RESEARCH ARTICLE OPEN ACCESS

Design and Implementation of Decimal Floating-Point Multiplier for Multimedia Applications

Anusha Peluri[1], S Srilali[2], B Subrahmanyeswara Rao[3]
SwarnandhraCollegge of Engineering & Technology[1,2,3]

Abstract-Decimal floating point (DFP) arithmetic is important in various applications such as currency conversion, billing, insurance, banking etc., as it is able to produce precise decimal fractions and minimize manual calculations that perform decimal rounding. But binary floating-point arithmetic fails to provide correct decimal rounding and exact decimal fractions such as 0.10,0.0418. A multiplier is one of the main components in most digital and high-performance systems such as digital signal processors, microprocessors and Finite Impulse Response (FIR) filters etc. As technology advances, many scientists have tried and are trying to design multipliers which provide either of the following- high speed, low power consumption and hence less area or even combination of them in multiplier.

Multiplication is also an important operation in decimal operation. This is aims to implement Binary integer decimal based floating point multiplier using the fastest adder. The maximum time of multiplication is consumed in accumulating the partial products and at the final stage of addition to get the significant product. so the multiplication time can be reduced if the partial products are accumulated with the help of fast adders. In this, the binary encoding for DFP numbers also known as Binary Integer Decimal (BID) format has also been implemented.where binary radix of 2 of 2 is used. Since BID encoding stores the significant as an unsigned binary integer for effective reuse of existing binary hardware. In this thesis, a hardware design is presented that multiplies BID encoded DFP numbers. An optimized technique that in parallel with significant multiplication is used to detect if rounding is needed and to find the number of product digits that are needs to be rounded. In this, both for significant multiplication and rounding, a single binary hardware along with carry save feedback is used. To design a BID multiplier, the partial products are generated using radix-8 algorithm, then array multiplier are used to accumulate partial products and different adders like ripple carry adder, carry select adder or carry look ahead adders are used at final stage to obtain the result. Then multiplier is synthesized and simulated using Xilinx ISE 14.5 targeting Spartan 6 FPGA device. Then their results in the terms of area and delay are compared for BID multiplier using different adders.

Index Terms-Binary Integer Decimal (BID),Decimal Floating Point (DFP),Finite Impulse Response (FIR), Floating-Point, Multimedia, Multiplication.

Date of Submission: 22-10-2019 Date Of Acceptance: 06-11-2019

I. INTRODUCTION

Multipliers play a key role in many high-performance systems. As the technology advances, the demand of high- speed digital systems increase, or we can say the demand of high-speed multiplier increases because the multiplier is a main component in every digital system. Multipliers are used as small blocks in large digital systems like FIR filters, microprocessors, digital signal processors, communication systems etc. So, to find performance of a large digital system is measured by the performance of the multiplier because the multiplier is mainly the slowest element in the large systems; consequently, multiplication dominates the execution time of most DSP algorithms, so a high-speed multiplier is always needed. In comparison to other operations in an ALU, the multiplier also uses more power[1]. Thus, scientists have always been trying to design a multiplier which utilizes less power, time and area and increases speed. Multiplication is such a mathematical operation in which a numberi.e. multiplicand is added to itself a given number of times as indicated by another number i.e. multiplier to form a result.

