Received: 2 June 2023 | *Revised:* 21 August 2023 | *Accepted:* 20 October 2023
DOI: 10.37917/ijeee.20.2.15 *Early View* | *December 2024*

Iraqi Journal for Electrical and Electronic Engineering
Original Article

Open Access
IJEEE University of Basrah College of Engineering

Design Efficient Vedic-Multiplier for Floating-Point MAC Module

Fatima Tariq Hussein*, Fatemah K. AL-Assfor
Computer Engineering Department, College of Engineering, University of Basrah, Basrah, Iraq

Correspondance
* Fatima Tariq Hussein
Department of Computer Engineering,
University of Basrah, Basrah, Iraq
Email: pgs.fatima.tariq@uobasrah.edu.iq

Abstract
Multiplication-accumulation (MAC) operation plays a crucial role in digital signal processing (DSP) applications, such as image convolution and filters, especially when performed on floating-point numbers to achieve high-level of accuracy. The performance of MAC module highly relies upon the performance of the multiplier utilized. This work offers a distinctive and efficient floating-point Vedic multiplier (VM) called adjusted-VM (AVM) to be utilized in MAC module to meet modern DSP demands. The proposed AVM is based on Urdhva-Tiryakbhyam-Sutra (UT-Sutra) approach and utilizes an enhanced design for the Brent-Kung carry-select adder (EBK-CSLA) to generate the final product. A (6*6)-bit AVM is designed first, then, it is extended to design (12*12)-bit AVM which in turns, utilized to design (24*24)-bit AVM. Moreover, the pipelining concept is used to optimize the speed of the offered (24*24)-bit AVM design. The proposed (24*24)-bit AVM can be used to achieve efficient multiplication for mantissa part in binary single-precision (BSP) floating-point MAC module. The proposed AVM architectures are modeled in VHDL, simulated, and synthesized by Xilinx-ISE14.7 tool using several FPGA families. The implementation results demonstrated a noticeable reduction in delay and area occupation by 33.16% and 42.42%, respectively compared with the most recent existing unpipelined design, and a reduction in delay of 44.78% compared with the existing pipelined design.

Keywords
Multiplier-Accumulator (MAC), Enhanced Brent-Kung Carry-Select Adder (EBK-CSLA), Adjusted Vedic multiplier (AVM), Carry Save Adder (CSA).

I. Introduction

Multiply-Accumulated (MAC) module is considered the most important module in digital signal processor (DSP), multimedia information processing, microprocessors systems and hence, it has a considerable effect on their performance in terms of speed and area occupancy [1]. MAC module can be employed for both fixed-point and floating-point computations. Floating-point arithmetic is widely utilized in DSP applications like, filters, correlation, convolution and image processing applications to attain higher accuracy. A MAC module is comprised of an (n*n) multiplier and a (2n+c) accumulated adder, with c symbolizes the extra bits to avert overflow case [2]. MAC performance mainly relies on its multiplier performance. Multiplication of two floating-point numbers can be accomplished using various multipliers and exploiting diverse algorithms. Vedic multiplier (VM) is considered as one of the most important multipliers for its good efficiency in terms of high speed [3–6]. The floating-point is a data structure utilized to exemplify real numbers in a binary form. The IEEE-754 Standard floating-point numbers encompasses two basics binary formats, namely single-precision (32-bit) and double precision (64-bit),that comprise arithmetic operations and round mechanisms [7]. IEEE-754 binary single-precision (BSP) and binary double-precision (BDP)formats are widely utilized in MAC modules. Fig. 1 illustrates the implementation of the BSP-floating-point operand (X) [8]. It

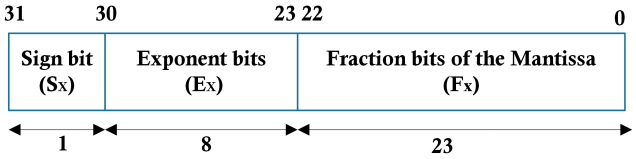

Fig. 1. The format of BSP floating-point number [8]

is consist of three filed: sign-bit (S_X), biased exponent (E_X), and the mantissa (or significand ($Mx = 1.F_X$) [9]. Where F_X is the fraction bits of the mantissa. These three fields are packed into a word such that.

