Comparative Review of Floating-Point Multiplier Systems

(IJCSNT) International Journal of Communication Systems and Network Technologies
ISSN Number: 2053-6283
Volume 8, Issue No. 2, 65–89

https://journals.mirlabs.in/index.php/ijcsnt
DOI-10.18486/ijcsnt/8.2.105

Marcus Lloyde George[1]

Abstract

This paper presents a comprehensive comparative review of existing floating point multiplier systems. The study focuses on single, double, quadruple and multi-precision floating point multiplier architectures and seeks to identify engineering techniques involved in their development. A comparison of the performance of these systems in terms of metrics such as path delay, hardware utilization and even power consumption in some cases have been carried out. Weaknesses in the systems reviewed along with possible gaps in the area of research have also been identified in this work. This paper also serves to identify several recommendations and considerations for the development of a multi-precision floating point multiplier system capable of treating with the weaknesses of multiplier systems identified.

Keywords

Arithmetic Logic Unit, Floating-point Multiplier, Multi-precision Floating Point, Multiplier System, Multiplier Architecture, FPGAs in Arithmetic

Received: 02 April 2019; **Revised:** 29 July 2019; **Accepted:** 23 August 2019;
Published: 31 August 2019;

[1]Department of Electrical and Computer Engineering, University of the West Indies, St. Augustine, Trinidad and Tobago.

Corresponding author:
Email: Marcus.George@sta.uwi.edu

Introduction

Arithmetic Logic Units (ALUs) are important components of processors that perform various arithmetic operations such as multiplication, division, addition, subtraction, cubing, squaring, etc.[1]. The ALU of some processors are divided into two units, an arithmetic unit (AU) and a logic unit (LU). Many times in personal computers (PCs) floating-point operations are performed by a floating-point unit which resides on a separate chip referred to as a numeric coprocessor[1]. According to[1] 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.[2]. 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[1]. Many operations and 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[1].[3] indicated that floating-point multiplication is the most commonly utilized operation in many applications involving digital signal processing.[3] further indicates that floating-point multiplication accounts for approximately 37% of the floating-point operations in benchmark applications. Multiplication consumes significant time compared to other arithmetic units used in basic mathematical computations[4]. Power management has also become extremely important especially in the case of portable electronic systems[5]. Large power dissipation results in the chip having a higher temperature profile and as such, this affects the performance of the chip[5]. According to[5] the multiplier is a major power dissipation source and at the same time high speed multiplication is a major requirement for high performance computing. 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. Before this can be achieved it may be useful to perform a comprehensive review of existing floating-point multiplier systems in such a way that can advise further evolution of the area of floating-point multiplication.

IEEE 754-2008 Standard for Floating Point Numbers

Floating-point numbers are numbers that cannot be represented by integers because they contain fractional components or simply because they fall outside the range of values possible within a systems bit-width[6]. By representing the numbers in floating-point format the accuracy and resolution of the numbers are preserved when compared to fixed-point format. In the binary system, floating-point numbers are represented in both single precision and double precision formats. Three components make up these formats: sign, exponent and mantissa components[7]. The single precision floating-point format comprises a 1-bit sign (bit 31), 8-bit exponent (bits 23-30) and 23-bit mantissa (bits 0-22). The bias in this format is 127. The double precision floating-point format comprises a 1-bit sign (bit 63), 11-bit exponent (bits 52-62) and 52-bit mantissa (bits 0-51). The bias in this format is 1023[1]. The Quadruple precision floating-point format comprises a 1-bit sign (bit 127), 15-bit exponent (bits 112-126) and 112-bit mantissa (bits 0-111). The bias in this format is 16383[7]. The Octuple precision floating-point format comprises a 1-bit sign (bit 255), 19-bit exponent (bits 236-254) and 236-bit mantissa (bits 0-235). The bias in this format is 262,143. A summary of this data is given in Table I.

Table 1. Comparison of Single, Double, Quadruple and Octuple Precision Floating-Point Formats

	Single Precision	Double Precision	Quadruple Precision	Octuple Precision
Bits	32	64	128	256
Sign bit	1 (bit 31)	1 (bit 63)	1 (bit 127)	1 (bit 255)
Biased Exponent	8 (bits 23-30)	11 (bits 52-62)	15 (bits 112-126)	19 (bits 236-254)
Mantissa	23 (bits 0-22)	52 (bits 0-51)	112 (bits 0-111)	236 (bits 0-235)
Bias	127	1023	16383	262143
Range	2^{-126}–2^{+127}	2^{-1022}–2^{+1023}	2^{-16382}–2^{+16383}	$2^{-262142}$–$2^{+262143}$

Floating-point multiplication utilizing single, double, quadruple and octuple precisions formats first require the computation of the sign bit which is done by the XOR operation of the two sign bits. The product exponent is computed by summation of the two exponents. The value obtained is then added to the bias. The mantissa is computed by applying binary multiplication of the mantissa components of both floating-point numbers[1].

Review of Existing Floating Point Multiplier Systems

Floating-point multiplication utilizing single, double, quadruple and octuple precisions formats first require the computation of the sign bit which is done by an XOR operation of the sign bits of the input floating-point numbers. The product exponent is computed by summation of the two exponents, after which a bias is added. The value obtained is then added to the bias. The mantissa is computed by applying binary multiplication of the mantissa components of both input floating-point numbers[1]. This section presents a review of existing implementations of floating-point multiplier systems.

Single Precision Floating-Point Multiplication

[8] presented the development of a fast and compact GaAs IEEE single precision floating-point multiplier. In[8] Booth's algorithm is used together with a modified carry-save array in order to reduce the partial product addition and interconnection requirements.[8] indicated that GaAs are inherently superior in radiation hardness, saturation velocity, high temperature operation and electron mobility[9]. The multiplier system consisted of exponent and mantissa block. The exponent block was responsible for the sign computation as well as the biased exponent computation. The system of[8] computed biased exponent simply by addition of the biased exponents of each floating-point number after which the bias was subtracted, hence resulting in the biased exponent. The mantissa block of[8] is a 24x24-bit parallel multiplier consisting of an array of modified carry save adders along with a final adder. The array of modified carry save adders is used in the partial product reduction process to reduce the number of partial products from 24 to 13. A Wallace tree is then used to further reduce the partial products to 2. The final adder completes the partial product reduction process[8]. The multiplier system was designed using Vitesse H-GaAs-II 0.8um technology. Spice was used for simulation of the performance of the system. When implemented on AT&T GaAsHFET 1um the delay of the system of[8] was 9.25ns, clock frequency was 74.75MHz, area was $8 \times 9.5\text{mm}^2$ and power utilization was 7W. When implemented on GaAsMESFET 0.8um the delay of the system of[8] was 4ns, clock frequency was 14MHz, area was $2.43 \times 3.77\text{mm}^2$ and power utilization was 3.5W.[10] presented a single precision

IEEE 754 floating-point multiplier with low power and high speed.[10] claimed that the bottleneck of any single precision floating-point multiplier was the 24x24 bit integer multiplier used in the mantissa calculation. In this implementation, the Urdhava Triyakbhyam algorithm that is a component of ancient Indian Vedic Mathematics was utilized for increasing efficiency of implementation. Reconfiguration was also utilized at runtime for the purpose of power savings. The system of[10] computed biased exponent simply by addition of the exponents of each floating-point number after which the bias is added, hence resulting in the biased exponent. The mantissa multiplication component of[10] comprised of a 24x24-bit multiplication operation that was done using four parallel 12x12-bit multiplication modules. These 12x12-bit multiplication units were constructed using a series of 4x4-bit multiplier units. As such the entire 24x24-bit multiplication operation was divided into thirty-six (36) 4x4-bit multiplier units operating in parallel. The system of[10] was implemented on the Xilinx Virtex E XCV300e package BG432, speed grade -8 using Xilinx Webpack 6.1. Two versions of the system were implemented – with and without reconfiguration.[10] claimed that the proposed multiplier without reconfiguration had an area utilization of 2967 slice registers and estimated delay of 37.553ns, while the proposed multiplier reconfiguration had an area utilization of 3149 slice registers and estimated delay of 41.203ns.[11] presented hardware description of IEEE 754 single precision floating-point multiplier. Analysis was made using Booths algorithm and Canonical Signed Digit (CSD) algorithm were used in the implementation. The system of[11] computes the biased exponent by sum of the two exponents of the input floating-point numbers followed by addition of the bias to sum of both exponents. The mantissa multiplication was done by use of the Booth Algorithm and CSD. The system was synthesized for the Virtex E XCV300e, package bg432, speed grade -8, using Xilinx ISE after which it was simulated using Modelsim.[11] indicates that proposed multiplier achieved a delay of 22.859 ns, area of 1331 slice registers and maximum frequency of 333.33MHz.[11] claimed that the system implemented resulted in an improvement of 57.77% in the area utilized and 44.52% in the system delay when compared to the system implemented in[10].[12] proposed a new reversible design for the single precision floating-point multiplier by use of an operand decomposition technique[12]. The system performed the 24x24 bit reversible multiplication using nine (9) reversible 8x8-bit Wallace tree multipliers. A new reversible design of the 8x8 bit Wallace tree multiplier which was optimized in terms of quantum cost, delay and number of garbage outputs was proposed by[12]. The system of[12] computed the biased exponent by first subtracting bias from the exponent of the multiplicand, then adding this result to the exponent of the multiplier input.[12] computed performed mantissa multiplication by use of a 24x24 reversible partition multiplier. There was no mention of the delay or hardware utilization of the system.[13] presented an efficient implementation of the IEEE 754 single precision floating-point multiplier unit. The system of[13] computed biased exponent simply by addition of the biased exponents of each floating-point number using a binary adder after which the bias was subtracted, hence resulting in the biased exponent. The mantissa multiplication operation of the system of[13] was done using a 24x24 carry-save multiplier which consisted of three stages which were constructed using carry-save adders. The system was implemented in VHDL using Xilinx ISE and Precision Synthesis tools MG (2010), the target being Xilinx Virtex-5 5VFX200TFF1738 using timing constraint of 300MHz.[13] indicates the rounding was not implemented in the system, however a normalizer was included in the system. The system was simulated to obtain area and maximum frequency. Its parameters were compared with that of the single-precision implementation found in the Xilinx Coregen.[13] did not indicate the delay or hardware utilization of the system. Based on the comparison it was realized that the system implemented in[13] utilized greater area than that of the system

