Area and Power Efficient Fused Floating-point Dot Product Unit based on Radix-2^r Multiplier & Pipeline Feedforward-Cutset-Free Carry-Lookahead Adder

M.Madhu Babu[a,b], K.Rama Naidu[b]

[a] *Research Scholar,JNTUA Ananthapuramu,India,515002*
[b] *Department of Electronics and Communication Engineering,JNTUA CEA,India,515002*

Abstract

Fused floating point operations play a major role in many DSP applications to reduce operational area & power consumption. Radix-2^r multiplier (using 7-bit encoder technique) & pipeline feedforward-cutset-free carry-lookahead adder(PFCF-CLA) are used to enhance the traditional FDP unit. Pipeline concept is also infused into system to get the desired pipeline fused floating-point dot product (PFFDP) operations. Synthesis results are obtained using 60nm standard library with 1GHz clock. Power consumption of single & double precision operations are 2.24mW & 3.67mW respectively. The die areas are 27.48 mm^2, 46.72mm^2 with an execution time of 1.91 ns , 2.07 ns for a single & double precision operations respectively. Comparison with previous data has also been performed. The area-delay product(ADP) & power-delay product(PDP) of our proposed architecture are 18%,22% & 27%,18% for single and double precision operations respectively.

Keywords: Pipeline fused floating-point dot product (PFFDP), pipeline feedforward-cutset-free carry-lookahead adder (PFCF-CLA), IEEE Std.754, double-base number system (DBNS)

1. Introduction

In the recent past, demand for the floating point operations has been increasing in DSP (scientific and engineering) applications. Floating-point operations has large dynamic range that overcomes scaling and overflow /underflow problems arises with fixed point operations[1, 2, 3]. IEEE-754 Standard [4] is considered as the base floating-point format to maintain uniformity and has universal acceptance. This format exhibits single and double precision formats with 32/64 bit operations respectively.

Fused floating-point primitive operations have been developed with primary focus on reduction of delay and circuit area. Fused dot product(FDP) operation is an extension of fused multiply-add(FMA) operation [1, 3, 5, 6] . They outperform discrete floating-point adders and multipliers in many aspects like latency [7] & area. The overall optimization in rounding error is an additional add-on to the operation. A single FDP can replace floating-point adder and floating-point multiplier. Many DSP algorithms are rewritten to suite FDP operations.

Based on radix-2^r arithmetic [8, 9, 10, 11], the operational upper limit is $\left\lfloor \frac{(N+1)}{r} + 2^{(r-2)} - 2 \right\rfloor$ where '$\lfloor\ \rfloor$' is the ceiling function i.e $\lfloor 20.12 \rfloor = 21$. The value 'r' can be calculate using $2 \cdot W(\frac{\sqrt{(N+1)\cdot log(2)}}{log(2)})$ where 'W' is Lambert function. For any value of N this upper limit should be lower than $(2 \cdot \frac{N}{log(N)})$. This method is a variant of Pinch's method, where splitting of binary representations using fixed weights are changed to fixed length r.

The remaining paper is organized as follows: Preliminaries of FDP algorithm is given in Section 2. The proposed architecture is elaborately presented along with design of radix-2^r multiplier & design of adder in Section 3. Section 4 deals with performance and discussion of the results. Finally conclusion and future scope of the work are given in Section 5.

2. Preliminaries

2.1. Floating-point DOT Product(FDP) Operation

Generalized expressions for floating-point addition (FPA) & floating-point dot product (FDP) are

$$FPA = \sum_{i-0}^{n} A_i \tag{1}$$

$$FDP = \sum_{i-0}^{n} A_i B_i \tag{2}$$

where A_i,B_i are floating-point operands that can be represented as

$$A_i = (-1)^{S_{ai}}.\, 2^{(E_{ai}-bais)}.\, M_{ai} \tag{3a}$$

Email addresses: madhubabu07.ece@jntua.ac.in (M.Madhu Babu), krnaidu.ece@jnuta.ac.in (K.Rama Naidu)

$$B_i = (-1)^{S_{bi}}.\, 2^{(E_{bi}-bais)}.\, M_{bi} \tag{3b}$$

where S_{ai}, S_{bi} are the sign bits, E_{ai}, E_{bi} are the biased exponents and M_{ai}, M_{bi} are the significant bits. The n-bit FDP can be expanded as

$$\sum_{i-0}^{n} A_i B_i = \sum_{i-0}^{n}[(-1)^{S_{ai}}.\, 2^{(E_{ai}-bais)} M_{ai}] \\ [(-1)^{S_{bi}}.\, 2^{(E_{bi}-bais)} M_{bi}] \tag{4}$$