$$X = (-1)^{S_X} \cdot M_X \cdot 2^{E_X} \quad (1)$$

Floating-point multiplier is used to carry out the mantissa multiplications of two floating-point operands [10]. Multiplication two (n-bit) numbers X and Y can be carried out by the following three steps [4] :
Step 1: generation of partial-products.
Step 2: reduction these partial-products using a set of adders, like (3:2) carry-save adders (CSA) (or simply CSA) to produce the intermediate product in sum and carry vectors form. The (3:2) in CSA denotes the number of inputs/outputs of the adder.
Step 3: generation the final product (final multiplication result) using fast adder. The inputs of this adder are the sum and carry vectors produced in step 2.
Generally, the multiplier speed essentially depends on the accumulation of the sum and carry vectors to produce the final multiplication result and the multiplication algorithm utilized. The goal of this work is to design high speeding and low area VM based on Urdhva-Tiryakbhyam-Sutra (UT-Sutra) approach for BSP floating-point MAC modules to achieve high-performance digital-signal processing. This goal may be accomplished throughout the steps bellow:
- Design an efficient adder to add the intermediate sum and carry vectors that generated from the CSA to produce the final multiplication result, since the speed of the multiplier is highly relied on the speed of that adder.
- Usage of the improved XOR gate in [11] to design the entire parts of the proposed multiplier, and
- Improve the speed of the proposed multiplier further using the pipelining concept.
Based on the above steps, this work presents a distinctive design for a (6*6)-bit VM called here adjusted-VM (simply, AVM). The design has utilized the conventional (3*3)-bit VM along with an enhanced design for the Brent-Kung carry-select adder (EBK-CSLA) in [11] to produce the final product result from the sum and carry vectors. The (6*6)-bit AVM circuit is in turn, utilized to design (12*12)-bit and then, a (24*24)-bit AVM which can be utilized to fulfill the mantissa multiplication for BSP floating-point numbers. The proposed (24*24)-bit AVM is then, optimized using the pipelining approach. The proposed AVM architectures are built using the improved XOR-gate to substantially improve the multiplier performance. The proposed AVMs are coded in VHDL, simulated in Xilinx 14.7 ISE software tool, and synthesized by different FPGA families, such as: Virtex-5, Virtex-6, Virtex-7, Zynq. and then, a complete analysis for their performances is provided.
This work is arranged as follows: Sec. II. reviews some of the previous works related to the floating-point multipliers and MAC modules. Sec. III. , explains the general design of a BSP floating-point MAC module. Section IV. affords details of the proposed AVM using an EBK-CSLA architecture, after which the details of the implementation results, simulations, and comparing the effectiveness of the proposed multiplier with the existent multiplier designs are offered in Sec. V. , and the conclusion is given in Section VI. .

II. Literature Review

In 2015, N. Jithendra et al [12] have presented two approaches to design MAC modules one to perform fixed-point signed numbers and the other to perform floating-point numbers. Their architectures were designed utilizing Wallace-tree multiplier and ripple carry adder (R-CA). Their multiplier and MAC designs had presented enhancement in terms of power, but gained higher delay and area consumption due to using the Wallce tree structure and due to the utilization of the R-CA which leads to high carry propagation delay during addition.

In 2016, authors in [13] had proposed a VM for floating-point operands. Their design had based on using three cascaded carry lookahead-adders (CLA-A)s to perform the partial-product reduction and the final addition to generate the final product. Nevertheless, their design had consumed higher area and had a considerable delay due to the carry propagation among the three adders.

In [14, 15], authors had designed (24*24)-bit Vedic based multipliers to perform mantissa multiplication for floating-point inputs. Their designs had comprised three cascaded levels of R-CAs to add the generated partial-products and to produce the final product. Their designs have achieved low speed due to using the R-CA which is considered the slowest adder among the adders.

G. Jha et al [16] had designed four kinds of multipliers to be used in MAC module, namely modified-booth, Wallace tree-reduction, add-shift, and combinational array multipliers and analyzed its performance when using these multipliers. However, none of these multipliers have introduced good performance in terms of power, delay and area occupation on the designed MAC. For example, the Wallce tree-reduction

based MAC had achieved better speed than the others, while the modified booth-encoding based MAC had the lowest area than other multipliers they used.

A. S. K. Vamsi et al [6] had proposed an (8*8)-bit VM using the UT-Sutra approach for (8*8)-bit MAC module (simply 16-bit MAC). Their design had utilized two cascaded CSAs to reduce the partial-products generated from the four (4*4)-bit VMs into two vectors: namely, the sum and carry vectors. However, their design is incomplete and inaccurate, as the design lacks the most important part which is the adder that must be used to generate the final multiplication result (namely, the final product). The inputs of that adder should be the sum and carry vectors generated from the cascaded CSAs of their design in order to produce the final multiplication result.

K. Thiruvenkadam and S. Saravanan [17] had used a modified full-adder (FA) to design array multiplier for BSP floating-point numbers. Their design has relied on Divide and Conquer (D-C) algorithm. The design had implemented using the pipelining concept. Although their design had some improvements in terms of power and area, but it had incurred more delay.