from Xilinx Coregen, while the system of[13] has a greater max frequency (301.114MHz) than that of the unit from Xilinx Coregen (221.484MHz).[14] presented an efficient floating-point multiplier system using the Karatsuba algorithm and the IEEE 754 format for floating-point numbers. The system was implemented using Verilog HDL on the FPGA cyclone II device as the target and Altera-Quartus II as the development environment.[14] incorporated a three-stage pipelining scheme with latency 8 clock cycles in the design. The system of[14] computed biased exponent simply by addition of the biased exponents of each floating-point number using ripple carry adders after which the bias was subtracted, hence resulting in the biased exponent. The mantissa multiplication operation of[14] was done using a 24-bit unsigned multiplier, Karatsuba Multiplier which utilizes Vedic multiplication. The multiplier system of[14] was simulated using Modelsim. The max frequency of operation was determined to be 77.434MHz while the delay was determined 12.92 ns.[15] presented the development of a 24-bit Vedic multiplier using 3x3 Vedic multiplier as the basic building block.[15] also proposed an implementation of a IEEE-754 single precision floating-point multiplier unit capable of handling rounding, overflow and underflow conditions. The proposed and conventional floating-point units are implemented and simulated using ISE simulation tool. The system implementation was done on iWave using the Spartan 6 XC6S1x25t-2fgg484 as the development platform. The system of[15] computed biased exponent simply by addition of the biased exponents of each floating-point number using a binary adder after which the bias is subtracted, hence resulting in the biased exponent. The system implemented by[15] utilizes 3x3 Vedic multiplier blocks (seen in Figure I). A 6x6 multiplier block is constructed using 3x3 blocks after which a 12x12 block is constructed using 6x6 blocks. Finally, a 24x24 multiplier block (seen in Figure II) is constructed using 12x12 Vedic multiplier blocks and three 24-bit ripple carry adders[16].

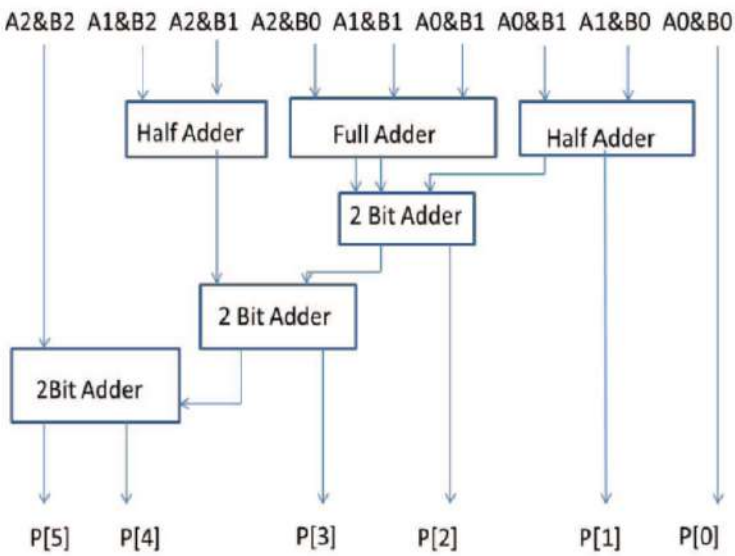

Figure 1. 3x3 Vedic Multiplier Block Implementation. *Source*: Data from[15]

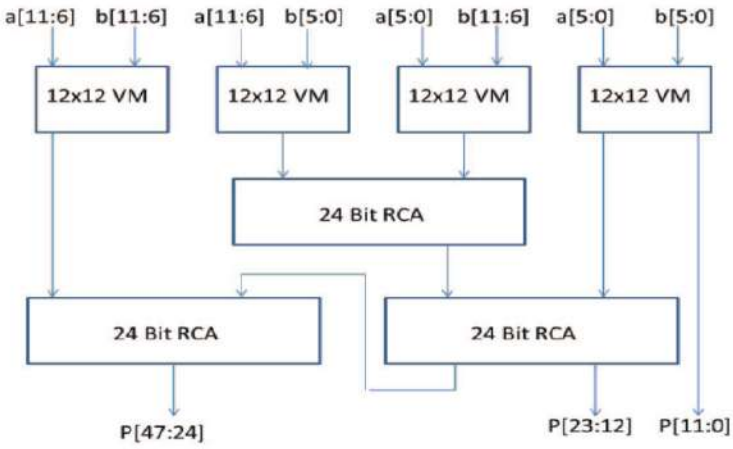

Figure 2. 24x24 Vedic Multiplier Block Implementation. *Source*: Data from[15]

The system developed in[15] and conventional multipliers were compared in terms of maximum combinational path delay and hardware utilization. Results indicate that the method utilized by[15] had speed which was 21.7% faster than that of the conventional system (seen in Table II). The area on the Spartan 6 utilized for the system of[15] is also smaller than that used by the conventional system (seen in Table III). The system also utilized 1018 sliced LUTs and 96 bonded IOBs as seen in Table III.

Table 2. Path Delay Comparison for Proposed and Conventional Systems

Proposed Floating Point Multiplier (ns)	Conventional Floating Point Multiplier (ns)	Increased Speed (%)
49.797	63.595	21.7

Table 3. Area Comparison for Proposed and Conventional Systems

Parameter	Proposed Floating Point Multiplier	Conventional Floating Point Multiplier
Number of Slice LUTs	1018	1347
Number of Bonded IOBs	96	96

[6] presented the design and implementation of floating-point multiplier with the aim of reducing path delay (see Figure III). Reduction of the path delay can be done by reducing the delay caused by propagation of the carry in adders utilized in the multiplier design[6]. The architecture implemented was based on IEEE-754 standard for single precision format. The system of[6] computed biased exponent simply by addition of the biased exponents of each floating-point number using a binary adder after which the bias is subtracted, hence resulting in the biased exponent. The system of[6] carried out mantissa multiplication simply by first computing partial products using radix-4 booth programming, after which a Wallace tree was used for partial product reduction.

The modules were implemented in Verilog and synthesized for the Spartan 3 XC3S500-4GFG320 device using Synopsys Design Compiler, and a pipelined approach was utilized. The system was later interfaced with a DSP processor. The system of[6] computed biased exponent simply by addition of the biased exponents of each floating-point number after which the bias was subtracted, hence resulting in the biased exponent. The multiplier unit implemented in[6] and the performance of the system along with all sub modules (Modified Booth Encoder, KoggeStone Adder, Pipelined KoggeStone Adder and Wallace Tree) were summarized. These four systems were compared in terms of power, area, path delay and hardware utilization and the results are shown in Tables IV-V.

