Decimal Floating Point Multiplication using RPS Algorithm

Rekha K James
Cochin University of Science and Technology
Kochi, Kerala
India

K Poulose Jacob
Cochin University of Science and Technology
Kochi, Kerala
India

Sreela Sasi
Gannon University
Erie
PA
USA

ABSTRACT
Floating-point representation can support a much wider range of values over fixed point representation. The performance of decimal floating-point operations is an important measure in many application domains such as financial, commercial, and internet-based computations. In this research, an iterative decimal floating-point multiplier design in IEEE 754-2008 format is proposed. This design uses a decimal fixed point multiplier using RPS algorithm that generates partial products for column accumulation from the least significant end in an iterative manner. It also incorporates the necessary decimal floating-point exponent processing, rounding and exception detection capability. The rounding process is initiated in parallel with the decimal fixed point multiplication of significand digits. The intermediate exponent, the product sign, sticky bit, round digit and the guard digit are determined on the fly with the accumulation of partial products. Simulation result for a 32-bit data in comparison with the existing designs in literature gives a delay reduction of 25.12%.

General Terms
Decimal arithmetic, VLSI Design.

Keywords
Decimal Floating Point Multiplier, RPS Algorithm, Rounding Logic

1. INTRODUCTION
The majority of the world's commercial and financial data are stored and manipulated in decimal form. Currently, general purpose computers do decimal computations using binary arithmetic. Binary data can be stored efficiently and manipulated very quickly on two-state computers. However, there are compelling reasons to consider decimal arithmetic, particularly for business computations. The reasons include human's natural affinity for decimal arithmetic, and the inexact mapping between some decimal and binary values. Binary floating-point values can only approximate certain common decimal numbers. For example a value of 0.1 requires an infinitely recurring binary pattern of zeros and ones. When an average user performs decimal addition of 0.1 and 0.9, the result is 1.0. If the decimal addition is performed in binary, the result may be 0.99, due to error generated by the decimal to binary conversion. In this world of precision, such errors are no more tolerable. In many cases, the law requires that results generated from financial calculations performed on a computer should exactly match with those carried out using pencil and paper. This is possible only if the calculations are done in decimal. Recently, support for decimal arithmetic has received increased attention due to the growing importance in financial analysis, banking, tax calculation, currency conversion, insurance, telephone billing and accounting. Hardware support for decimal operations, however, has been limited. The scenario is set to change with the continually dropping cost, and with the significant speedup achievable in hardware. This leads to the design of processors that will support Decimal Floating Point (DFP) arithmetic in near future.

Several hardware designs using IEEE 754–2008 standard [1] for DFP multiplication have been proposed in literature. The DFP multiplier in [2] makes use of decimal carry save adders [3] for Decimal Fixed Point (DFxP) multiplication of its significand digits. The design of DFP multipliers whose partial product accumulation is based on non-pipelined iterative implementation using decimal carry save adders is presented in [4]. A combined decimal and binary floating-point multiplier is presented in [5]. To attain high speeds, parallel multipliers are used at the expense of area. A parallel DFP multiplier using parallel DFxP multiplier is presented in [6, 7]. Parallel designs are adopted when latency and throughput are considered more important than area.

In most of the iterative DFP multipliers published so far, the entire multiplicand is multiplied by one multiplier digit to generate a partial product in each cycle. An alternative approach is to generate the partial product using single digit multipliers [8, 9]. The single digit multiplier in [8] uses a standard 4 × 4 unsigned binary multiplier for generating an 8-bit binary output that needs to be corrected to two decimal digits. A faster multiplier design for single digit decimal multiplication, with reduced critical path delay and area is proposed in [9]. A DFxP multiplication using RPS algorithm is implemented in [10] using single digit multipliers of [9]. RPS algorithm generates partial products that are needed for column accumulation from the least significant end in an iterative manner.

The DFP multiplier design presented in this research utilizes the DFxP multipliers of [10]. It also incorporates the necessary DFP exponent processing, rounding and exception detection capability. The rounding process is initiated in parallel with the DFxP multiplication of significand digits. The intermediate exponent, the product sign, sticky bit, round digit and the guard digit are determined in parallel with the generation and accumulation of partial products.