R. Sravani et al [18] have presented a BSP floating-point VM based on the Karatsuba algorithm. The multiplier is called carry-save VM and it consists of a top-level of 23 half-adders (HAs) followed by multilevel of FAs; each level consists of 23 FAs. The delay of their carry-save VM design is relatively high due to the carry propagation of the multilevel FAs.

Authors in [19, 20] have proposed multipliers using Karatsuba algorithm for fixed-point/floating-point MAC module. Their multipliers had involved three cascaded CSAs to reduce the generated partial-products into sum and carry vectors. To generate the final product, the authors had they used the Kogge-stone adder. However, their designed multiplier incurred delay and area occupation due to the use of three cascaded CSAs prior to the Kogge-stone adder.

III. Floating -Point Mac Module

Floating-point MAC module is desirable for higher accuracy and performance computations to perform the operation F as shown:

$$F = \sum X * Y \tag{2}$$

Where X and Y are inputs operands. It incorporates a floating-point adder, floating-point multiplier, and an accumulator-register. The performance of a DSP system depends substantially on the performance of its MAC module and precisely on the speed of multiplication processes within the MAC module [21, 22].

The backbone of all digital signal computations is the lies in floating-point arithmetic field [23–26]. BSP format of IEEE-754 standard is used to design (32*32)-bit floating-point MAC (64-bit MAC) module [7], as depicted in Fig. 2. Fig. 3 illustrates the floating-point MAC module internal organization. To perform a (32*32) floating-point multiplication, two 32-bit floating-point operands X and Y are used, each consists of three fields namely mantissa (M): $M = 1.F$, where F denotes to the fraction bits (23-bit) and the integer bit 1 is hidden), 8-bit exponent (E), and a sign bit (S) are used [27–32].

The multiplication of X and Y is accomplished using three operations performed in parallel as follows:

-Generate the sign of the product:

-The most significant bits (MSB)s of the two inputs X and Y are XORed together to produce the sign of the product.

-Compute the exponent (E) of the product: The exponent of X and Y (E_X,E_Y) are added using any adder and the bias is subtracted from the result, namely

$$E = E_X + E_Y - bias \tag{3}$$

(the bias=127 for single-precision) [23].

-Multiplication of the mantissas (M): A (24*24)-bit multiplier such as VM is utilized to perform mantissa multiplication as shown:

$$M = M_X * M_Y \tag{4}$$

where $M_X = 1.F_X$ and $M_Y = 1.F_Y$

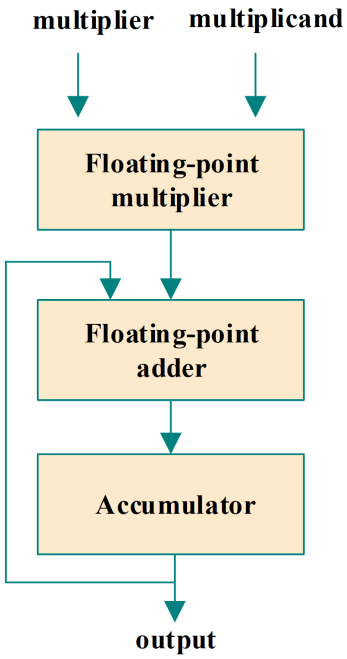

Fig. 2. Block scheme of floating-point MAC module

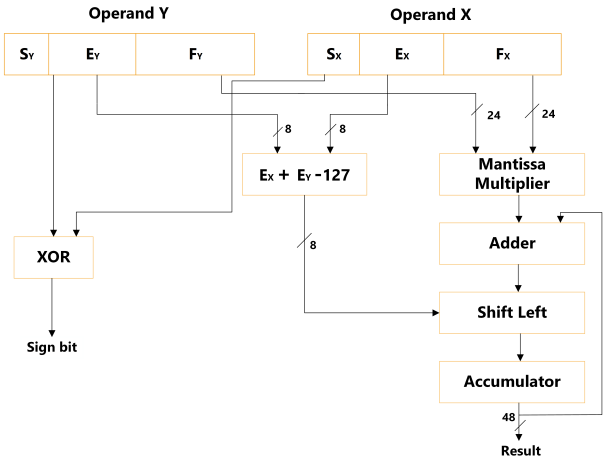

Fig. 3. Internal organization of Single-precision floating point MAC.

IV. Proposed Adjusted VM (AVM)

This section presents a distinctive design for a (24*24)-bit VM based on UT-Sutra approach called here adjusted VM (AVM) to be used for floating-point DSP's MAC module. The work starts by designing a (6*6)-bit AVM, then it is extended to design (12*12)-bit and (24*24)-bit AVMs accordingly.