A digital circuit used to multiply two numbers is called a multiplier. As in mathematics, multiplication includes shifting and adding the partial products. The same approach is used here in digital multipliers. For unsigned number multiplication, AND gates are utilized for generating the partial products and full adders are utilized for adding those generated partial products[2], [3]. To multiply signed numbers, the negative numbers should be first converted into its 2‟s complement representation to make all the partial products positive. Digital multipliers can be classified into three types i.e. parallel, serial and serial-parallel multiplier. Both the inputs are entered serially in a serial multiplier. Such an implementation will be led to lesser area and lesser hardware cost and also lower power consumption. But the major drawback is its poor speed, because the inputs are entered serially. Now the speed can be increased by using parallel multiplier implementation because operation is carried out in parallel. Butit is more complex as compared to serial multipliers as it occupies larger area. Also, its power consumption is higher. Parallel multipliers further can be classified into two types i.e. array multipliers[4]. For carrying out fast multiplication. To take benefit of small area of serial multiplier and high-speed operation of parallel multiplier, serial-parallel multipliers are used. In serial-parallel multipliers, one operand is entered serially while other is stored in parallel. This requires less area and enhances the speed of the multiplier. The numbers can be represented in two ways fixed point and floating-point representations. The representation in which digits after and before the decimal is fixed is called fixed point number representation. But the decimal point can float in floating point representation, hence given the name floating point[5], [6]. To represent extremely large and small values like distance between sun and earth or mass of electron, floating point numbers are used. To achieve better accuracy and larger range floating point representation is used although it is slower than fixed point representation.

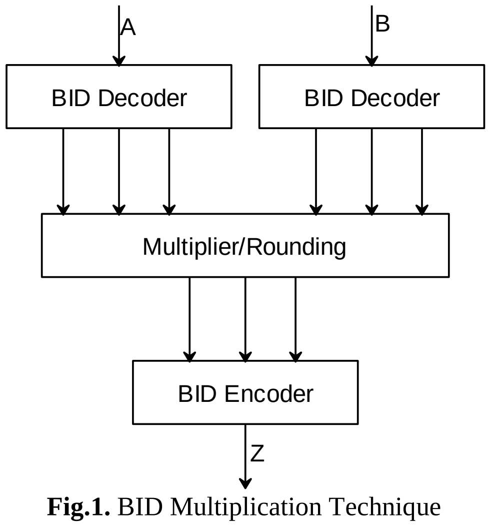

Fig.1. BID Multiplication Technique

Decimal floating point (DFP) number systems can be utilized to represent a wide variety of decimal numbers and do manual calculations that perform decimal rounding. However advanced PC‟s performs binary arithmetic, which have imperfections to represent and round decimal numbers i.e. it can neither give correct decimal rounding nor precise represent many decimal fractions. Due to which, errors from Binary floating point (BFP) arithmetic can combine to form a yearly billing error of over $6 million for a vast billing system. So, a variety of business applications that can't endure errors because of BFP arithmetic have traditionally utilized software to perform DFP calculations. As the interest for decimal floating-point arithmetic is growing, the IEEE P754 Draft Standard for floating point arithmetic incorporates details for decimal floating-point arithmetic. For decimal floating-point numbers, two encodings are specified by IEEE P754 Draft. One decimal floating point (DFP) encoding is specified in IEEE P754. It represents its mantissa as a binary integer and is called as binary integer decimal encoding. This encoding reuses the hardware of existing high-speed binary arithmetic circuits. Second, densely packed decimal encoding is used. But it requires more expensive hardware. A floating-point multiplier has three main units: exponent unit, sign unit, mantissa unit, which works in parallel and a normalization unit.

As scientists always tried to achieve fast speed, low power, small equipment's in any equipment. To achieve fast speed, the delay from input can be decreased. To reduce area, sizes of transistors can be reduced. And power dissipation will automatically decrease with area. But there is a trade-off between these parameters, as we know we cannot improve all of them simultaneously. To estimate performance of a system, sometimes power delay product parameter is used.

II. LITERATURE REVIEW

S.G. Navarro et.al [7] presented the first complete design of a Binary Integer Decimal (BID)-based Decimal Floating Point (DFP) multiplier to achieve effectively adjusted results when two IEEE 754-2008 decimal64 numbers are multiplied. This multiplier works on Binary Integer Decimal (BID) encoded decimal floating-point (DFP) numbers[8]. This design showed that BID multiplication can be proficiently executed in hardware rather than a software implementation. This multiplier can likewise be shared to perform BFP multiplication. This design has variable latency to take advantage of the fact that multiplication results are not often adjusted. For significant multiplication and rounding, a single binary multiplier along with carry save feedback was utilized to reduce the area and critical path delay. Optimizations have decreased the BID multiplier's critical path delay and area.