The organization of the paper is as follows: Initially, the background information on IEEE 754–2008 DFP formats are presented. Then the design for DFP multiplication is discussed. This is followed by the descriptions of intermediate exponent, rounding and exception handling. Simulation is done for 32-bit DFP data, and synthesized using Leonardo Spectrum from Mentor Graphics using ASIC Library. This result is then compared for area and delay with the existing design of [4].

2. DFP FORMATS

The IEEE 754–2008 standard specifies DFP formats of 3 representations: 32-bit format with 7 significand digits, 64-bit format with 16 significand digits and 128-bit format with 34 significand digits. These encodings allow a range of positive and negative values, together with values of ±0, ±Infinity, and Not-a-Number (NaN). In IEEE 754-2008, the value of a finite DFP number ‘ v ’with an integer significand is given as

$$v = (-1)^s \times C \times 10^q$$

where ‘S’ is the sign, ‘q’ is the unbiased exponent, and ‘C’ is the significand portion. Figure 1 shows the DFP format.

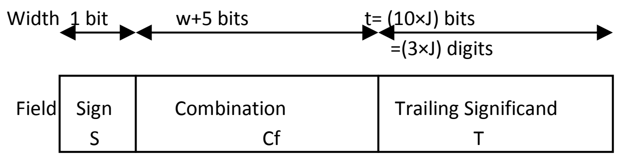


Fig. 1. Decimal Floating Point (DFP) format

The 1-bit sign field, ‘S’ indicates the sign of a number. The (w+5) bits combination field shown as ‘Cf’ in Figure 1 provides the Most Significant Digit (MSD) of the significand portion, and the biased exponent, ‘E’. The exponent encoded in binary is a non-negative number, in the range ‘0’ through ‘E_{limit}’, from which the exponent parameter is calculated by subtracting a bias. The ‘Cf’ field also has special values, such as Not-a-Number (NaN) and infinity (∞). The remaining digits of the significand portion are specified in the t-bit trailing significand field, ‘T’. The number of bits in the ‘T’ field is an integer multiple of ten, indicated as 10×J where ‘J’ has a value of 2, 5 and 11 in decimal32, decimal64 and decimal128 formats respectively. IEEE 754-2008 specifies two encodings for the trailing significand field: Densely Packed Decimal (DPD) encoding and the Binary Integer Decimal (BID) encoding. This research makes use of DPD encoding that encodes three decimal digits in 10 bits, giving a 20% more efficient encoding than a simple Binary Coded Decimal (BCD). The maximum precision or the maximum length of the significand is denoted as ‘P_{limit}’, which is equal to 7, 16, and 34 digits, for decimal32, decimal64, and decimal128 formats respectively. If the DFxP multiplication operation of the significand portion results in more digits than ‘P_{limit}’ then the result will be rounded.

3. DFP MULTIPLICATION ALGORITHM

The DFP multiplier design presented in this paper extends the iterative DFxP multiplier design published in [10]. The design is shown in Figure 2.The operands encoded in DPD are decoded to get BCD significand digits and binary exponents. Then the product of significand digits of the two operands is generated using RPS algorithm of iterative DFxP multiplication. The Sticky bit (Sb), Round digit (R) and Guard digit (G) generation are done in parallel with this process. Rounding is accomplished by selecting either the product truncated to ‘P_{limit}’ digits or its incremented value. Initiation of the rounding operation along with the DFxP multiplication speeds up the entire process. After rounding, a ‘zero’ at MSD leads to a single digit shift towards left. As evident from literature, the shifter module is a major component that contributes to the critical path delay of a floating point multiplier. So, a second rounding module is used to round the product, excluding the MSD. Then a selection is made from the two rounded results based on MSD. The exponent is adjusted, exceptions are set accordingly, and the result is encoded back in DPD.