*A. Proposed (6*6)-bit AVM.*

A (6*6)-bit AVM is designed using four (3*3)-bit conventional VMs, a single (6-bit) CSA to add three intermediate partial products and generate two (6-bit) vectors: the sum and carry vectors. An enhanced 6-bit Brent-Kung carry-select adder (EBK-CSLA) has proposed and be incorporated in the AVM design to add the resultant two vectors generated from the CSA for producing the final product as illustrated in Fig. 4. The 6-bit EBK-CSLA consists of two (3-bit) Brent-Kung (BK) adders, a 4-bit binary-to-access-1 convertors (BEC1), and a 3-bit MUX. The three MSBs of the final product can be obtained using a 3-bit increment-by-1 convertor (IB1C) instead of using an adder, for its high speed, since it consists of fewest logic gates in comparison to with any adder.

*B. Design of (24*24)-bit AVM*

The (6*6)-bit AVM can be easily extended to design a (12*12)-bit AVM. In this case, four (6*6)-bit AVM modules, a 12-bit CSA to reduce the intermediate partial products from three vectors to two (12-bit) vectors (namely, sum and carry), and then, an 11-bit EBK-CSLA is used to produce the final product bits of the intermediate stage, followed by a 6-bit EBK-CSLA which is used to generate the most significant 6- bits of the product, as demonstrated in Fig. 5.

Similarly, the (24*24)-bit AVM is designed from the (12*12)-bit AVM, as illustrated in Fig. 6. The proposed

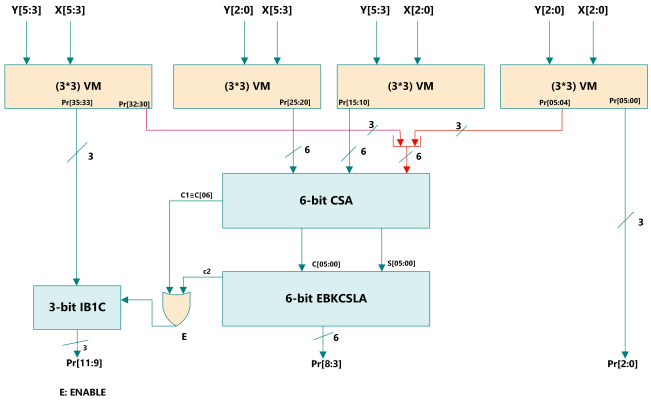

Fig. 4. The Proposed (6*6)-bit AVM.

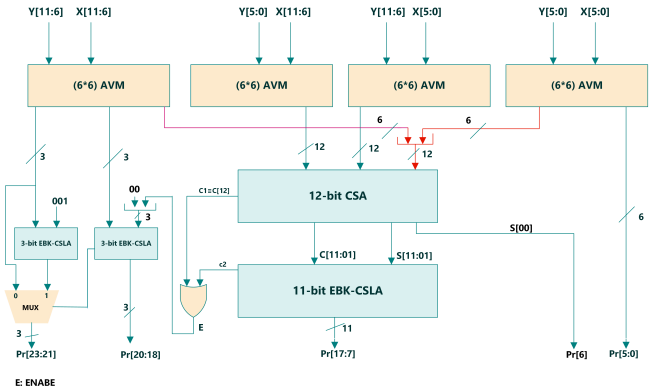

Fig. 5. The Proposed (12*12)-bit AVM.

(24*24)-bit AVM consists of four (12*12)-bit AVMs, followed by a single (24-bit) CSA to reduce the partial products generated from the (12*12)-bit AVMs to two vectors: the sum (S-vector) and the carry vector (C-vector). The S-vector consists of 24-bit (bits S[23:00]), while the carry vector (C-vector) consists of 24-bit (bits C[24:01]). The least significant bit (LSB) of the S-vector (namely; bit S[00]) represents the 12th bit of the final product (namely, S[00]≡Pr[12]). This means that one can minimize the size of the proposed EBK-CSLA which is used to generate the final product bits from 24-bit to 23-bit. This step will further improve the performance of the AVM design. The output of the (23-bit) EBK-CSLA represents the product bits Pr[35:13]. Fig. 7 depicts the design of the (23-bit) EBK-CSLA, it consists of six blocks of variable bit sizes BK-adders along with five BEC and a set of MUXs to carry out carry selection. To generate the most significant 12-bit of the final product (namely, bits Pr[47:36]), two (6-bit) EBK-CSLA modules are utilized. The two output-carry signals c1 and c2 generated from the CSA and the (23-bit) EBK-CSLA, respectively are OR-ed together to enable the first (6-bit) EBK-CSLA for generating the product bits Pr[41:36]. The output