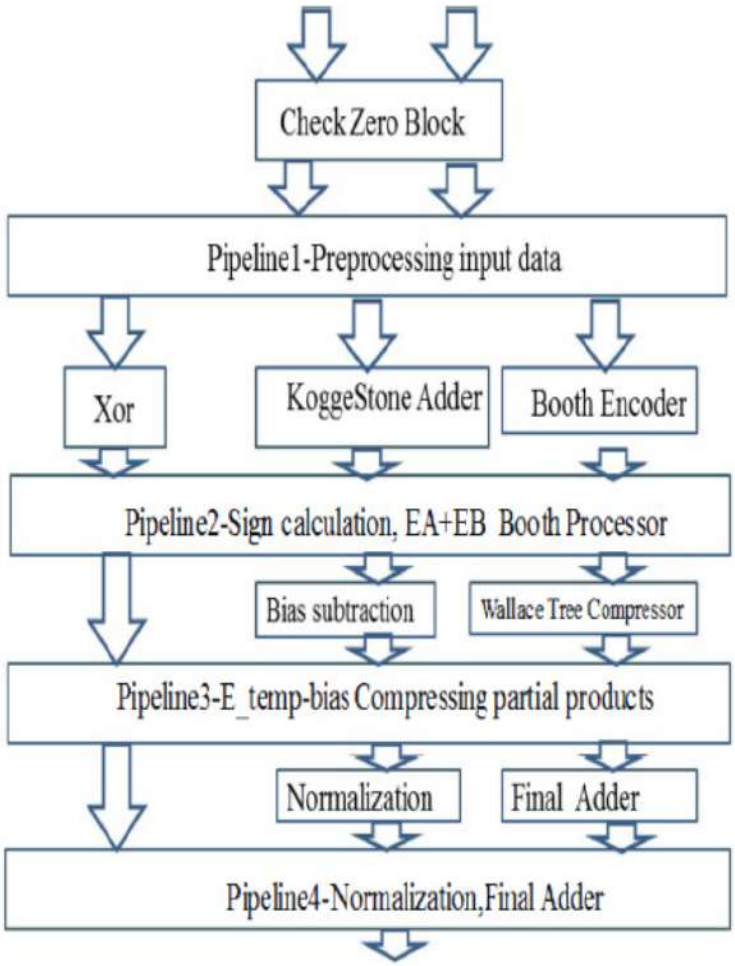

Figure 3. Proposed Multiplier Unit. *Source*: Data from[6]

Table 4. Performance Summary of Multiplier Sub-Units

	Power (μW)	Area (μm^2)	Delay (ns)	No. of LUTs	No. of Slices
Proposed Multiplier Unit	173.72	13325.45	29.72	2270 /9312	1208 /4656
Modified Booth Encoder	21.72	355.35	4.84	125 /9312	72 /4656
KoggeStone Adder	27.05	360.00	3.72	355 /9312	196 /4656
Pipelined KoggeStone Adder	37.25	420.00	3.92	358 /9312	208 /4656
Wallace Tree	44.25	524.20	7.92	895 /9312	515 /4656

Table 5. Performance Summary of Proposed Multiplier

	Power (μW)	Area (μm^2)	Delay (ns)	No. of LUTs	No. of Slices
Proposed Multiplier Unit	173.72	13325.45	29.72	2270 /9312	1208 /4656

[17] presented the design and implementation of a low power probabilistic floating-point multiplier (seen in Figure IV). In[17] probabilistic computation is utilized in order to attain large energy savings in the floating-point multiplier at the expense of calculation errors/accuracy.

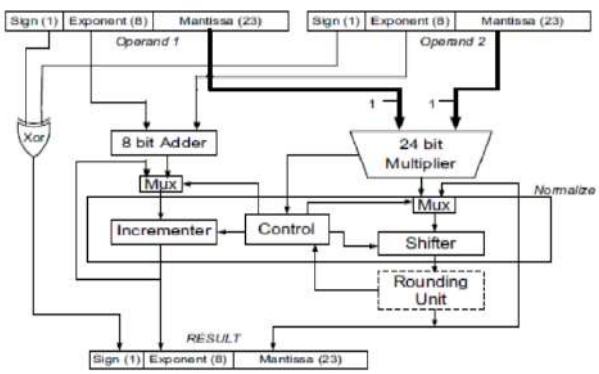

Figure 4. Architecture of 32-bit Single Precision Floating-Point Multiplier. *Source*: Data from[17]

[17] indicated that the 24-bit multiplier block of a single precision floating-point multiplier consumes 81% of the power consumed by the floating-point multiplier block.[17] also indicated that 18% of the power consumed is done by the rounding component. In[17] rounding is not utilized; hence this means that 99% of the power is consumed by the 24-bit multiplier block. According to[17] AND gates account for 5% of the power consumed by the 24-bit multiplier block, while full adders account for the remaining 95%. Since most of the power is utilized by full-adders,[17] applied a low power approach to the design and implementation of the full-adders.

[17] presented two methods which could have been applied to making the 24-bit multiplier operate at low power. The first termed as the Sleep Technique allows some of the less significant full adder logic of the multiplier go to sleeping mode while the remaining stay awake while operating at the normal

voltage level. The second method utilizes BIased VOltage Scaling (BIVOS) where the full-adders of 24-bit multiplier are provided with a biased supply voltage depending on the significance of the computation being carried out.[17] proposed a new 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 are supplied with a biased voltage. The columns of lesser significance with which are not currently in sleep mode are operated at low voltage when compared to the columns of greater significance[17]. Results of[17] indicate that that implementing the floating-point multiplier using the BIVOS+Sleep strategy results in power saving of 62% as indicated in Table VI.

Table 6. Comparison of Image Quality vs Power Consumption for BIVOS, Sleep and BIVOS+Sleep Floating-Point Multiplier

PSNR (dB) Relative to the Ideal Image	BIVOS	Sleep	BIVOS + SLEEP
58 dB	80%	75%	66%
47 dB	55%	51%	38%

[18] presented a self-timed 32-bit floating-point multiplier with a carry lookahead adder. The implementation was based on the IEEE 754 32-bit floating-point multiplier standard.[18] also presented a self-timed carry lookahead adder that was implemented using dual-rail signalling[19] in the input, sum and carry bits. The sign bit was generated from an XOR operation of sign bits of both inputs. The biased exponent was obtained by summation of both biased exponents of inputs using the self-timed carry-lookahead adder followed by subtraction of the bias. The mantissa multiplication operation was done by addition and shifting operations. The addition operations were done using the self-timed carry-lookahead adders constructed in[18]. The system was implemented using Xilinx ISE 14.4 and performance was compared with that of a Synchronous implementation.[18] claimed that the new system saw marginal improvements in path delay compared to synchronous implementation.[18] also claimed that the use of the self-timed floating-point multiplier developed resulted in a 20% improvement in power consumption when compared to the synchronous multiplier implementation.

Double Precision Floating-Point Multiplication

[20] presented the implementation of high speed floating-point double-precision multiplier which is compliant with the IEEE 754 standard for floating-point numbers, hence handling overflow, underflow and rounding conditions. The system of[20] computed the biased exponent by the summation of both biased exponents of inputs using binary adders followed by subtraction of the bias.[20] broke up the input mantissa bits into ten (10) parts, partial products were generated, after which partial products were summed together. The system of[20] was implemented on the Virtex-6 xc6vlx75t-3ff484 using Xilinx ISE 12.1i. The system was simulated using Modelsim 6.6c. The minimum period was determined to be 2.411ns which maximum frequency was determined to be 414.714MHz. The results of the implementation of[20] were compared with that of single precision implementations from[13] and Xilinx Core from Xilinx Coregen. Area utilized and maximum frequency of[20] was determined to be greater than that of[13] and Xilinx Core from Xilinx Coregen.

[21] indicated that the double-precision floating-point multiplier required a 52x52 mantissa multiplication operation and as such have proposed a novel approach towards the decrease of this huge mantissa multiplication operation.[21] utilized the Karatsuba and Urdhva Tiryagbhyam techniques in the

implementation of the multiplier system.[21] indicated that the traditionally the partial products of the multiplier are added separately and took a significant amount of time to completion. The proposed method presented by[21] concurrently added the partial products during the multiplication operation, hence reducing the delay. The double-precision floating multiplier system of[21] was implemented in Verilog HDL on Xilinx ISE using Virtex-5 FPGA as the target. The system of[21] catered for the detection of all the exceptional cases of the IEEE standard such as overflow, underflow and infinite zero. Normalization is also incorporated in the implementation. The system of[21] computed the biased exponent by the summation of both biased exponents of inputs using binary adders followed by subtraction of the bias.[21] claims that the path delay of the Karatsuba multiplier was determined to be 18.139 ns, while the path delay of the Urdhva Tiryagbhyam multiplier was determined to be 15.034 ns. Hardware utilization was determined to be (798 sliced registers, 1378 sliced LUTs) and (525 sliced registers, 936 sliced LUTs) respectively[21].[22] presented an area efficient runtime reconfigurable double precision floating-point multiplier architecture. The system conformed to the IEEE-754 floating-point standard.[22] presented three multiplier architectures for double precision floating-point multiplication. The first architecture (Truncated Block Multiplication - TBM) is a truncated multiplication block. This system was designed using Vedic mathematics. In this system the two 53-bit mantissa are multiplied together after which the result is trimmed back to 53-bits by suitable truncation or rounding of the result. This was expected to result in minor loss of accuracy[22]. The second architecture (3-Partition Karatsuba Multiplication - PKM) was included for regaining the loss of accuracy from the multiplication operation performed by the first block. In this architecture the both 53-bit mantissa are separated into three parts – two 18-bit and one 17-bit component. Each component of both numbers are multiplied and the partial products are generated and added. The three results are then added and the overall result is brought back to standard form by post normalization and rounding methods[22]. The third architecture (Double Precision Dual Single Precision Multiplier - DPdSP) was responsible for performing either single or double precision floating-point operation. The design was achieved via a resource sharing approach. The system can produce one double precision floating-point result of two single precision floating-point results[22]. The system of[22] computed the biased exponent by the summation of both biased exponents of inputs using binary adders followed by subtraction of the bias. All three systems developed in[22] were implemented in VHDL using Xilinx ISE and simulated using the Synopsys tool. The target devices for this research were the Virtex-4 xc4vfx100-12ff1517 and the Virtex-5 xc5v1x155-3ff1760. Performance Comparison for the three architectures is presented in Table VII.