That can be written as

$$\sum_{i-0}^{n} A_i B_i = \\ \sum_{i-0}^{n}[(-1)^{S_{ai}\wedge S_{bi}}.2^{(E_{ai}+E_{bi}-2bais)}(M_{ai}M_{bi})] \tag{5}$$

where $^{S_{ai}\wedge S_{bi}}$, $^{E_{ai}+E_{bi}}$ & $M_{ai} \times M_{bi}$ are represented as $^{S_{ci}}$, $^{E_{ci}}$ & M_{ci} respectively. "$\wedge$" represents xor operation. The resultant value will be

$$\sum_{i-0}^{n} A_i B_i = \sum_{i-0}^{n}[(-1)^{S_{ci}}.\, 2^{(E_{ci}-2bais)}.\, M_{ci}] \tag{6}$$

2.2. Review of Error Analysis

Floating-point computations suffer from two types of errors: propagation error and rounding error. Propagation error deals with input data and rounding error occurs due to rounding of end results of performed operations [3].

The propagation error can be defined as

$$x(a,b) = x(\widehat{a},\widehat{b}) + \\ \frac{\partial x(\widehat{a},\widehat{b}))}{\partial a}(a-\widehat{a}) + \frac{\partial x(\widehat{a},\widehat{b}))}{\partial b}(b-\widehat{b}) \tag{7}$$

where a,b are precised values and $\widehat{a}, \widehat{b}$ are floating values. By using Lagrange's Mean-value theorem Eq.7 is modified as

$$\varepsilon_{prop} = \frac{\left|x(a,b) - \widehat{x}(\widehat{a},\widehat{b})\right|}{x(a,b))} \\ \approx \frac{x'(\widehat{a},\widehat{b})\widehat{a}}{x(\widehat{a},\widehat{b})}\varepsilon_a + \frac{x'(\widehat{a},\widehat{b})\widehat{b}}{x(\widehat{a},\widehat{b})}\varepsilon_b = M_a\varepsilon_a + M_b\varepsilon_b \tag{8}$$

Here M_a, M_b are amplification factors. $\varepsilon_a, \varepsilon_b$ are error deviations w.r.to a & b. They are purely based on the type of operation performed (addition / multiplication). Amplification factors for floating-point addition operations are as follows

$$M_a = \frac{x'(\widehat{a},\widehat{b})\widehat{b}}{x(\widehat{a},\widehat{b})} = \frac{\widehat{b}}{\widehat{a}+\widehat{b}} \tag{9a}$$

$$M_b = \frac{x'(\widehat{a},\widehat{b})\widehat{b}}{x(\widehat{a},\widehat{b})} = \frac{\widehat{a}}{\widehat{a}+\widehat{b}} \tag{9b}$$

The propagation error amplification factors for floating-point multiplication operation can be written as

$$M_a = \frac{x'(\widehat{a},\widehat{b})\widehat{a}}{x(\widehat{a},\widehat{b})} = \frac{\widehat{ab}}{\widehat{ab}} = 1.0 \tag{10a}$$

$$M_b = \frac{x'(\widehat{a},\widehat{b})\widehat{a}}{x(\widehat{a},\widehat{b})} = \frac{\widehat{ab}}{\widehat{ab}} = 1.0 \tag{10b}$$

The rounding error is defined as overall error of a floating-point operation.The relevant formula is derived as follows, where the floating-point significant precious value is given by

$$z = (1.0 + p_1 2^{-1} + p_2 2^{-2} + .. + p_n 2^{-n} + \\ p_{n+1}2^{-(n+1)} + ... + p_{n+22}2^{-(n+22)} + p_{n+23}2^{-(n+23)})2^e \tag{11}$$

The floating-point representation is

$$\widehat{z} = (1.0 + p_1 2^{-1} + p_2 2^{-2} + ... + p_n 2^{-n})2^e \tag{12}$$

So the rounding error will be

$$\varepsilon_{round} = \frac{z-\widehat{z}}{z} \\ = \frac{(p_{n+1}2^{-(n+1)} + ... + p_{n+23}2^{-(n+23)})}{(1.0 + p_1 2^{-1} + p_2 2^{-2} + ... + p_{n+23}2^{-(n+23)})} \tag{13}$$

Equations(8) & (13) are considered as propagation and rounding errors respectively. Combination of both leads to overall error for any floating-point arithmetic model.

In fused floating-point dot product unit overall error can be calculated by using

$$E_{Gn} = 3.\, \varepsilon_{prpo} + 3.\, \varepsilon_{round} \tag{14}$$

$$E_{FDP} = 3.\, \varepsilon_{prpo} + \varepsilon_{round} \tag{15}$$

From Eqs.(14)&(15) we can conclude that rounding error of FDP unit is one-third of rounding error of discrete operations.