The algorithm is explained using a 32-bit DFP multiplication. Initially the two 32-bit operands are read from registers, and decoded to generate 7 significand digits, 8-bit biased exponent (E) and a Sign bit (S). The exceptions such as ‘NaN’ and ‘Infinity’ are also decoded out from the 32-bit input. The output exceptions are set at the logic block named ‘Handle exception 1’ depending on the input exceptions. There are four exceptions that may be signaled during multiplication: Invalid operation, Overflow, Underflow, and Inexact. The exponent is computed, and the sign bit of the result is determined as the XOR combination of sign bit of the two operands. The significand digits are multiplied using a DFxP multiplier. A 7 digit × 7 digit DFxP multiplier is used for 32-bit DFP input. This DFxP multiplier makes use of the RPS algorithm of [10] that generates the final product (FP) in ‘n+1’ cycles. Figure 3 gives the detailed schematic of the blocks included in the dashed box in Figure 2, for a 7 digit × 7 digit DFxP multiplication. The MSD of the significand of each input operand is suggested to be a non-zero number. This leads to a unique representation of the DFP inputs avoiding the need to count the number of leading zeroes. The smallest number that can be represented in this format for a 32 bit representation is 1000000×10^{-101} and the largest number is $9999999\times10^{+90}$. Any number that is greater than the largest number is encoded as ‘Infinity’ with an ‘Overflow’ exception. Similarly any number that is less than the smallest number is truncated to ‘Zero’ with an ‘Underflow’ exception. If the result from an operation needs more digits than ‘P_{limit}’ which is the maximum precision of the significand, then the result will be rounded. If this rounding causes removal of non-zero digits then, the ‘Inexact’ exception is set.

3.1 Rounding

Rounding is required when all the essential digits of the ‘2n’digit product of an ‘n’ digit × ‘n’ digit fixed point multiplication cannot be accommodated in ‘P_{limit}’ digits. This is accomplished by selecting either the product truncated to ‘P_{limit}’ or its incremented value and by using Sticky bit (Sb), Round digit (R) and Guard Digit (G).

3.2 Sticky Bit Generation

The least significant ‘n-2’ digits determine the sticky bit (Sb). It is desirable to generate sticky bit (Sb) on-the-fly with DFxP multiplication to improve the speed of DFP. This can be readily accomplished in this design. The sticky bit (Sb) generation is done in parallel with the DFxP multiplication. It is shown from Figure 3 that for a 7 digit × 7 digit DFxP multiplication, the least significant product digits FP_4- FP_0 are available after the fourth clock cycle. Hence sticky bit (Sb) can be generated after the fourth clock cycle. To generalize, for an ‘n’ digit DFxP multiplication using RPS algorithm, sticky bit generation can be done after $\lceil (n/2) \rceil$cycles.