C. Tsen et.al[9] proposed a hardware design for a rounding unit for 64-bit DFP numbers that utilizes the Binary integer decimal encoding (BID). This design evaluates area, critical path delay and latency for combinational and pipelined designs. A 55-bit by 54-bit binary multiplier consumed more than 86% of the rounding unit‟sarea, which further could be shared with a double-precision binary floating-point multiplier.

S.G. Navarro et.al [7] presented an IEEE P754-compliant multiplier which works on those values that utilize the binary encoding of DFP numbers which is also called as the Binary Integer Decimal (BID) encoding. A single high-speed binary hardware was utilized by this multiplier that has variable latency and enhanced for the basic case in which the product did not should be rounded. In this multiplier, an optimized method was utilized in parallel with the significant multiplication that finds if rounding is required and calculates the number of product digits that should have been adjusted. In this design, to multiply the significands and round the product, a single multiplier is used.

B.J. Hickmann et.al [10] introduced a parallel decimal floating-point multiplier. This parallel DFP multiplier provides low latency and high throughput. This design utilized alternate decimal digit encodings to decrease area and delay.

Ganesh et.al[11] proposed two plans for fixed-point decimal multiplication that utilized decimal carry-save addition to decrease the critical path delay. Initial, multiplier that utilizes decimal carry-save addition in the iterative portion of the configuration and stores a smaller number of multiplicand multiples was displayed.

Y. Xie et.al [12] proposed a new redundant booth encoding scheme, in which the idea was to polarize two adjacent booth encoded digits to directly form an RB partial product to avoid the hard multiple of high radix booth encoding without incurring any correction vector.

Nguyen et.al[13] proposed a hardware design of a joined decimal and binary floating- point multiplier complaint with IEEE P754-2008 Floating-point Standard.

C. Tsen et.al [14] introduced an innovative algorithm and hardware design of a DFP adder. This adder performs subtraction and addition on 64-bit operands that utilized the IEEE P754-2008 binary encoding of decimal floating-point numbers. This BID adder has utilized a hardware component for decimal digit counting and an improved version of formerly published BID rounding unit. They established that a BID-based DFP adder design can be accomplished with a little area increase in comparison to a single 2-stage pipelined fixed-point multiplier.

Liang-Kai Wang et.al [15] displayed novel designs for a decimal floating-point multifunction unit and a decimal floating-point adder. To decrease their delay, both the multifunction unit and the adder utilized decimal injection-based rounding, another type of decimal operand arrangement and a quick flag-based technique for adjusting and overflow detection.

Fahmy et.al [16] presented new algorithms and properties which are utilized as a part in a software implementation of the IEEE P754 decimal floating-point arithmetic, with emphasis on utilizing binary operations effectively.

P. Gurjar et. al.[17] described the manufacturing of high-speed adder circuit using Hardware Description Language (HDL). The purpose behind this is that an adder is a very critical building block of arithmetic and plays an important role in determining the performance of central processing unit (CPU).

Gargaveet. al. [18] presented a proficient implementation of an IEEE P754 single precision floating point multiplier. This multiplier handles both the overflow and underflow cases. To give more precision to the multiplier in a multiply and accumulate (MAC) unit rounding could not be implemented.

III. PROPOSED MULTIPLICATION TECHNIQUE

The basic algorithm to multiply BID numbers to understand the hardware implementation of BID multiplier is as follows.

A DFP number consists of a triple (S, E-bias, C) where S is the sign bit, E is the biased exponent and bias is a constant value which makes E non-negative and C is the significant. For e.g.: - A and B being two IEEE P754-2008 input operands have

(AS, AE, AC) and (BS, BE, BC) as their triples respectively.

