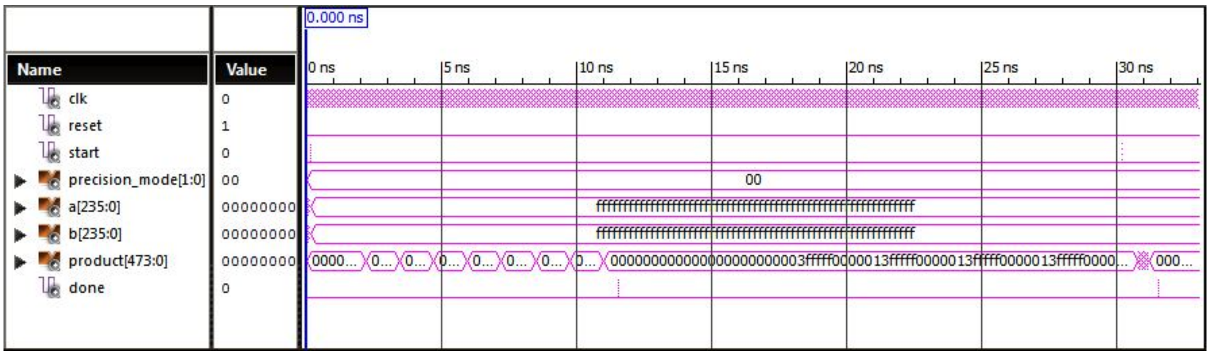
0.000 ns
Name
Value
0 ns
5 ns
10 ns
15 ns
20 ns
25 ns
30 ns
clk
reset
start
precision_mode[1:0]
a[235:0]
b[235:0]
product[473:0]
done
0
1
0
00
00000000
00000000
00000000
0
00
ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
0000...
0...
00000000000000000000000003fffff0000013fffff0000013fffff0000013fffff0000...
000...

Fig. 3. ISim Simulation for Novel MSI-Interfaced Mantissa Calculator

The actual outputs of the multiplier at all four precision modes were compared to the expected outputs to verify that they correctly multiplied the inputs. The comparison confirmed that all indicated that the multiplier's actual outputs for all four precision modes corresponded to their expected results.

VII. VALIDATION OF NOVEL MANTISSA SIMILARITY INVESTIGATOR

The path delay of the MSI-Interfaced Mantissa Calculator for several FPGA targets (Table I) 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. Table II presents the maximum path delay of the novel MSI-Interfaced Mantissa Calculator with and without MSI used.

TABLE I: FPGA PLATFORMS USED IN VALIDATION TESTING

FPGA Platform	Nomenclature
Spartan 3	XC3S1000-4FG256
Spartan 3A	XC350A-4TQ144
Spartan 3E	XC3S100E-5TQ144
Spartan 6	XC6SLX4-3TQG144
Virtex 2	XC2V1000-5FG456C
Virtex 4	XC4VFX12-10FF668
Virtex 5	XC5VFX30T-2FF665
Virtex 6	XC6VLX75T-3FF484

TABLE II: MAXIMUM PATH DELAY OF NOVEL MSI-INTERFACED MANTISSA CALCULATOR ARCHITECTURE WITH AND WITHOUT MSI

Precision Mode	Platform	Maximum Path Delay / ns	
		No MSI	MSI
Single (8 batches)	**Spartan 3**	.163	.841
	Spartan 3A	.087	.498
	Spartan 3E	.898	.672
	Spartan 6	.806	.385
	Virtex 2	.631	.136
	Virtex 4	.515	.108
	Virtex 5	.622	.896
	Virtex 6	.816	.347
Double (4 batches)	**Spartan 3**	.484	.476
	Spartan 6	.984	.180
	Virtex 2	.712	.689
	Virtex 4	.629	.610
	Virtex 5	.079	.674
	Virtex 6	.374	.377
Quadruple (2 batches)	**Spartan 3**	.274	.132
	Spartan 6	.546	.435
	Virtex 2	.681	.114
	Virtex 4	.856	.817
	Virtex 5	.189	.637
	Virtex 6	.153	.547
Octuple (1 batch)	**Spartan 3**	).612	.534
	Spartan 6	.608	.428
	Virtex 2	.488	.818

	Virtex 4	.441	.323
	Virtex 5	.566	.576
	Virtex 6	.269	.021

When the developed mantissa calculator is equipped with MSI, the system is arranged such that if it determines that present inputs were used in binary multiplication operations in the past (referred to as 'similar inputs'), the system would avoid performing the multiplication operation completely and just utilize the previously computed product as the result, hence reducing the path delay. The more input batches that are determined to be similar inputs, the shorter the path delay is expected to get.

To analyze the system performance in this case, the worst case and best case scenarios must have been considered. Once the MSI-interfaced mantissa calculator detects no similar inputs in the input batches, the system would not benefit from MSI at that instant, but is likely to benefit from MSI in the future if such inputs were determined to be similar for subsequent multiplication operations. The longest path delay for this system will correspond to the scenarios where all bits of input batches are non-zero. As such the inputs to the novel architecture were arranged such that all multiplicand and multiplier bits for all batches were non-zero. This was applied to all precision modes.

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 MSI-interfaced mantissa calculator 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 MSI-interfaced multi-precision 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 MSI-interfaced mantissa calculator implemented in this paper performed much better than its existing counterparts.

TABLE III: PERFORMANCE COMPARISON BETWEEN NOVEL 24-BIT BINARY MULTIPLIER ARCHITECTURE AND VARIOUS 8-BIT BINARY MULTIPLIERS REVIEWED

Source	Multiplier	Path Delay / ns
This Paper	Novel MSI-Interfaced Mantissa Calculator (Single Precision mode)	<u>See Table II</u>
[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
	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 IV: PERFORMANCE COMPARISON BETWEEN NOVEL 24-BIT BINARY MULTIPLIER ARCHITECTURE AND VARIOUS 16-BIT BINARY MULTIPLIERS REVIEWED

Source	Multiplier	Path Delay / ns
This Paper	Novel MSI-Interfaced Mantissa Calculator (Single Precision mode)	<u>See Tables II & III</u>
[9]	Vedic Multiplier	27.15
[10] – Virtex 6	Vedic Multiplier	13.45

(XC6VLX75T-3FF484)		
[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 V: PERFORMANCE COMPARISON BETWEEN NOVEL 24-BIT BINARY MULTIPLIER ARCHITECTURE AND VARIOUS 24-BIT BINARY MULTIPLIERS REVIEWED

Source	Multiplier	Path Delay / ns
This Paper	Novel MSI-Interfaced Mantissa Calculator (Single Precision mode)	See Table II
[11]	Vedic Multiplier	16.32
[12] – Virtex 4 (XC4VFX12-10FF668)	Vedic Multiplier	13.00

CONCLUSION

This paper presented the implementation, verification and validation of a novel Mantissa Similarity Investigator (MSI) capable of being attached to any product mantissa calculator to result in reduction in path delay of the mantissa multiplication process. Finally, the system developed without the use of MSI performs has shorter path delay than existing systems. However, the development of the MSI unit and the incorporation of it in the a novel MSI-interfaced mantissa calculator resulted in a system that has path delay that is significantly shorter than that of existing binary multiplier systems.MSI can be applied to other arithmetic units such as floating point dividers and square units to further reduce the path delays of their operations.

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] Perry, D. 1998. VHDL. 3rd ed. New York: McGraw-Hill.

★★★