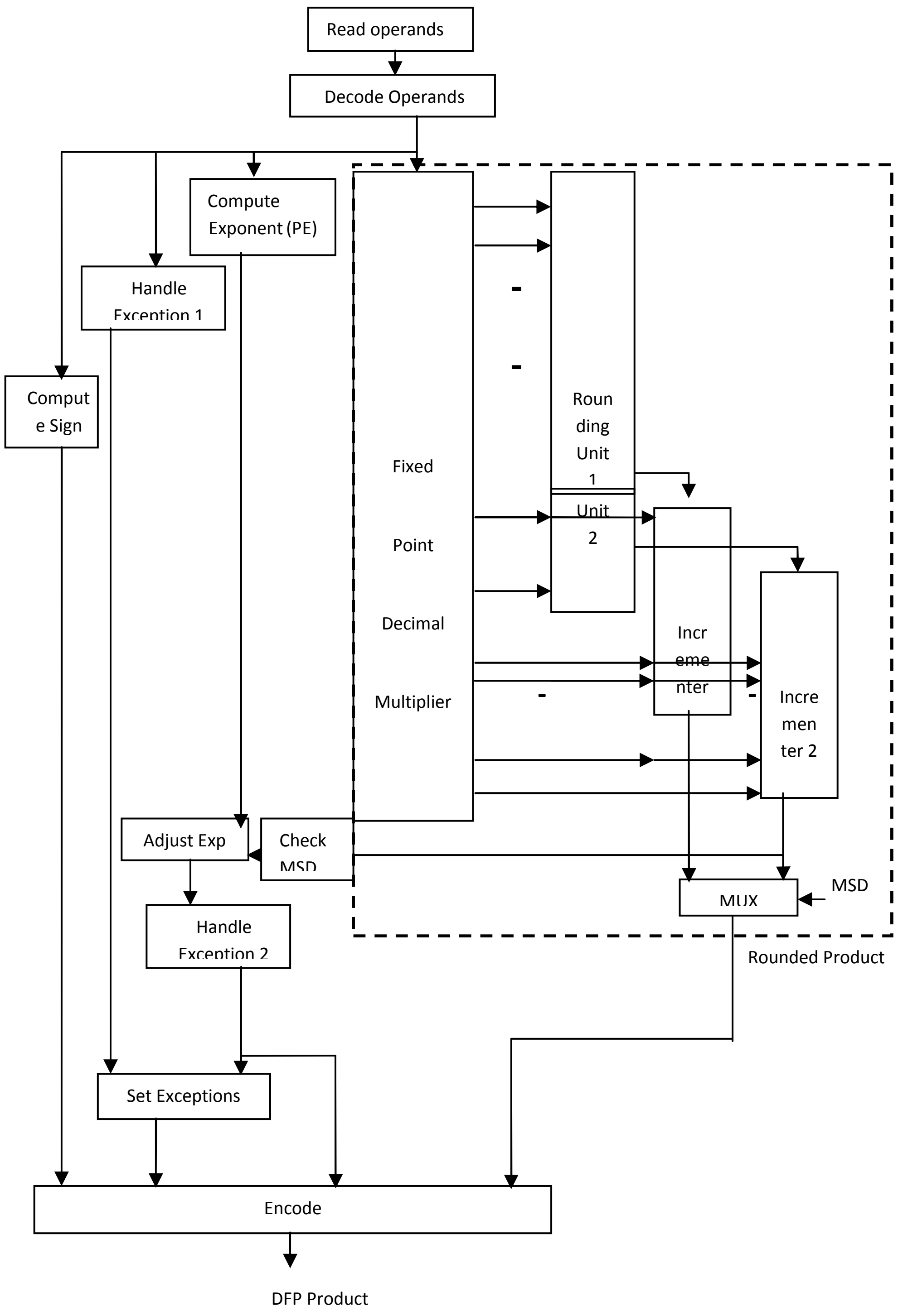


Fig. 2. Decimal Floating point (DFP) Multiplier

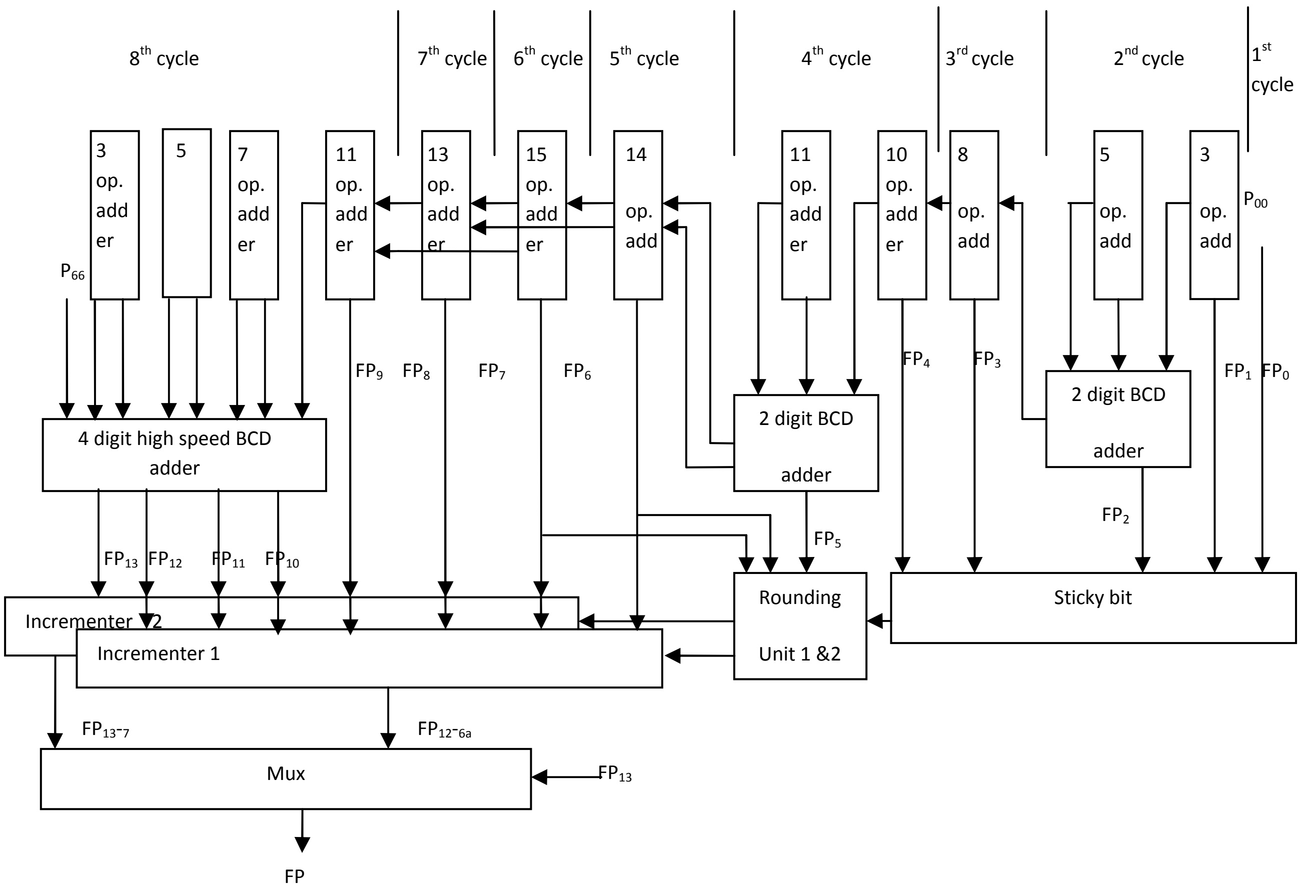


Fig. 3. Block Schematic of DFxP Multiplier and Rounding Unit

3.3 Round digit and Guard digit Generation

In the next two clock cycles, the Round digit (R) and the Guard digit (G) are generated. Rounding has to be done on the most significant 'n' digits. The rounding schemes are tabulated in Table1.

Table 1: Rounding Schemes

Rounding mode	Condition

Round up	G > 5
	G = 5 and R ≠ 0
	G = 5 and Sb ≠ 0

Round down	G < 5
Round to nearest even	G = 5 and R = 0 and Sb = 0

When the final product of the DFxP multiplication is one digit less than the total length, it may be necessary to shift left the product by one digit to make the MSD a non-zero number. The corrective left shift of one digit necessitates maintaining an additional digit to the right of the decimal point. This digit is referred to as the guard digit and is similar to the guard bit used in binary floating-point multiplication. The corrective left shift may lead to an overflow to the (n+1)th digit while rounding the result. This situation is demonstrated below using a 7 digit × 7 digit DFxP multiplier of a 32-bit DFP multiplication.

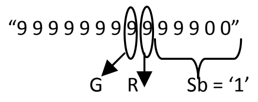
Let C_1 be the significand of operand 1, and C_2 the significand of operand 2.

If C_1 = "3 3 3 3 3 3 0"
and C_2 = "3 0 0 0 0 0 3"

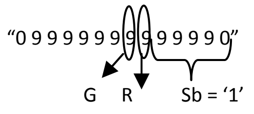
Then the 14 digit result of C1 × C2 =
"0 9 9 9 9 9 9 9 9 9 9 9 9 0"

Since the MSD is zero a left shift is performed to avoid leading zero and the exponent is adjusted accordingly.

Now the new result is



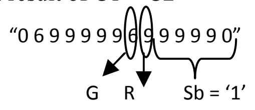
Since G >5, the first 7 digits are to be rounded up, and results in an 8 digit number "10 000 000". This overflow to the eighth digit has to be corrected again. This additional correction or shift can be overcome by doing the shift after rounding. So, in this case the rounding is done initially for the product and the result is shown below.



Even now, since G > 5, the most significant 7 digits are rounded up to get a value “1 000 000”. The MSD of the rounded result is not zero anymore and hence left shift is not required.