To start the operation, firstly to extract sign, biased exponent and significant, the BID encoded operands are unpacked then Ac and BC are multiplied using binary multiplication and intermediate product IPC is obtained. Simultaneously the exponents are added, and bias is removed, and intermediate exponent is produced, and two sign bits are together to produce the final sign bit. In the next step, IPC is examined to determine if rounding is required and how many more digits are needed to be rounded, If the number of decimal digits in IPC is less than the result precision p, rounding is not needed and multiplication is finished else IPC must be rounded and accordingly IPEXP should be adjusted .In the last stage the sign ,biased exponent and significant of the result are packed to calculate the BID encoded result of the multiplication. For e.g.-Consider the decimal 32 BID encoded operands of A = 5B08BBCB16 and B = 8F00007316as inputs to the multiplication. As discussed above inputs are first decoded to obtain:
A= 0,182 − 101,572363 and B = 1,30 − 101,115

Then the significant are multiplied to produce IPC = 65821745. In parallel the biased exponents are added, and bias is subtracted to obtain IPC = 182 + 30 − 101 = 111. In this IPc contains 8 digits and precision p=7 digits, so rounding off IPC by one digit is necessary. Depending on the rounding mode, the rounded result is either
Z = 1,112 − 101,6582174 = 6582174 × 10^{11} or
Z = 1,112 − 101,6582175 = 6582175 × 10^{11}

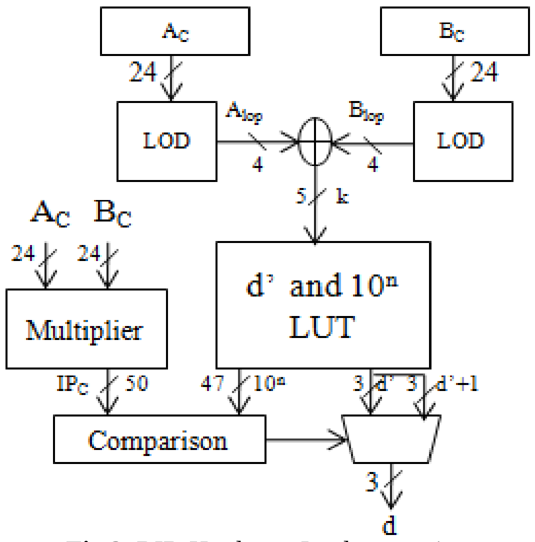

Fig.2. BID Hardawre Implementation

IV. IMPLEMENTATION OF BID MULTIPLIER

Different radix-based algorithm was proposed by BID – encoded decimal32 numbers, out of which radix-8 multiplier is used to reduces the number of partial products. So, in binary multiplier radix 8 multiplier is used to generate the partial products and then carry save adders are used to accumulate the partial products. The binary multiplier produces the product in the form of carry save as

(PC, PS) =M. N+S+C. Here PC is partial carry output and PS is partial sum output. The binary multiplier combines the sum of feedback information (C, S) and carry save partial product (PC, PS) using a 4:2 compressor which further decreases the area and delay of BID multiplication [11]. Figure 2 represents whether rounding is required or not. As already discussed, rounding is needed if IPC is greater than or equal to 107. A direct comparison of IPC to 107 results in higher delay [5]. So, to reduce delay, this BID multiplier utilizes an optimized technique where Atop, B top and a 25-bit adder that adds the 25 least significant bits of PC and PS are used. To add these bits any one of carry select adder, carry look ahead adder or ripple carry adder can be used.