Table 7. Performance Comparison for TBM, PKM and DPDSP

Performance Parameters	Target Device	TBM	PKM	DPdSP
Number of Slice Registers	Virtex 4	1923	3419	3288
	Virtex 5	2439	4468	4484
Number of 4-Input LUTs	Virtex 4	3348	5950	5729
	Virtex 5	-	-	-
Delay / ns	Virtex 4	44.91	46.74	58.93
	Virtex 5	40.03	42.23	49.91

Quadruple Precision Floating-Point Multiplication

[23] indicated that quadruple precision multiplication is required especially in high precision requirements of a given application. The system of [23] computed the biased exponent by the summation of both biased exponents of inputs using binary adders followed by subtraction of the bias. In the case of the mantissa multiplication procedure, [23] proposed a hardware efficient approach for the implementation of fully pipelined integer multipliers. [23] later presented the implementation of quadruple precision floating-point multiplier which utilized DSP48 as the basic building blocks. Block sizes of 17-bits were used as the starting point. These were used in constructing bigger multiplier blocks such as 34-bit, 51-bit, 66-bit, 130-bit and 113-bit multiplier blocks. The 113-bit multiplier block was utilized in mantissa multiplication for this system. The systems were implemented in Verilog HDL using Xilinx ISE. The target utilized was the Xilinx Virtex 4 xc4vfx100-12ff1517. The implemented system was simulated using Modelsim-SE. [23] indicated that proposed 113-bit binary multiplier and quadruple floating-point multiplier had hardware utilizations of 2373 and 2464 cycles respectively. [23] also indicated that proposed 113-bit binary multiplier and quadruple floating-point multiplier had max frequencies of 310MHz each. It was unclear whether the latency given for both were in nanoseconds or cycles. [24] presented a new quadruple precision floating-point multiplication algorithm which was based on polynomial multiplication. The system of [24] computed the biased exponent by the summation of both biased exponents of inputs using binary adders followed by subtraction of the bias. The mantissa multiplier using the polynomial multiplication method required the system to utilize 98 multipliers and 1598 LEs if a multiplier core from the development environment is used. The new system was then implemented in Verilog HDL using Quartas II 8.1 IDE. The target for this implementation was the Altera EP2C7089C6. Results of [24] indicated that the multiplication algorithm used effectively reduced hardware resource occupancy. [24] also indicated that the delay of the floating-point multiplier in this case is 96 ns [25].

Multi-Precision Floating-Point Multiplication

[1] presented a Vedic multiplier based on the IEEE 754 which was capable of both single and double precision floating-point multiplication with performance comparable to the Karatsuba and Booth type floating-point multipliers (see Figure V). The system of [1] computed the biased exponent by the summation of both biased exponents of inputs using binary adders followed by subtraction of the bias. The mantissa multiplication component of [1] was developed by utilization of Vedic multiplier principles and as such all bits of both inputs were subjected to vertical and crosswise multiplication. The floating-point multiplier of [1] was synthesized for the Virtex-7 7v2000tflg1925-2. According to [1] the application of the Vedic multiplier to digital signal processing reduced path delay by 40-60% when compared to conventional procedures. The findings of [1] conclude that the Vedic multiplier utilized less hardware and also has shorter path delay than both the Karatsuba floating-point multiplier. The Vedic multiplier utilized more hardware than the Booth type floating-point multiplier and had shorter delay. [1] also concluded that Booth type is a poor choice for single-cycle multiplication but however can be efficiently implemented in the case of highly pipelined systems [26].

[27] attempted to improve the performance of the multiplier developed by [15], hence producing a multiplier which was capable of facilitating both IEEE 754 single and double precision floating-point multiplication. Implementation was done using Vedic Mathematics using same approach outlined in [15]. The system of [27] computed the biased exponent by the summation of both biased exponents of inputs

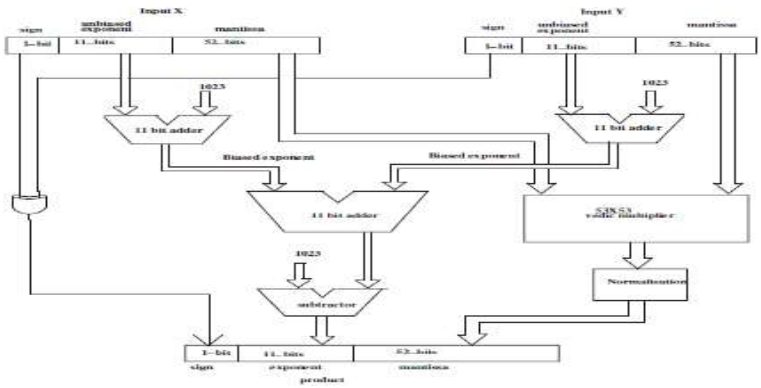

Figure 5. Block Diagram of IEEE-754 Double Precision Floating-Point Multiplier. *Source*: Data from [1]

using binary adders followed by subtraction of the bias. The mantissa multiplication component of the single precision multiplier was implemented using a 24x24 Vedic Multiplier while the double precision mantissa multiplication was done using a 53x53 Vedic Multiplier as shown in Figure VI.

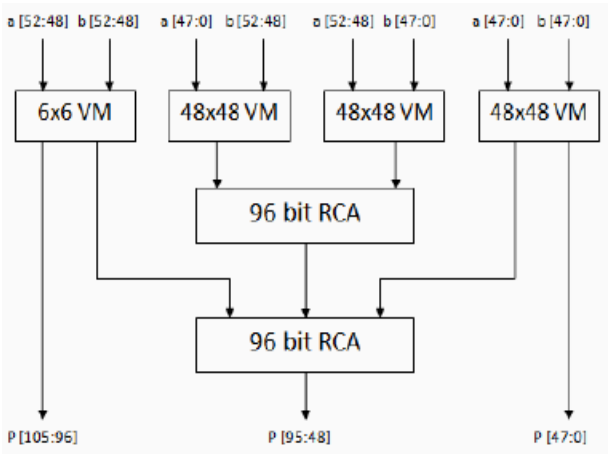

Figure 6. Proposed 53x53 Vedic Multiplier Block Implementation. *Source*: Data from [27]

The system was implemented using Xilinx ISE 14.6. Table VIII presents the area utilization for the double precision floating-point multiplier of [27] while Table IX presents the delay comparison for previous system from [15] and proposed system from [27].

[3] presented a pipelined IEEE floating-point multiplier system which was capable of carrying out either single-precision or double precision multiplication. [3] claims that the latency of the system in single precision operation was 2 cycles while in double-precision operation it was 3 cycles. The clock cycles however are not related to the reference clock but rather to the clock common to all pipelined registers.