Now consider another example.
If C1 = “2 3 3 3 3 3 0” and C2 = “3 0 0 0 0 0 3”

The 14 digit result of C1 × C2 =



Since G >5, the first 7 digits are rounded up, and results in a number “070000000”. The MSD of the rounded result is a zero, and hence a left shift is done and the exponent value is adjusted. The shifted result is now “7 0 0 0 0 0 0” if the ‘shifted in digit’ is zero, or a more erroneous result if the ‘shifted in digit’ is the Guard digit (G). Shifting before rounding would have given a result “6 9 9 9 9 9 7”. This result is equivalent to rounding the ‘n’ digits excluding the MSD. It also utilizes the correct purpose of Guard digit. It can be seen that by rounding the result before shifting the error generated is more. So, in such cases shifting has to be done first. In other words, select the result after rounding the ‘n’ digits excluding the MSD. A suitable selection method is required to determine if rounding is to be done before shifting or vice versa.
In the proposed DFP multiplier, rounding is done for both the most significand ‘n’ digits (using Incrementer 2 of Figure 3) and for the ‘n’ digits excluding MSD (using Incrementer 1 of Figure 3). The appropriate result is selected based on the MSD of the Incrementer 2. This improves the accuracy of the result if a left shift is required, and also avoids the need for a shifter module. This in turn reduces the critical path delay. After rounding, the required ‘exponent adjust’ is done and if necessary, exceptions are modified. The final result is then encoded back to DFP format.

4. SYNTHESIS RESULTS

The proposed DFP multiplier was coded for a 32-bit input in VHDL, and synthesized to evaluate the area and delay associated with the design. Synthesis was done using Leonardo Spectrum from Mentor Graphics Corporation with ASIC Library. An area and delay breakdown for an approximate contribution of major components of the design shown in Figure 2 is given in Table 2.

Even though the total area is the sum of area of different stages, the total delay for the complete circuit is only 53.67ns, which is less than the total sum of delays of all stages. This is because of the inherent parallelism in the design. Table 3 shows the synthesis results of DFxP Multiplier and rounding unit of Figure 3.

Table 2: Area and Delay for different stages of DFP Multiplier (7-digit × 7-digit)

	Area	Delay
Component	μm^2	ns
Decoding Logic	**1439**	**3.26**
Exponent generation	**229**	**4.42**
Exception handling	**66**	**4.21**
DFxP multiplier & Rounding Unit	**25481**	**41.8**
Encoding Logic	**600**	**5.25**
Total	**27926**	**53.67**

Table 3: Area and Delay of Rounding Unit and DFxP Multiplier (7-digit × 7-digit)

	Area	Delay
Component	μm^2	ns
DFxP Multiplier	**18089**	**30.2**
Incrementer 1	**3536**	**22.71**
Incrementer 2	**3536**	**22.71**
Mux	**205**	**0.24**
Sticky bit	**51**	**1.15**
Rounding	**64**	**1.96**
Total	**25481**	**41.8**

Here, also the total delay is less than the sum of delays of each component as rounding starts before the DFxP multiplication is completed. This is possible since the sticky bit (Sb), the round digit (R) and the guard digit (G) are generated on-the-fly during the DFxP multiplication process. For a 7 digit × 7 digit DFxP multiplication, the sticky bit (Sb) generation can be done after the fourth cycle; round and guard digit generation is done in the next 2 cycles. The delay break up of different components of DFP Multiplication is shown in Figure 4.

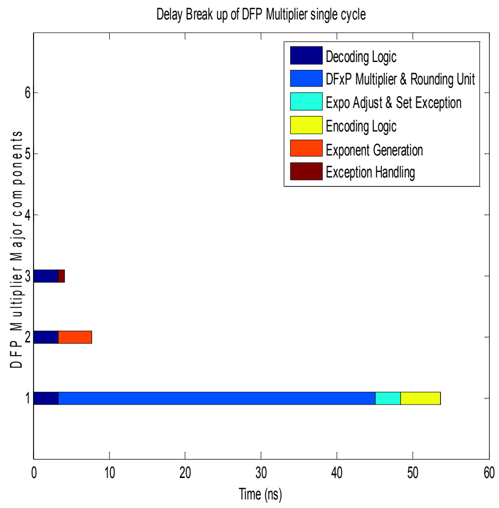
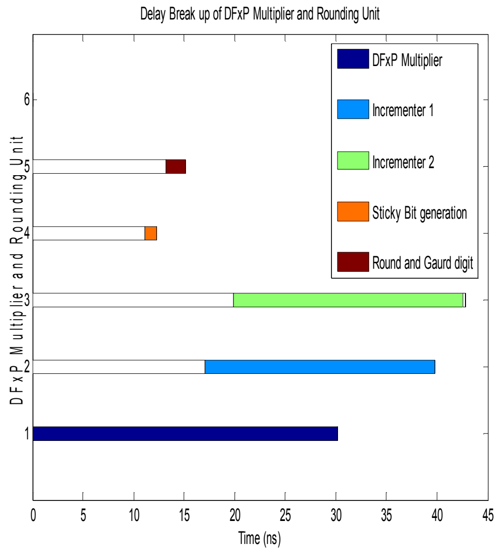