If (Atop+ B top > 23) or if the output of the 25-bit adder is greater than 107, then rounding is done. The 25-bit adder forms the least significant 25 bits of IPC Then the most significant bits of IPC are obtained in parallel with least significant bits ofIPC by using a compound adder. A compound adder adds the most significant 25 bits of PS and PC to produce sum=PS [9: 25] + PC [49 + 25] and sum + 1. Depending on the carry-out from the 25-bit adder, the multiplexer selects the correct sum for the most significant 25 bits of IPC to calculate precisely number of digits to round off, the BID multiplier does the comparison between 10n and IPC = PS + PC using a 50-bit 3:2 compressor. PC + 1, PS and the bitwise negation of 10n are the inputs to 3:2 compressor. This produces two vectors that are then added together, as shown in Fig. 3.10. After this the inspection of sign of the addition of the two vectors is done to determine if IPC > 10^n. The LSB of PC is set to 1 so that PC + 1 can be obtained without any addition, as LSB of a carry vector is normally 0.

This approach has less delay than subtracting 10n from IPC. The exact number of decimal digits to round off is calculated, once it is determined if (IPC > 10^n, as described above.

The rounding is carried out by splitting IPC and Wd into upper and lower 24-bit halves and then obtaining four passes through the 24×24-bit multiplier [4]. By using this approach, both significant multiplication and rounding are performed by same multiplier, Output of multiplier/rounder block is the output significant, biased exponent and sign bit. Then these outputs are given to the BID encoder block that will pack these results and provide the final IEEE P754-2008 result

So, decimal rounding using reciprocal multiplication has been done to remove a division circuit and to reuse the binary multiplier used for BID significant multiplication. Reciprocal multiplication can be seen as multiplication by a pre-calculated and scaled estimation of 10^{-d}.

The method to execute BID rounding comprises of multiplying the number to round by a precalculated estimation of $w_d = 10^{-d}$ to obtain effective division by m = 10^d.

Table 1. Product Fileds

Quotient Field(Q) (variable width(u+1 − v) bits)	Remainder Field(R) (variable width: v bits)	Discarded Field(D) (fixed width: u bits)

Entire algorithm can be described simply by below mentioned steps.

Step1: Inputs A and B are decoded to find (A_S, A_E, A_C) and (B_S, B_E, B_C). Step2: Compute sign: $Z_S = A_S$ XOR B_S

Add Exponents: $IP_C = A_E \times B_E$ − bias Multiply Coefficients: $IP_C = A_C \times B_C$ Find leading one position of A_C: A_{top}

Find leading one position of B_C: B_{top}

Find leading one position of IP_C: $K = A_{top} + B_{top}$

Step3: Finding if rounding is required: round_required= ($IP_C \geq 10^7$) If (round_required) //Find d
$d' = n - p$ //d′ is stored in LUT that is indexed by k
$d = d' + (IP_C \geq 10^7)$ //$10^n$is stored in LUT that is indexed by k
Step4: Execute rounding if required:
If (! round _required) //multiplication operation is finished
$Z_E = IP_E$
$Z_C = IP_C$
Else //rounding required
$TP_C = floor (IPC \times 10^{-d})$ //from $IP_C \times w_d$
Step5: Obtain r* and s* information to get the result Optional increment of IP_C depending on sign, rnd_mode, r* and s*
Z_{C} - tmp = $TP_C + 1$ or $Z_C = TP_C$
Step6: Check whether adjustment is required□

V. RESULTS AND DISCUSSION

After designing the BID based floating point multiplier in verilog HDL, the design is simulated using ISIM simulator. Figure 3 shows the simulation results for BID based floating point multiplier.