Table 8. Area Utilization for Double Precision Floating-Point Multiplier

Logic utilization	Used	Available	Utilization
No. of slices LUTs	5153	204000	2%
No. of bonded IOBs	192	600	32%

Table 9. Delay Comparison for Previous System and Proposed Systems

Delay	Used	Available
SP FPM	49.497 ns	21.823 ns
DP FPM	-	45.169 ns

Hardware requirements for this system were minimized by use of half-sized multiplication array and by also ensuring that both precisions use the same rounding units.[3] indicated that the system presented supported all IEEE rounding modes.[3] presented a new rounding algorithm which supplied different precisions, hence allowing its use by multiple precisions.[28] presented a hardwired algorithm for the computation of variable precision floating-point multiplication based on the use of a parallel multiplier of size m which would be used to compute nm bits.[28] only focused on the mantissa component of the floating-point inputs. A very important component of the implementation of[28] is that the partial products are added as soon as they are computed in order to reduce demands on memory for storing partial products.[28] the variable precision is brought about by the use of both floating-point and fixed-point formats. Resources and the resulting architecture are dedicated to the multiplication of operands of size ranging from 1x64 to 64x64 bits with period of n^2 33ns. The system of[28] computed the biased exponent by the summation of both biased exponents of inputs using binary adders followed by subtraction of the bias. The system of[28] was implemented on Virtex 2 XC2V1000 (-5) bg575 using Xilinx ISE 6.3. The system was simulated using Modelsim PE 6.0. Results of[28] indicated that the path delay for the system was 19,008 ns in single precision mode and 135,168 ns in double precision mode. The system also occupied 2381 slice registers on the Virtex 2 platform.[29] presented a multi-mode floating-point multiplier system which operated with the single, double and quadruple precision formats specified by IEEE 754-2008 standard. The system of[29] computed the biased exponent by the summation of both biased exponents of inputs using binary adders followed by subtraction of the bias. The system of[29] is pipelined in order to maximize throughput, hence allowing the execution of one quadruple precision floating-point multiplication, two double precision floating-point multiplications processes and four single precision floating-point multiplication processes. The system was implemented in VHDL and synthesized for a 45NM technology using Synopsys for the front-end implementation and Cadence for the back-end.[29] indicated that the proposed system was implemented using a divide and conquer technique which basically utilized high precision multiplications by performing smaller sized multiplications and at the end adding up the partial products to produce the final result.[29] performed an example implementation of the system on VLSI and as a subsequent verified that the implementation achieved a maximum operating frequency of 505MHz.[23] presented a configurable dual-mode floating-point multiplier which was capable of functioning as double precision mode as well as process in single precision mode (see Figure VII). The architecture proposed is based on flow of floating-point multiplication capable of processing in both normal and sub-normal operands along with exceptional

case handling. The system proposed by[23] was implemented for ASIC (UMC 90nm) technology.[23] utilized state of the art computational flow of floating-point multiplication in the design of the proposed system. Each individual stage was reconstructed using "efficient resource sharing and tuned datapath"[23] as a means of minimizing the system's multiplexing circuitry. The system is also pipelined to maximize throughput and is aimed at ASIC UWMC 90nm technology. The system comprised of four pipeline stages. The first pipeline stage dealt with data extraction, sign and exponent processing and dual-mode mantissa multiplication (using a Dadda-tree multiplier and Kogge-Stone adder). The first 6 levels of the Dadda-tree multiplier lay in the first pipeline stage while the last 2 levels fell in the second pipeline stage. The system of[23] computed the biased exponent by the summation of both biased exponents of inputs using binary adders followed by subtraction of the bias. After implementation the system was tested. The system implemented in[23] utilized 11.6% more area and 6.74% more time in its execution when compared to a single mode double-precision multiplier.

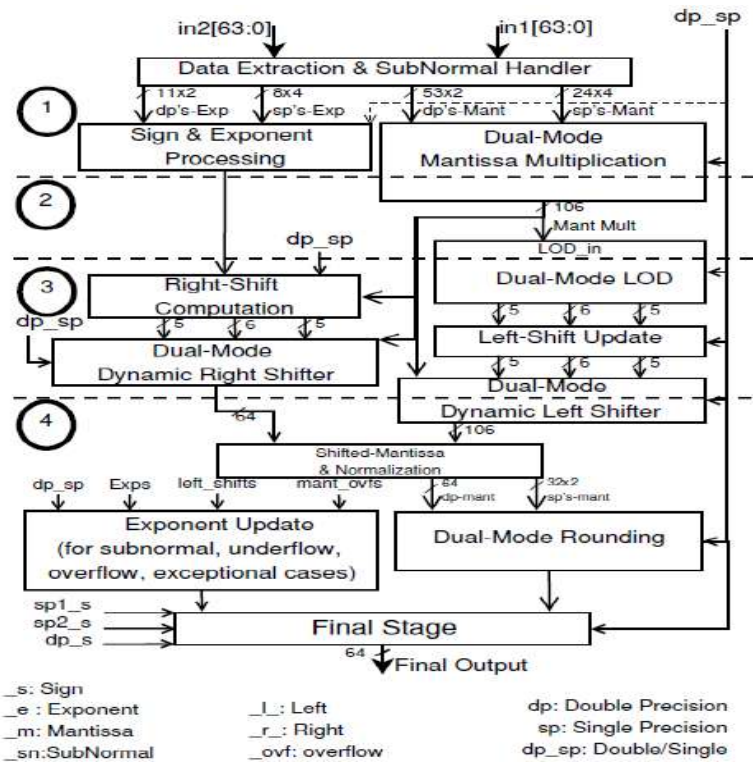

Figure 7. DPdSP Multiplier Architecture. *Source*: Data from[23]

[2] presented a run-time reconfigurable floating-point multiplier system which was implemented on the Xilinx Virtex 4 using Xilinx ISE 14.7.[2] utilized a combination of the Urdhva-Tiryagbhyam algorithm and Karatsuba algorithm in order to implement the unsigned binary multiplier, in order to further increase the efficiency of the multiplier. This multiplier utilizes crosswise and horizontal multiplication operations.

The multiplier system implemented in[2] had 6 modes of operation which are selected based on the requirements of the application to which it must be used. Therefore, the system can perform floating-point multiplication for varying mantissa sizes all depending on the precision requirements of the application. The system of[2] computed the biased exponent by the summation of both biased exponents of inputs using binary adders followed by subtraction of the bias. Results of[2] indicate that the proposed 32-bit binary multiplier had a delay of 13.141 ns while the proposed single-precision floating-point multiplier had a delay of 16.392 ns.[30] presented the design and implementation of a multi-precision floating-point multiplier using Vedic mathematics as indicated in Figure VIII. The resulting multiplier supports single, double and quadruple precision floating-point multiplication. The system of[30] computed the biased exponent by the summation of both biased exponents of inputs using binary adders followed by subtraction of the bias. The mantissa multiplication component is based on Vedic mathematics which utilizes crosswise and vertical multiplication operations. The results of[30] indicated that the multi-precision multiplier presented utilized slightly more area than the double-precision floating-point multiplier.

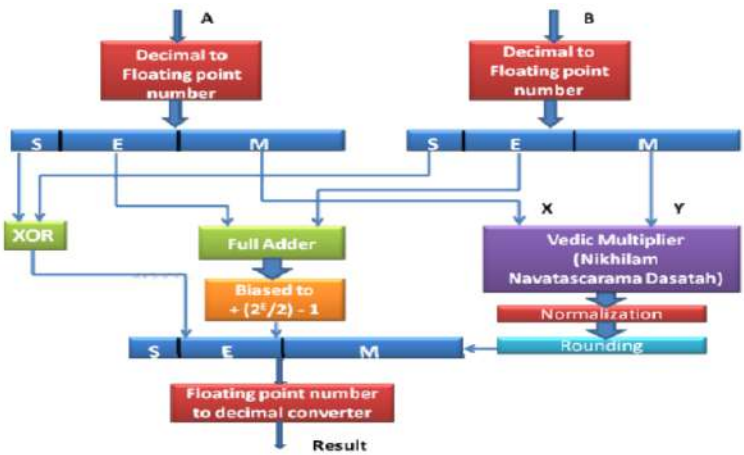

Figure 8. Block Diagram of the Resulting Universal Multiplier System. *Source*: Data from[30]

[31] presented a multi-function double precision floating-point multiplier system capable of performing one double-precision multiplication or one vector multiplication of two 2D vectors in single-precision floating-point format. The system of[31] computed the biased exponent by the summation of both biased exponents of inputs using binary adders followed by subtraction of the bias. The system of[31] utilized multiplexers for selection of data in the two modes – double precision and single-precisions mode. The binary multiplication section utilizes 4:2 Compressor Tree logic for partial product reduction. Pipelined and non-pipelined versions are produced and compared with conventional floating-point multiplier in terms of latency, area and power consumption. Results of tests conducted by[31] as seen in Figure IX indicate that the latency of the proposed system is 16% less than that of the conventional floating-point multiplier and the latency of the pipelined version of the proposed system is 24% less than that of the conventional floating-point multiplier.[31] also reported that the area of the proposed system is 47%

smaller than that of the conventional floating-point multiplier and the area of the pipelined version of the proposed system is 49% smaller than that of the conventional floating-point multiplier.[31] finally reported that the proposed system consumes 6% more power than the conventional floating-point multiplier and the pipelined version of the proposed system consumes 16% more power than the conventional floating-point multiplier.