Fig. 4. Delay Break up of DFP Multiplier

Figure 5 gives a detailed delay breakup of the DFxP Multiplier and the rounding unit. The graph showing the delay of a 7 digit

× 7 digit DFxP multiplication for 8 cycles is given in Figure 6. The delay break up for one DFP multiplication of 32-bit inputs is shown in Figure 7. DFP multiplication takes 9 cycles to complete one 32-bit DFP multiplication with worst cycle time being 19.97 ns. Hence, the total delay is 179.73 ns.



Fig, 5. Delay Break up of DFxP Multiplier and Rounding Unit

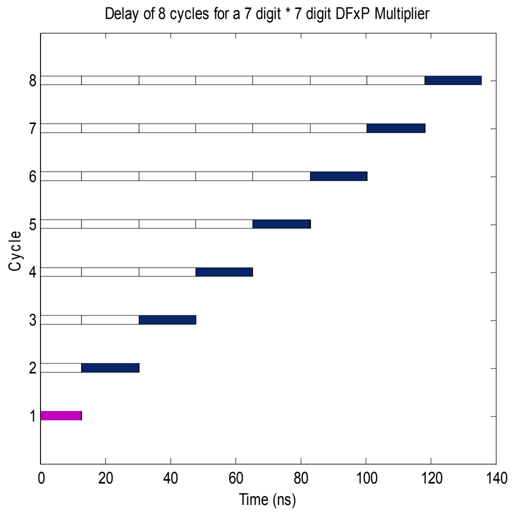


Fig. 6. Graph showing the 'delay' of 7 digit × 7 digit DFxP Multiplier for 8 cycles

When multiplying two DFP numbers with 'n' digit significands, using the proposed approach, the worst case latency is 'n + 2' cycles, and initiation interval is 'n+1' cycles. In other words, a new multiplication can begin every (n+1)th cycle. The final cycle of current set of inputs and the first cycle for next set of inputs are done simultaneously.

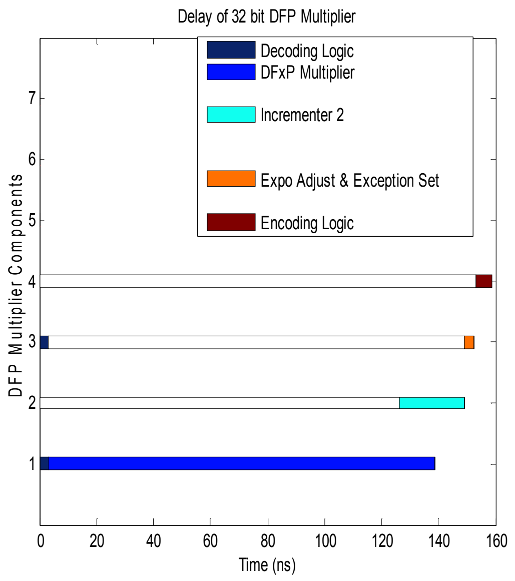


Fig. 7. Delay break up of 32 bit DFP Multiplier

The proposed design differs in partial product generation by using RPS algorithm and in rounding logic from the iterative approach for the DFxP multiplication in [4]. Both these designs are synthesized in the same environment. Table 4 shows the comparisons of the DFxP multiplier designs using design of [10] and that of design of [4]. The design of [10] needs double the area for almost the same latency. But when the design of [10] is used as the DFxP Multiplier of a DFP multiplication, the speed is increased as tabulated in Table 5. This is because the rounding process is initiated before the DFxP Multiplication cycles are over. This parallelism achieved in the proposed DFP multiplier design decreases the delay of the critical path. This in turn reduces the worst cycle time and increases the throughput. A delay reduction of 25.12% is achieved using this approach compared to the design in [4].