Table 2. Operations involved in Multiplication

X_{i+1}	xi	X_i-1	Partial products pp_i	operation on multiplicand
0	0	0	0	Add 0
0	0	1	+Y	Add multiplicand
0	1	0	+Y	Add multiplicand
0	1	1	+2Y	Add 2*multiplicand
1	0	0	-2Y	Subtract 2*multiplicand
1	0	1	-Y	Subtract multiplicand
1	1	0	-Y	Subtract multiplicand
1	1	1	0	Subtract 0

After designing the BID based floating point multiplier in verilog HDL, the design is simulated using ISIM simulator. Figure 5.1 shows the simulation results for BID based floating point multiplier.
Considering the following are the inputs in BID format
a= 00000000111111110000111100001111
b=00110011001100110000011100000000
Depending upon the rounding mode used, the output obtained after calculating the sign, significant and the exponent and after rounding using the steps described above should be
c=00000000111111110000111100001111or001100 110011001100000111100000000

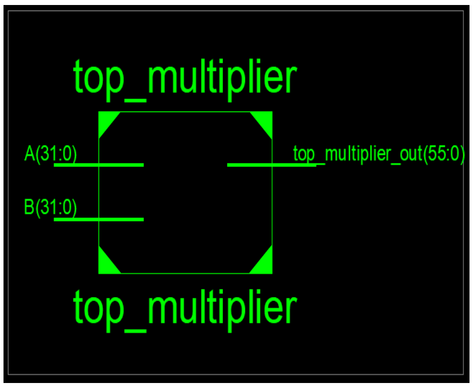

Fig. 3. RTL Schematic Diagram of Multiplier

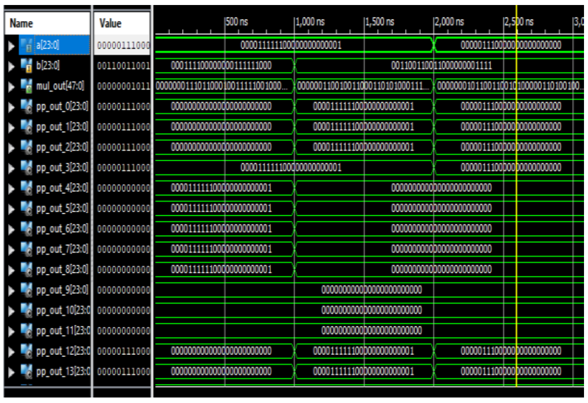

Fig. 4. Simulation Results of Multiplier

Simulation results shown in figure 4 describes the multiplication procedure for two floating point numbers represented in decimal format. Multiplication result is represented using variable Z. In order to get clear understanding multiplication on signed numbers is calculated and shown in figure 4. To get a negative number as result, one positive number and one negative number taken as multiplier and multiplicand. Also both the negative numbers are multiplied to show a positive number as result.

Table 3. Resource Utilization Summary

Resources	Traditional Multiplier	Proposed multiplier
LUT 2	1438	1229
LUT3	92	69
LUT4	339	137
FFS	941	704
MAXIMUM COMBINATIONAL PATH DELAY	112.24ns	93.702ns

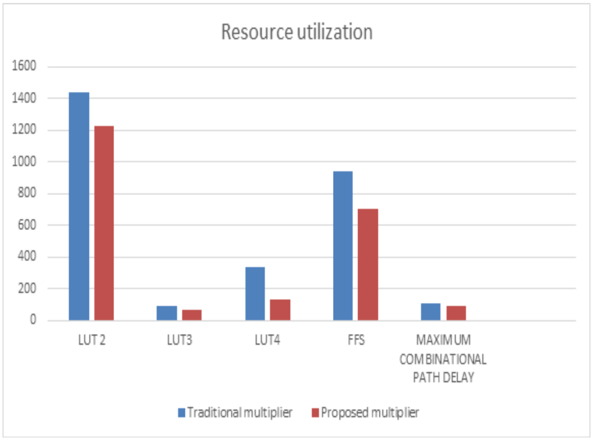

Fig. 5. Graph representing Resource Utilization

VI. CONCLUSIONS

Modified architecture for floating point multiplier is designed and is described in this paper. This research work includes the designing and implementing of a less area utilizing floating point multiplier. Simulation results shows that 19% reduction in size of the chip is observed with the proposed architecture. This leads to reduction in complexity of the system design. Also, the proposed design is of less combinational path delay. Delay reduction is may lead to increase in the processing speed. Entire system is described in Verilog HDL and is simulated using Xilinx ISE. Also physical verification is done using Spartan 3 FPGA.