LATENCY COMPARISON

Latency		32	64	Dual-64	[7]0.25μm	[8]90nm
COMB	ns	1.10	1.45	1.60	---	2.59
	FO4	34	45	49	---	58
PIPE	ns	0.70	0.90	0.98	5.07	---
	FO4	21.5	27.7	30	40.6	---

AREA COMPARISON

Area		32	64	Dual-64	[7]0.25μm	[8]90nm
COMB	μm^2	13587	52097	56569	---	253500
	gates	9435	36178	39284	---	176042
PIPE	μm^2	11745	41336	45571	---	---
	gates	8156	28076	31646	35103	---

POWER OF THE PROPOSED DESIGN

Power(mW)	32	64	Dual-64	[7]0.25μm	[8]90nm
COMB	6.71	24.00	25.41	--	212
PIPE	8.05	25.68	29.78	--	--

Figure 9. Comparison of Latency, Area and Power Consumption for Proposed Systems and Conventional System. *Source*: Data from[31]

[32] presented an FPGA based iterative hardware architecture for quadruple precision floating-point multiplication which is also capable of processing single, double and double-extended precision data.[32] utilized a series expansion method of division along with wide integer multiplication to further optimize the FPGA implementation. A number of expansion equations of[32] were implemented: equation (1) of[32] was implemented using adders, subtractors and multipliers. For the multi-precision component, a unified expression (3) of[32] was utilized for supporting all four precision computations. The system in this case utilized a look-up table of size $2^8 \times 113$ along with a multiplier of size 114x114-bit which is used iteratively with a FSM control unit. The system of[32] computed the biased exponent by the summation of both biased exponents of inputs using binary adders followed by subtraction of the bias. The system of[32] was demonstrated on Xilinx Virtex-7 FPGA and according to[32] the implementation

yielded improvement in latency along with significant saving in area utilized when compared to[33] as shown in Table X.

Table 10. Performance Comparison between Multipliers Produced

	(Diniz and Govindu 2006)	(Jaiswal and So 2016)
Latency	118 (QP)	9/11/12/13 (SP/DP/DPE/QP)
Throughput	NA (QP)	8/10/11/12 (SP/DP/DPE/QP)
LUTs	26811	7440
FFs	13809	2584
DSP48	-	18
Freq (MHz)	50	89

[34] presented a fully parallel decimal floating-point multiplier based on parallel fixed-point multiplier[35] and which was compliant with the 2007 draft of the IEEE P754 standard for floating-point arithmetic (see Figure X).[34] claimed that the novelty in the design was that at the time, it was the first parallel decimal floating-point multiplier which offered low latency and high throughput. The multiplier system in[34] used 64-bit decimal floating-point format with significands encoded in Densely Packed Decimal (DPD)[36]. The system had 16 mantissa bits and a bias of 398.

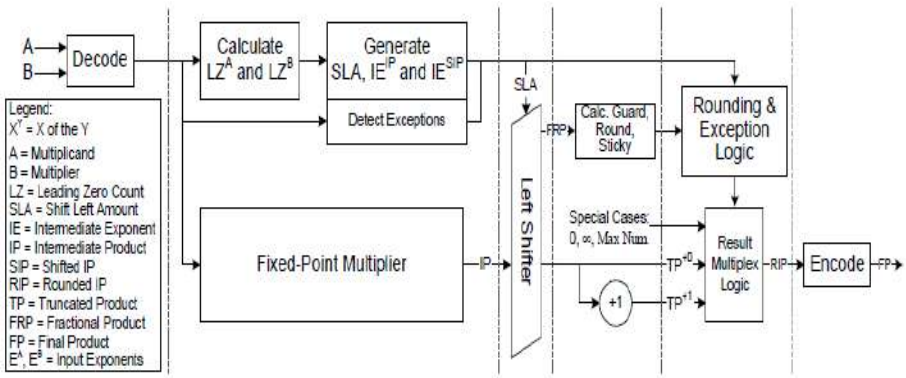

Figure 10. Block Diagram of the Decimal Floating-Point Multiplier System. *Source*: Data from[34]

The radix-10 parallel fixed-point multiplier utilized special BCD digit recoding for the reduction of the logic utilized by the system[34]. The system allowed for the generation of partial products in parallel[34]. The fixed-point multiplier design consisted of three components – generation of multiplicand multiples, partial product selection, and partial products reduction[34]. The multiplier of[34] converted the floating-point number to BCD format and then utilized the fixed-point multiplier to generate a 32-bit BCD number referred to as an intermediate product (IP). The fixed-point multiplier utilized by[34] generated sixteen (16) decimal partial products in parallel and then computed the sum of these partial products using a carry-save adder (CSA) tree. The 32-bit intermediate product was then shifted to form the shifted immediate product (SIP). The SIP was then utilized in computation of the final result of the decimal floating-point multiplier system[34]. The system of[34] computed the biased exponent by

the summation of both biased exponents of inputs using binary adders followed by subtraction of the bias. The system of[34] was implemented in Verilog on LSI Logic's gflxp 0.11um CMOS standard cell library and Synopsys Design Compiler Y-2006.06-SPI. The performance summary of the parallel decimal floating-point multiplier is given in Table XI.

Table 11. Performance Comparison of Parallel Decimal Floating-Point Multiplier

Component	Delay (ps)	%	Area (μm^2)	%
Fixed-point Multiplier	2990	67.3%	326,841	50.2%
Round Logic	810	18.2%	14,125	2.2%
Left Shift	280	6.3%	17,414	2.7%
Datapath	360	8.1%	26,605	4.1%
Interconnect	N/A	N/A	270,243	41.5%
Entire Design	4400	%	651,435	100%

[37] presented an IEEE 754-2008 compliant parallel decimal floating-point multiplier system capable of exploiting the features of the Virtex-5 FPGA (see Figure XI). The system of[37] implemented early estimation of the shift-left amount, SLA and efficient decimal rounding technique. The system of[37] also provided all required exception handling, rounding modes, overflow and gradual underflow. It also incorporated pipelining to increase throughput of the system. The system is fully combinational and based on fast partial product generation, BCD recording schemes and BCD-4221 carry save adder (CSA) reduction tree[37]. The system first begins by decoding the input operands to extract the floating-point components – sign, biased exponent and mantissa[37]. The system then performs decimal fixed-point multiplication on the mantissa to give a result. The product sign is produced by XOR operation on the input signs. The intermediate exponent (IE) is calculated by an exponent computation unit that sums up the exponents and applies the bias[37].

[37] claimed that the developed systems achieved decimal floating-point multiplication within 35ns and an operating frequency of 192 MHz using 13 pipeline stages.[37] also indicated that the hardware utilization was approximately 8000-9000 LUTs.[38] proposed a variable-latency floating-point multiplier architecture which was compliant with IEEE 754-1985 and was deemed suitable for low-power applications (see Figure XII). The multiplier architecture of[38] splits the mantissa multiplier into the upper and lower components, and predicts the, sticky bit, carry bit, and mantissa product from the upper part. In the event that the prediction is correct the computation of the lower part is disabled and the rounding operation is simplified, hence allowing the system to consume less power.

The proposed multiplier was implemented in Verilog and synthesized using Synopsys Design Compiler with TSMC CMOS standard cell library. Synopsys Nanosim and Cadence SOC Encounter and were used to perform placement, routing, and full transistor-level power simulation[38]. Power simulation was performed at a clock frequency of 33.3 MHz and 1.8 V with 10,000 random input patterns.[38] indicated that results show that the proposed multiplier saved power and energy at the expense of area and delay overheads[38]. The performance summary for the low-power floating-point multiplier is given in Table XII.

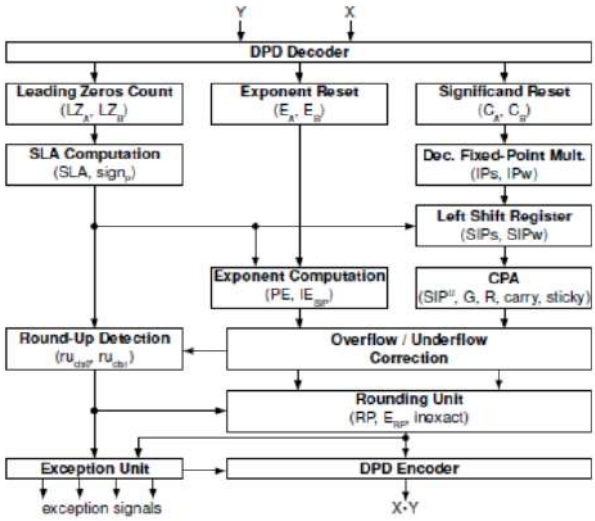

Figure 11. Block Diagram of the Parallel Decimal Floating-Point Multiplier. *Source*: Data from[37]

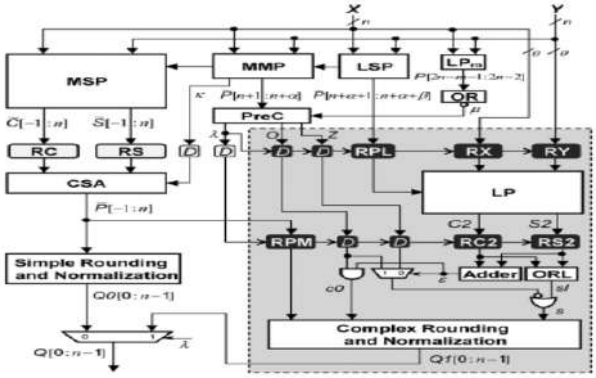

Figure 12. Block Diagram of Proposed Low-Power Floating-Point Multiplier. *Source*: Data from[38]

Table 12. Performance Summary for Low-Power Floating-Point Multiplier

n	Multiplier	Area (μm^2)	Delay (ns)
53	Fast_FM	548138	7.15
	VL_FM (α, m)	574268	8.00
24	Fast_FM	150416	4.52
	VL_FM (α, m)	153039	5.20

Summary of Performance of Various Floating-Point Multipliers

The reviewed multipliers of sections II(a) - II(d) were compiled and provided in Tables XIII - XV. Some sources reviewed did not provide all the required information and as such those sections in the tables were left unfilled. It is important to note also that no existing implementations of Octuple-Precision Floating-point multiplication were found.