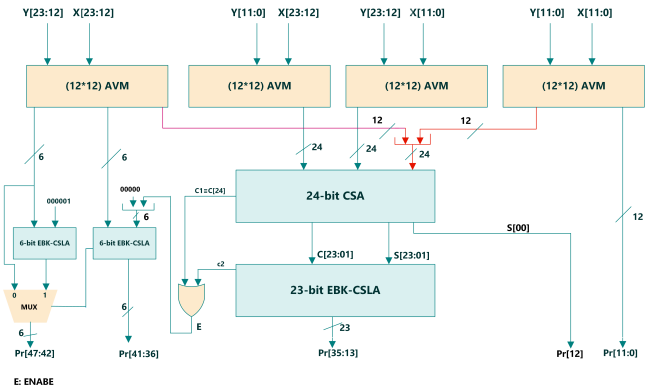

Fig. 6. The Proposed architecture of a (24*24)-bit AVM using fast 23-bit EBK-CSLA.

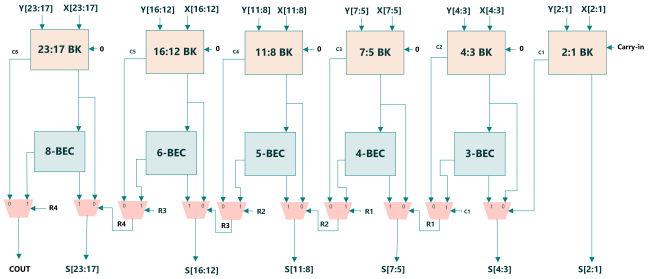

Fig. 7. Internal organization of the proposed 23-bit EBK-CSLA.

carry of this adder is used as a carry-in to the second (6-bit) EBK-CSLA to update the final product bits (bits Pr[47:42]) by adding the six most-significant bits of the partial-product with '000001' instead of utilizing IB1C circuit.

In this work, the improved XOR-gate is utilized in designing the AVM to improve its performance further. The improved XOR-gate architecture consists of three logic gates (namely, AND, NAND, and OR) instead of five logic gates. To optimize the speed of the AVM more, the pipeline concept is applied [23]. The pipeline stage can be applied either before the CSA or within the EBK-CSLA to generate the product bits from the multiplier faster. In this work, it is found that the (24*24)-bit AVM designed utilizing (3x3)-bit pipelined-VM gives the best performance results.

V. Results And Discussion

All the proposed multipliers in this work, namely, the (6*6)-bit, (b)-bit, and (24*24)-bit AVMs with/without pipelining are coded in VHDL and their performances in terms of the area occupation in FPGA and the delay (speed=1/delay) are assessed in Xilinx using four FPGA families: Virtex-5, Virtex-6, Virtex-7, and Zynq, as demonstrated in Table I. Note that the

TABLE I.
Performance Results Of Different Bit Size AVM Circuits

Parameters	FPGA Family	Proposed AVM			
		(6*6) -bit	(12*12) -bit	(24*24)-bit	
				unpipelined	piplined
No. of FPGA LUTs	Virtex-5	72	310	1260	568
	Virtex-6	72	306	1014	594
	Virtex-7	68	294	962	563
	Zynq	61	288	1014	558
Delay(ns)	Virtex-5	4.88	7.956	12.74	3.65
	Virtex-6	4.82	7.92	12.395	3.352
	Virtex-7	4.71	7.832	11.583	2.843
	Zynq	4.34	7.074	11.583	2.58

FPGA Virtex families are chosen for the comparison purpose with previous designs, while the Zynq family is utilized for its high features in achieving the best performance parameters which can be noticed clearly in Table I.

Fig. 8 demonstrates the register-transfer-level (RTL)-schematic of the synthesized (24*24)-bit AVM. It can be noticed that the design has utilized the following number of hardware modules: four modules of (12*12)-bit AVM, a CSA to carry out 24-bit addition, one (23-bit) EBK-CSLA, two (6-bit) EBK-CSLA, a (2:1) MUX, and an OR-gate.