Table 4: Comparison of DFxP multipliers (7-digit × 7-digit)

Parameters	RPS	Design of [4]	Ratio
Area (μm^2)	18089	8268	2.18
Delay of 7-digit x 7-digit multiplication (ns)	133.55	130.4	1.024

Extending the iterative DFxP multiplier design to support DFP multiplication affects the area, cycle time, latency, and initiation interval. The area of the DFP multiplier design is 50% more than that of the DFxP multiplier design. The worst case period of the DFP multiplier design is 12% more than that for the DFxP design. The latency of the DFP multiplier is two cycles more than the DFxP multiplier and the dispatch spacing between multiply operations is increased by one cycle.

Table 5: Comparison of DFP Multipliers for 32-bit input

Parameters	RPS	Design of [4]	Ratio
Worst cycle time (ns)	19.97	26.67	0.749
Maximum frequency (MHz)	50	37.5	1.333
Delay of 32-bit DFP multiplication in terms of worst cycle time (ns)	179.73	240.03	0.749

5. CONCLUSION

The iterative DFP multiplier presented in this research includes the floating point extensions to the iterative DFxP multiplier design of [10] in terms of exponent processing, rounding, and exception detection and handling. This DFP multiplier design is in compliance with IEEE 754-2008. The VHDL code for a 32 bit DFP multiplier is synthesized to find the area and delay using Leonardo Spectrum from Mentor Graphics Corporation with ASIC Library. The proposed design and the design in [4] for the DFP Multiplication are synthesized in the same environment. A delay reduction of 25.12% is achieved because of the initiation of rounding process during the DFxP multiplication. This parallelism decreases the delay of the last cycle, which in turn reduces the worst case period and increases the throughput. When multiplying two DFP numbers with 'n' digit significands, using this approach the worst case latency is 'n+2' cycles, and initiation interval is 'n+1' cycles. Novel features of this multiplier include the DFxP multiplication using RPS algorithm, on-the-fly generation of the sticky bit and the initiation of rounding process during the DFxP multiplication.

6. REFERENCES

[1] IEEE Working Group of the Microprocessor Standards Subcommittee, IEEE Standard for Floating-Point Arithmetic. New York: The Institute of Electrical and Electronics Engineers, 2008.

[2] M. A. Erle, M. J. Schulte, and B. J. Hickmann, Decimal Floating-Point Multiplication Via Carry-Save Addition," in 18th IEEE Symposium on Computer Arithmetic, pp. 46{55, IEEE Computer Society, June 2007.

[3] M. A. Erle and M. J. Schulte, "Decimal Multiplication via Carry-Save Addition," in 14th IEEE International Conference on Application-Specific Systems, Architectures, and Processors, pp. 348{358, June 2003

[4] M. A. Erle, B. J. Hickmann and M. J. Schulte, "Decimal Floating-Point Multiplication", IEEE Transactions on Computers, Volume 58 , Issue 7, July 2009, Pages 902-916, ISSN:0018-9340.

[5] Charles Tsen, Sonia González-Navarro, Michael Schulte, Brian Hickmann, Katherine Compton, "A Combined Decimal and Binary Floating-Point Multiplier," 20th IEEE International Conference on Application-specific Systems, Architectures and Processors, 2009, pp.8-15.

[6] H. A. H. Fahmy, R. Raafat, A. M. Abdel-Majeed, R. Samy, Tarek ElDeeb, Y. Farouk, "Energy and Delay Improvement via Decimal Floating Point Units," Proceedings on19th IEEE Symposium on Computer Arithmetic, 2009, pp.221-224.

[7] A. Vazquez, E. Antelo, P. Moutuschi, "Improved Design of High-Performance Parallel Decimal Multipliers", Proceedings of the IEEE Transactions on Computers, Vol. 59 no: 5, May 2010, pp679–693.

[8] G. Jaberipur, A. Kaivani, "Binary-coded decimal digit multipliers", Computers & Digital Techniques, IET Vol. 1,Issue 4, July 2007, pp 377–381

[9] R. K. James, Shahana T. K, K. P. Jacob and S. Sasi, "Decimal Multilpication using Compact BCD Multiplier", International Conference on Electronic Design , ICED Dec 2008, Penang, Malaysia, pp 1-6

[10] R. K. James, Shahana T. K, K. P. Jacob and S. Sasi, " Fixed Point Decimal Multiplication Using RPS Algorithm", International Symposium on Parallel and Distributed Processing with Applications, Dec 2008. ISPA '08, Sydney**,** Australia, pp 343-350