REFERENCES

[1]. M. K. Jaiswal and H. K.-H. So, “DSP48E efficient floating point multiplier architectures on FPGA,” in *2017 30th International Conference on VLSI Design and 2017 16th International Conference on Embedded Systems (VLSID)*, 2017, pp. 1–6.

[2]. 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)*, 2015, pp. 1–4.

[3]. S. Kim and R. A. Rutenbar, “An Area-Efficient Iterative Single-Precision Floating-Point Multiplier Architecture for FPGA,” in *Proceedings of the 2019 on Great Lakes Symposium on VLSI*, 2019, pp. 87–92.

[4]. D. Peroni, M. Imani, and T. Rosing, “Runtime Efficiency-Accuracy Trade-off Using Configurable Floating Point Multiplier,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2018.

[5]. N. S. Kim, S. Gilani, and M. Schulte, “High efficiency computer floating point multiplier unit.” Google Patents, 13-Dec-2016.

[6]. M. Imani, D. Peroni, and T. Rosing, “Cfpu: Configurable floating point multiplier for energy-efficient computing,” in *Proceedings of the 54th Annual Design Automation Conference 2017*, 2017, p. 76.

[7]. S. G. Navarro and J. Hormigo, “New Results on Non-normalized Floating-point Formats,” *IEEE Transactions on Computers*, 2019.

[8]. A. K. Gupta, A. Singh, and V. Yadav, “VLSID 2014.”

[9]. P. T. P. Tang, E. Schneider, and C. Tsen, “A Software Implementation of the IEEE 754R Decimal Floating-Point Arithmetic Using the Binary Encoding Format,” *Computer Arithmetic: Volume III*, vol. 3, p. 411, 2015.

[10]. W. E. Ferguson, B. J. Hickmann, and T. D. Fletcher, “Method, apparatus, system for single-path floating-point rounding flow that supports generation of normals/denormals and associated status flags.” Google Patents, 22-Sep-2015.

[11]. B. S. Ganesh, J. E. N. Abhilash, and G. R. Kumar, “Design and Implementation of Floating Point Multiplier for Better Timing Performance,” *International Journal of Advanced Research in Computer Engineering & Technology*, vol. 1, no. 7, 2012.

[12]. L. Fang, B. Li, Y. Xie, and H. Chen, “A Unified Reconfigurable CORDIC Processor for Floating-Point Arithmetic,” 2018.

[13]. T. D. Nguyen and J. E. Stine, “A combined IEEE half and single precision floating point multipliers for deep learning,” in *2017 51st Asilomar Conference on Signals, Systems, and Computers*, 2017, pp. 1038–1042.

[14]. C. Tsen, E. M. Schwarz, and M. J. Schulte, “A survey of hardware designs for,” *Computer Arithmetic: Volume III*, vol. 3, p. 437, 2015.

[15]. L. Wang *et al.*, “Floating-Point Multiply-Add with Down-Conversion.” Google Patents, 12-Oct-2017.

[16]. H. A. H. Fahmy, “Decimal Floating Point Number System,” in *Embedded Systems Design with Special Arithmetic and Number Systems*, Springer, 2017, pp. 89–111.

[17]. P. Palsodkar and A. Gurjar, “Fused Floating Point Arithmetic Unit for Radix 2 FFT Implementation,” *IOSR Journal of VLSI and Signal Processing*, vol. 6, no. 2, pp. 58–65, 2016.

[18]. S. Gargave, Y. Agrawal, and R. Parekh, “Single-Precision Floating Point Matrix Multiplier Using Low-Power Arithmetic Circuits,” in *Advances in Power Systems and Energy Management*, Springer, 2018, pp. 683–691.