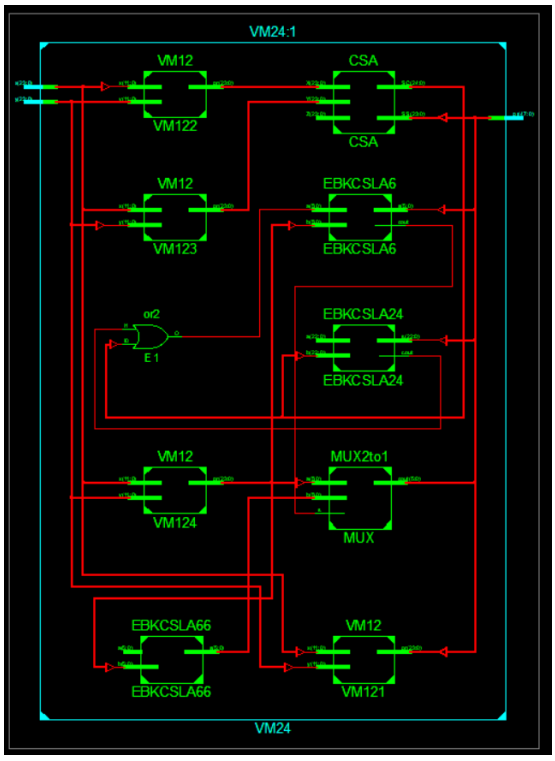

Fig. 8. RTL- scheme of the proposed (24*24)-bit AVM

TABLE II.
PERFORMANCE COMPARISON OF UNPIPELINED FLOATING-POINT (24*24)-BIT MULTIPLIERS

Ref.	FPGA family		Delay(ns)	No. of LUTs
[26]	Virtex-7	Design1	47.33	164
		Design2	28.02	2928
		Design3	27.76	1121
[3]	Virtex-6		21.823	-
[28]	Virtex-7		17.33	1763
Proposed	Virtex-5		12.74	1260
	Virtex-6		12.395	1018
	Virtex-7		11.583	1015
	Zynq		11.583	1014

TABLE III.
PERFORMANCE COMPARISON OF PIPELINED FLOATING-POINT (24*24)-BIT MULTIPLIERS

Ref.	FPGA family	Delay(ns)	No. of LUTs
[31]	Virtex-5	6.61	-
Proposed	Virtex-5	3.65	568
	Virtex-6	3.452	564
	Virtex-7	3.117	563
	Zynq	2.58	558

The internal organization RTL-scheme of the synthesized AVM with more details is depicted in Fig. 9.

The (24*24)-bit AVM is simulated to validate their functionality in multiplying mantissa parts of two floating-point operands. The functionality of the (24*24)-bit AVM is verified by providing several cases of inputs (the inputs are in decimal representation) to verify the corresponding outputs. For example, case1: 100*24 = 2400, case2: (570*320) =182400, and case3: (1320*23450) =30954000, etc. as illustrated in Fig. 10.

Tables II and III show a comparison between the proposed (24*24)-bit multiplier without/with pipelining with some existing multipliers. It is shown from Table II that the proposed unpipelined (24*24)-bit AVM has achieved reduction in delay and area utilization of 33.16 % and 42.42%, respectively than the multiplier offered one in [28] for the same FPGA family which is Virtex-7.

For pipelined design case, it can be noticed from table III that the proposed (24*24)-bit AVM has achieved less delay of 44.78% than the one proposed in [31] for the same FPGA family (virtex-5), and that the lowest delay and area occupation for the pipelined (24*24)-bit AVM are obtained when using the FPGA Zynq family. It is clear from Tables II and III that the proposed (24*24)-bit AVM yields less delay and achieves significant reduction in area utilization compared with the mentioned multipliers. The reduction in delay is due to the use of the EBK-CSLA along with a single CSA to eliminate any propagation for the carry during the partial product reduction step and the final addition step performed to generate the final product. Thus, the proposed (24*24)-bit AVM can be used to design an efficient floating-point MAC module to meet the requirements of cutting-edge DSP applications.

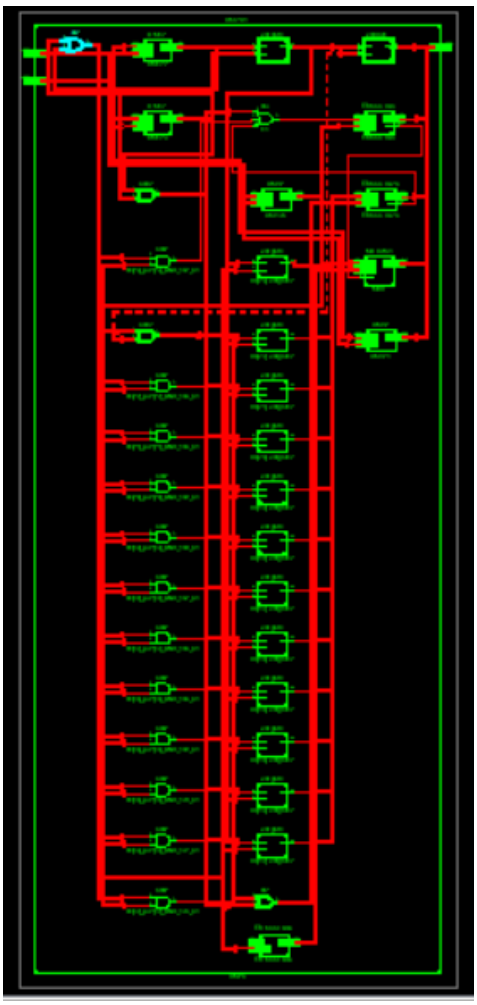

Fig. 9. Internal organization of the (24*24)-bit AVM scheme in RTL.