Table 13. Summary of Various Single Precision Floating-Point Multipliers Reviewed

Source	Path Delay / ns	Hardware Utilization (Slice Registers)	Maximum Frequ
[8] – Compact GaAs	4.00		
[10] – without reconfiguration – Xilinx Virtex E (XCV300E-8BG432)	37.55	2,967	
[10] – with reconfiguration – Xilinx Virtex E (XCV300E-8BG432)	41.20	3,149	
[11] – Xilinx Virtex E (XCV300E-8BG432)	22.86	1,331	333.33
[28] – Xilinx Virtex 2 (XC2V1000-5BG575)	$n^2 \times 33$ per cycle	2,381	
[13] – Xilinx Virtex 5 (XC5VFX200TFF1738)			301.11
[14] – Altera Cyclone II	12.92		77.43
[15] – Xilinx Spartan 6 (XC6S1X2ST-2FGG484)	49.80	1,018	
[6] – Xilinx Spartan 3 (XC3S500 – 4GFG320)	29.72	1,208	
[1] – Xilinx Virtex 7 (7V2000TFLG1925-2)	28.02	2,928	
[2] – Xilinx Virtex 4 (XC4VFX12-10FF668)	16.39	977	226.51
[21] – Karatsuba Version – Xilinx Virtex 5 (XC5VLX110T-1FF1136)	18.14		
[21] – Urdhva Tiryagbhyam Version – Xilinx Virtex 5 (XC5VLX110T-1FF1136)	15.03		
[27] – Xilinx Virtex 6 (nomenclature not specified)	21.82	2,153	
[32] – ASIC (UMC 90nm)	9.00	7,440	89.00

Table 14. Summary of Various Double Precision Floating-Point Multipliers Reviewed

Source	Path Delay / ns	Hardware Utilization (Slice Registers)	Maximum Frequen
[2] – Xilinx Virtex 4 (XC4VFX12-10FF668)	18.97	3,877	173.95
[1] – Xilinx Virtex 7 (7V2000TFLG1925-2)	34.08	15,494	
[22] – TBM – Xilinx Virtex 4 (XC4VFX100-12FF1517) / Virtex 5 (XC5VLX155-3FF1760)	44.91 / 40.03	2,439	
[22] – PKM – Xilinx Virtex 4 (XC4VFX100-12FF1517) / Virtex 5 (XC5VLX155-3FF1760)	46.70 / 42.23	4,468	
[22] – DPdSP – Xilinx Virtex 4 (XC4VFX100-12FF1517) / Virtex 5 (XC5VLX155-3FF1760)	58.93 / 49.91	4,484	
[27] – Xilinx Virtex 6 (nomenclature not specified)	45.17	2,153	
[32] – ASIC (UMC 90nm)	11.00	7,440	89.00

Considerations for Development of Novel Floating-Point Multiplier System

The development of a novel floating-point multiplier system capable of eliminating (or at least minimizing) the effects of the weaknesses of existing multiplier systems is a viable consideration for the

Table 15. Summary of Various Quadruple Precision Floating-Point Multipliers Reviewed

Source	Path Delay / ns	Hardware Utilization (Slice Registers)	Maximum Frequer
[33] – Xilinx Spartan 3 (nomenclature not specified)	118.00	26811	50.00
[39] – Xilinx Virtex 4 (XC4VFX100-12FF1517)		2464	310.00
[24] – Altera EP2C7089C6	96.00		
[32] – Xilinx Virtex 7 (nomenclature not specified)	13.00	7440	89.00

benefit of digital electronic systems. All existing systems reviewed in this paper utilized biased exponent calculators which computed the product biased exponent by the summation of both biased exponents of both inputs using binary adders followed by subtraction of the bias using a subtractor or by twos' complement subtraction using adders. All existing systems catered for biased exponent calculation in the case of single precision only or double precision only or, quadruple precision only or, both single and double precision modes only. None has catered for the biased exponent calculation for all four precision modes: single, double, quadruple and octuple precision modes. None of them are capable of computing multiple batches of biased exponents for the various precision modes. The development of a novel multi-precision biased exponent calculator capable of supporting single, double, quadruple and octuple-precision modes and also capable of computing the biased exponent of multiple batches of input floating-point numbers in single, double, quadruple-precision modes (hence increasing the overall system throughput) is a viable consideration.

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 [1], [40] and [41] perform partial product reduction using Wallace or Dadda multipliers, thereafter the results are compressed using compressors. Others like [42] 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 [4] and [10] developed Vedic multipliers by utilizing smaller multipliers as building blocks to developing bigger multipliers.

Some systems like in [17] 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. Other systems such as the multiplier architecture of [38] 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. There is a need for development of a multiplier system that is capable of accumulating partial products as they are generated, hence reducing the delay at the expense of hardware utilization. At the same time 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 [27] 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. The development of such a system is also a viable consideration.

Finally, none of the existing binary multiplication systems reviewed 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 inclusion of a novel system called Mantissa Similarity Investigation (MSI) in the floating point multiplier system which can capable of further reduction in path delay is also a viable consideration.

Conclusions

This paper presented a comprehensive comparative review of existing floating point multiplier systems. The study focused on single, double, quadruple and multi-precision floating point multiplier architectures and identified engineering techniques involved in their development. A comparison of the performance of these systems in terms of metrics such as path delay, hardware utilization and even power consumption in some case were carried out. Weaknesses in the systems reviewed along with possible gaps in the area of research were identified. This paper also served to identify several recommendations and considerations for the development of a multi-precision floating-point multiplier system capable of treating with the weaknesses of multiplier systems identified. The development of systems involving such considerations are expected to result in shorter path delay than all existing implementations of floating-point multiplication reviewed in this paper. These contributions will likely be extremely useful to arithmetic operations in digital and computer systems presently and in the future.

References

[1] Kodali RK, Boppana L and Yenamachintala SS. Fpga implementation of vedic floating-point multiplier. In *IEEE International Conference on Signal Processing, Informatics, Communication and Energy Systems (SPICES)*. New York: IEEE, pp. 1–4. DOI: 10.1109/SPICES.2015.7091534.

[2] Arish S and Sharma RK. Run-time reconfigurable multi-precision floating-point multiplier design for high speed, low-power applications. In *2nd International Conference on Signal Processing and Integrated Networks (SPIN)*. New York: IEEE, pp. 902–907. DOI: 10.1109/SPIN.2015.7095315.

[3] Even G, Mueller SM and Seidel PM. A dual mode ieee multiplier. In *Proceedings of 2nd Annual IEEE International Conference on Innovative Systems in Silicon*. New York: IEEE, pp. 282–289. DOI: 10.1109/ICISS.1997.630271.

[4] Sharma R, Kaur M and Singh G. Design and fpga implementation of optimized 32-bit vedic multiplier and square architectures. In *International Conference on Industrial Instrumentation and Control (ICIC)*. New York: IEEE, pp. 960–964. DOI: 10.1109/IIC.2015.7150883.

[5] Anitha P and Ramanathan P. A new hybrid multiplier using dadda and wallace method. In *International Conference on Electronics and Communication Systems (ICECS)*. New York: IEEE, pp. 1–4. DOI: 10.1109/ECS.2014.6892623.

[6] Sunesh NV and Sathishkumar P. Design and implementation of fast floating-point multiplier unit. In *International Conference on VLSI Systems, Architecture, Technology and Applications (VLSI-SATA)*. New York: IEEE, pp. 1–5. DOI: 10.1109/VLSI-SATA.2015.7050478.

[7] IEEE (Institute of Electrical and Electronic Engineers). 754-2008 - ieee standard for floating-point arithmetic. revision of ansi/ieee std 754-1985. Technical report, IEEE, New York, 2008.

[8] Cui X, Liu W, Chen X et al. A modified partial product generator for redundant binary multipliers. *IEEE Transactions on Computers* 2015; 65(4): 1165–1171. DOI: 10.1109/TC.2015.2441711.

[9] Huntsman C and Cawthron D. The mc68881 floating-point coprocessor. *IEEE Micro* 1983; 3(6): 44–54. DOI: 10.1109/MM.1983.291185.

[10] Thapliyal H and Srinavas MB. A novel time-area-power efficient single precision floating multiplier. In *Proceedings of MAPLD*. New York: IEEE, pp. 1–3.

[11] Siddamal SV, Banakar RM and Jinaga BC. Design of high-speed floating-point multiplier. In *4th IEEE International Symposium on Electronic Design, Test and Applications (DELTA)*. New York: IEEE, pp. 285–289. DOI: 10.1109/DELTA.2008.19.

[12] Nachtigal M, Thapliyal H and Ranganathan N. Design of a reversible single precision floating-point multiplier based on operand decomposition. In *10th IEEE Conference on Nanotechnology (IEEE-NANO)*. New York: IEEE, pp. 233–237. DOI: 10.1109/NANO.2010.5697746.

[13] Al-Ashrafy M, Salem A and Anis W. An efficient implementation of floating-point multiplier. In *Saudi International Electronics, Communications and Photonics Conference (SIECPC)*. New York: IEEE, pp. 1–5. DOI: 10.1109/SIECPC.2011.5876905.

[14] Mehta A, Bidhul CB, Joseph S et al. Implementation of single precision floating-point multiplier using karatsuba algorithm. In *International Conference on Green Computing, Communication and Conservation of Energy (ICGCE)*. New York: IEEE, pp. 254–256. DOI: 10.1109/ICGCE.2013.6823439.

[15] Paldurai K and Hariharan K. Fpga implementation of delay optimized single precision floating-point multiplier. In *International Conference on Advanced Computing and Communication Systems*. New York: IEEE, pp. 1–5. DOI: 10.1109/ICACCS.2015.7324094.

[16] Mano MM and Kime CR. *Logic and Computer Design Fundamentals*. New Jersey: Prentice Hall, 1997.

[17] Gupta A, Mandavalli S, Mooney VJ et al. Low power probabilistic floating-point multiplier design. In *2011 IEEE Computer Society Annual Symposium on VLSI*. New York: IEEE, pp. 182–187. DOI: 10.1109/ISVLSI.2011.54.

[18] Beohar S and Nemade S. Vhdl implementation of self-timed 32-bit floating-point multiplier with carry look ahead adder. In *International Conference on Advanced Communication Control and Computing Technologies (ICACCCT)*. New York: IEEE, pp. 772–775. DOI: 10.1109/ICACCCT.2016.7831743.

[19] Cheng FC, Unger SH, Theobald M et al. Delay-insensitive carry-look ahead adders. In *Proceedings of 10th International Conference on VLSI Design*. New York: IEEE, pp. 37–63.

[20] Ramesh AP, Tilak AVN and Prasad AM. An fpga-based high speed ieee-754 double precision floating-point multiplier using verilog. In *International Conference on Emerging Trends in VLSI, Embedded System, Nano Electronics and Telecommunication System (ICEVENT)*. New York: IEEE, pp. 1–5. DOI: 10.1109/ICEVENT.2013.6496575.

[21] Rao YS, Kamaraju M and Ramanjaneyulu DVS. An fpga implementation of high speed and area efficient double-precision floating-point multiplier using urdhva tiryagbhyam technique. In *Conference on Power, Control, Communication and Computational Technologies for Sustainable Growth (PCCCTSG)*. New York: IEEE, pp. 271–276. DOI: 10.1109/PCCCTSG.2015.7503923.

[22] Shanmugapriyan S and Sivanandam K. Area efficient run time reconfigurable architecture for double precision multiplier. In *IEEE 9th International Conference on Intelligent Systems and Control (ISCO)*. New York: IEEE, pp. 1–6. DOI: 10.1109/ISCO.2015.7282355.

[23] Jaiswal MK and So HKH. Dual-mode double precision / two-parallel single precision floating-point multiplier architecture. In *IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC)*. New York: IEEE, pp. 213–218. DOI: 10.1109/VLSI-SoC.2015.7314418.

[24] Lei K and Xiao-Ying Y. Design and implementation for quadruple precision floating-point multiplier based on fpga with lower resource occupancy. In *5th International Conference on Intelligent Systems Design and Engineering Applications (ISDEA)*. New York: IEEE, pp. 326–329. DOI: 10.1109/ISDEA.2014.80.

[25] Tomar GS and George M. Modified binary multiplier architecture to achieve reduced latency and hardware utilization. *Wireless Personal Communication* 2018; 98(4): 3554–3561.

[26] George M and Tomar GS. Hardware design procedure: Principles and practices. In *5th International Conference on Communication Systems and Network Technologies*. New York: IEEE, pp. 834–838. DOI: 10.1109/CSNT.2015.198.

[27] Havaldar S and Gurumurthy KS. Design of vedic ieee 754 floating-point multiplier. In *IEEE International Conference on Recent Trends in Electronics, Information & Communication Technology (RTEICT)*. New York: IEEE, pp. 1131–1135. DOI: 10.1109/RTEICT.2016.7808008.

[28] Anane N, Bessalah H, Issad M et al. Hardware implementation of variable precision multiplication on fpga. In *4th International Conference Design & Technology of Integrated Systems in Nanoscale Era*. New York: IEEE, pp. 77–81. DOI: 10.1109/DTIS.2009.4938028.

[29] Manolopoulos K, Reisis D and Chouliaras VA. An efficient multiple precision floating-point multiplier. In *18th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*. New York: IEEE, pp. 153–156. DOI: 10.1109/ICECS.2011.6122237.

[30] Mangalath NS, Priya RA and Malathi P. An efficient universal multi-mode floating-point multiplier using vedic mathematics. In *International Conference on Advances in Communication and Computing Technologies (ICACACT)*. New York: IEEE, pp. 1–4. DOI: 10.1109/EIC.2015.7230724.

[31] Liu D, Wang M, Wang Y et al. A multi-functional floating-point multiplier. In *IEEE 9th International Conference on Anti-counterfeiting, Security, and Identification (ASID)*. New York: IEEE, pp. 56–60. DOI: 10.1109/ICASID.2015.7405661.

[32] Jaiswal MK and So HKH. Architecture for quadruple precision floating-point division with multi-precision support. In *IEEE 27th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*. New York: IEEE, pp. 239–240. DOI: 10.1109/ASAP.2016.7760807.

[33] Diniz P and Govindu G. Design of a field-programmable dual-precision floating-point arithmetic unit. In *International Conference on Field Programmable Logic and Applications*. New York: IEEE, pp. 1–4. DOI: 10.1109/FPL.2006.311302.

[34] Hickman B, Krioukov A and Schulte M. A parallel ieee p754 decimal floating-point multiplier. In *25th International Conference on Computer Design*. New York: IEEE, pp. 56–62. DOI: 10.1109/ICCD.2007.4601916.

[35] Vazquez A, Antelo E and Montuschi P. A new family of high-performance parallel decimal multipliers. In *18th IEEE Symposium on Computer Arithmetic (ARITH '07)*. New York: IEEE, pp. 195–204.

[36] Cowlishaw M. Densely packed decimal encoding. *IEE Proceedings – Computers and Digital Techniques* 2002; 149(3): 102–104.

[37] Baesler M, Voigt SO and Teufel T. An ieee 754-2008 decimal parallel and pipelined fpga floating-point multiplier. In *International Conference on Field Programmable Logic and Applications*. New York: IEEE, pp. 489–495. DOI: 10.1109/FPL.2010.98.

[38] Kuang SR, Wang JP and Hong HY. Variable-latency floating-point multipliers for low-power applications. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 2010; 18(10): 1493–1497. DOI: 10.1109/TVLSI.2009.2025167.

[39] Jaiswal MK, Ray C and Cheung C. Area-efficient fpga implementation of quadruple precision floating-point multiplier. In *IEEE 26th International Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW)*. New York: IEEE, pp. 376–382. DOI: 10.1109/IPDPSW.2012.46.

[40] Abraham S, Kaur S and Singh S. Study of various high speed multipliers. In *International Conference on Computer Communication and Informatics (ICCCI)*. New York: IEEE, pp. 1–5. DOI: 10.1109/ICCCI.2015.7218139.

[41] Vyas K, Jain G, Maurya VK et al. Analysis of an efficient partial product reduction technique. In *International Conference on Green Computing and Internet of Things (ICGCIoT)*. New York: IEEE, pp. 1–6. DOI: 10.1109/ICGCIoT.2015.7380417.

[42] Arish S and Sharma RK. An efficient binary multiplier design for high speed applications using karatsuba algorithm and urdhva-tiryagbhyam algorithm. In *Global Conference on Communication Technologies (GCCT)*. New York: IEEE, pp. 192–196. DOI: 10.1109/GCCT.2015.7342650.