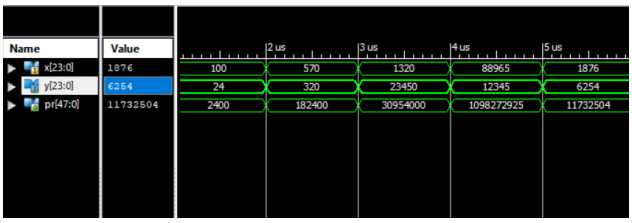

Fig. 10. Simulation input/output waveforms of (24*24)-bit AVM.

VI. Conclusion

In this work, a (24*24)-bit adjusted-Vedic multiplier (AVM) with high speediness and low area consumption is proposed and implemented. The utilization of an enhanced design for the Brent-Kung carry-select adder (EBK-CSLA) which used to perform the final addition has eliminated any propagation for the carry and thus, its highly reduced the delay of the multiplier. Moreover, the use of the improved XOR-gate to design the multiplier is extremely reduced the area occupation and the delay of the design. The proposed (24*24)-bit AVM offers efficient speed and area utilization for integration into fast floating-point MAC module of the DSP systems. The proposed AVM achieves reduction in delay and FPGA area occupation by 33.16% and 42.42%, respectively for the unpipelined design, and reduction in delay of 44.78% for pipelined design compared with the existing designs.

CONFLICT OF INTEREST

The author have no conflict of relevant interest to this article.

References

[1] K. Rajesh and G. U. Reddy, "Fpga implementation of multiplier-accumulator unit using vedic multiplier and reversible gates," in *2019 Third International Conference on Inventive Systems and Control (ICISC)*, pp. 467–471, IEEE, 2019.

[2] N. Nagaraju and S. Ramesh, "Implementation of high speed and area efficient mac unit for industrial applications," *Cluster Computing*, vol. 22, no. Suppl 2, pp. 4511–4517, 2019.

[3] S. Havaldar and K. Gurumurthy, "Design of vedic ieee 754 floating point multiplier," in *2016 IEEE International Conference on Recent Trends in Electronics, Information & Communication Technology (RTEICT)*, pp. 1131–1135, IEEE, 2016.

[4] K. Paldurai and K. Hariharan, "Fpga implementation of delay optimized single precision floating point multiplier," in *2015 International Conference on Advanced Computing and Communication Systems*, pp. 1–5, IEEE, 2015.

[5] M. B. Murugesh, S. Nagaraj, J. Jayasree, and G. V. K. Reddy, "Modified high speed 32-bit vedic multiplier design and implementation," in *2020 International Conference on Electronics and Sustainable Communication Systems (ICESC)*, pp. 929–932, IEEE, 2020.

[6] A. S. K. Vamsi and S. Ramesh, "An efficient design of 16 bit mac unit using vedic mathematics," in *2019 international conference on communication and signal processing (ICCSP)*, pp. 0319–0322, IEEE, 2019.

[7] K. J. Ramesh *et al.*, "A review: Multiply and accumulate architectures for digital signal processing and digital image processing," *Turkish Journal of Computer and Mathematics Education (TURCOMAT)*, vol. 12, no. 12, pp. 3797–3804, 2021.

[8] S. S. Mahakalkar and S. L. Haridas, "Design of high performance ieee754 floating point multiplier using vedic mathematics," in *2014 International Conference on Computational Intelligence and Communication Networks*, vol. 52, pp. 985–988, IEEE, 2014.

[9] G. Di Meo, G. Saggese, A. G. Strollo, D. De Caro, and N. Petra, "Approximate floating-point multiplier based on static segmentation," *Electronics*, vol. 11, no. 19, p. 3005, 2022.

[10] A. Kanhe, S. K. Das, and A. K. Singh, "Design and implementation of floating point multiplier based on vedic multiplication technique," in *2012 international conference on communication, information & computing technology (ICCICT)*, pp. 1–4, IEEE, 2012.

[11] F. K. Al Assfor, I. S. Al-Furati, and A. T. Rashed, "Vedic-based squarers with high performance," *Indonesian Journal of Electrical Engineering and Informatics (IJEEI)*, vol. 9, no. 1, pp. 163–172, 2021.

[12] N. J. Babu and R. Sarma, "A novel low power and high speed multiply-accumulate (mac) unit design for floating-point numbers," in *2015 International Conference on Smart Technologies and Management for Computing, Communication, Controls, Energy and Materials (ICSTM)*, pp. 411–417, IEEE, 2015.

[13] S. S. Mohite, S. S. Nimbalkar, and M. Makarand, "32 bit floating point vedic multiplier," *IOSR Journal of VLSI and Signal Processing (IOSR-JVSP)*, vol. 6, no. 2, pp. 16–20, 2016.

[14] M. A. Menon and R. Renjith, "Implementation of 24 bit high speed floating point vedic multiplier," in *2017 International Conference on Networks & Advances in Computational Technologies (NetACT)*, pp. 453–457, IEEE, 2017.

[15] S. Javeed and S. S. Patil, "Low power high speed 24 bit floating point vedic multiplier using cadence," *Int. Res. J. Eng. Tech.(IRJET)*, vol. 5, pp. 1771–1775, 2018.

[16] G. Jha and E. John, "Performance analysis of single-precision floating-point mac for deep learning," in *2018*

IEEE 61st International Midwest Symposium on Circuits and Systems (MWSCAS), pp. 885–888, IEEE, 2018.

[17] T. Krishnan and S. Saravanan, "Design of low-area and high speed pipelined single precision floating point multiplier," in *2020 6th International conference on advanced computing and communication systems (ICACCS)*, pp. 1259–1264, IEEE, 2020.

[18] R. Sravani and M. Salauddin, "Design of single precision floating point multiplier using fpga," *Journal of Engineering sciences*, vol. 12, no. 11, pp. 203–201, 2021.

[19] C. Mounica, S. Krishna, S. Veeramachaneni, and N. Mahammad S, "Efficient implementation of mixed-precision multiply-accumulator unit for ai algorithms," *International Journal of Circuit Theory and Applications*, vol. 48, no. 8, pp. 1386–1394, 2020.

[20] H. Zhang, D. Chen, and S.-B. Ko, "New flexible multiple-precision multiply-accumulate unit for deep neural network training and inference," *IEEE Transactions on Computers*, vol. 69, no. 1, pp. 26–38, 2019.

[21] S. Tamilselvan and A. Arun, "An efficient mac design for image processing application," *Indian Journal of Science and Technology*, vol. 11, no. 19, 2018.

[22] M. Yuvaraj, B. J. Kailath, and N. Bhaskhar, "Design of optimized mac unit using integrated vedic multiplier," in *2017 International conference on Microelectronic Devices, Circuits and Systems (ICMDCS)*, pp. 1–6, IEEE, 2017.

[23] S. Anjana, C. Pradeep, and P. Samuel, "Synthesize of high speed floating-point multipliers based on vedic mathematics," *Procedia Computer Science*, vol. 46, pp. 1294–1302, 2015.

[24] P. Singh and B. Kakani, "Performance comparison of floating point multipliers by using different multiplication algorithm," *International Journal of Electronics and Communication (IIJEC)*, vol. 3, no. 1, pp. 44–52, 2015.

[25] A. Eshack and S. Krishnakumar, "Pipelined vedic multiplier with manifold adder complexity levels," *International Journal of Electrical and Computer Engineering*, vol. 10, no. 3, pp. 2951–2958, 2020.

[26] R. K. Kodali, L. Boppana, and S. S. Yenamachintala, "Fpga implementation of vedic floating point multiplier," in *2015 IEEE international conference on signal processing, informatics, communication and energy systems (SPICES)*, pp. 1–4, IEEE, 2015.

[27] C. Hanuman and J. Kamala, "Hardware implementation of 24-bit vedic multiplier in 32-bit floating-point divider," in *2018 4th International Conference on Electrical, Electronics and System Engineering (ICEESE)*, pp. 60–64, IEEE, 2018.

[28] R. Deokate, N. Chore, and M. Thakre, "Design and performance evaluation of 32-bit floating point multiplier using vedic multiplier," vol. 7, no. 3, pp. 1783—-1786, 2021.

[29] A. Jain, B. Dash, A. K. Panda, and M. Suresh, "Fpga design of a fast 32-bit floating point multiplier unit," in *2012 International Conference on Devices, Circuits and Systems (ICDCS)*, pp. 545–547, IEEE, 2012.

[30] S. Arish and R. Sharma, "An efficient floating point multiplier design for high speed applications using karatsuba algorithm and urdhva-tiryagbhyam algorithm," in *2015 international conference on signal processing and communication (ICSC)*, pp. 303–308, IEEE, 2015.

[31] S. Mehta, B. Singh, and D. Kumar, "Performance analysis of floating point mac unit," *International Journal of Computer Applications*, vol. 78, no. 1, pp. 38—-41, 2013.

[32] S. V. P. GS, G. Seshikala, and S. Niranjana, "Design of efficient single precision floating point multiplier using urdhva triyagbhyam sutra of vedic mathematics," *Int. J. Innov. Technol. Exploring Eng.(IJITEE)*, vol. 8, no. 2, pp. 160—-162, 2019.