3. Proposed Architectures

Figure 1 shows enhanced version of traditional FDP [12] with four stage pipeline concept, where the focus is on parallel multiplier and PFCF-CLA adder. The remaining stages are alignment, 2's compliment, leading zero anticipatory(LZA) and finally rounding & normalization operations. LZA is pre-corrected operand to calculate the number of leading zeros. LZA is composed of two vectors computation followed by leading zero detector(LZD)[13]. In order to make this fused FDP much faster pipeline concepts are implemented, by replacing traditional ripple

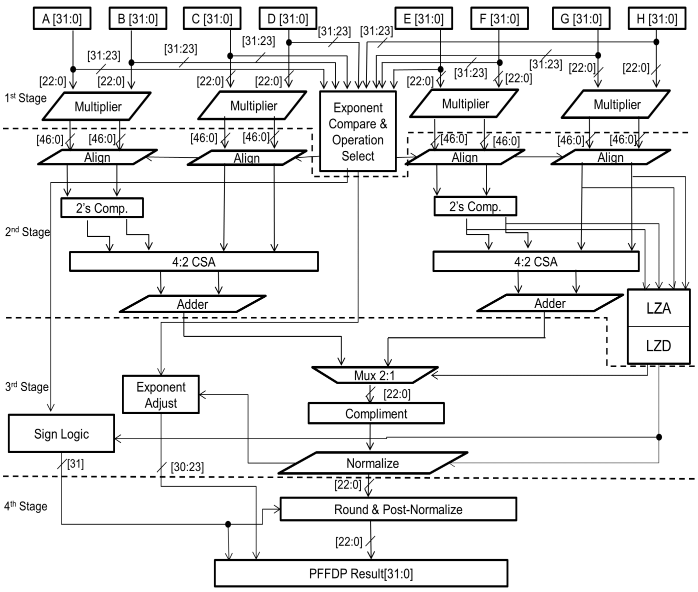

Figure 1: PFFDP Operational flowchart

carry adder with PFCF-CLA adder & traditional array multiplier with 2^r multiplier, rest of the stages are retained. These changes enhance the overall efficiency of the pipeline fused FDP (PFFDP) operations.

Double based number system(DBNS) needs $O(\frac{k}{logk})$ addition operations to perform k-bit multiplication operation [14]. Complexity of the multiplier depends on window size of the operation. General conversion tasks are of two types look-up table(LUT) and memory free algorithms approach. LUT conversion approach is considerably faster than that of memory less approach. Because, there is no need for extra hardware to implement the conversion logic. Considering hardware point of view, it is decided not to use large size LUTs. In general, larger the LUTs lesser the usage of addition/subtraction operations. However, computational experiments prove that as the size of LUTs increase beyond a certain point, the usage of addition/subtraction operations doesn't reduce significantly and also leads to exponential growth in area complexity.

The generalized representation of multiplicand will be

$$Y = \sum_{i=0}^{n} a_i \left(\sum_{j=1}^{b(i)} s_i^j 2^{c_i^j} \right) \tag{16}$$

where n is maximum value, a_i, $b(i)$ denote number of binary exponents that are multiplied by a_i, c_i^j represents j^{th} binary exponent of a_i, $s_i^j = \pm 1$ is sign of $2^{c_i^j}$. This generalized expression given in 16 is much better than DBNS. It is sufficient that $a_i = \{1, 3, 5, 7\}$ to represent any integer with 7-bit, where as the DBNS representation needs digit set of $a_i = \{1, 3, 9, 27, 81\}$ [15].The generalized non negative 7-bit integer can be represented in $x_1 \pm x_2$, where $x_1, x_2\ \epsilon \{1.2^n, 3.2^n, 5.2^n, 7.2^n\}$ and $n = \{0, 1, ..., 6, 7\}$.

Table 1: 7-bit Encoder Representation

1^{st} Term	2^0	2^1	2^2	2^3	2^4	2^5	2^6	2^7
1	0	0	0	0	0	0	0	1
3	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0
2^{nd} Term								
1	0	0	0	0	0	0	0	0
3	-1	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0

Any number can be represented by using matrix representation form that resembles DBNS representation. Unlike DBNS representation, we use first four rows to form one term and the second four rows to form the other term. Combination of these two terms gives the desired value as shown in Table 1. This table helps to represent any 7-bit number with the combination of first and second terms. However, we have to take care of the first term which should be always positive number and the second term may be positive or negative number that depends on the number representation. As an example, numeric value 125 is represented in Table 1. The value 125 is deduced by adding $128 + (-3) = 125$. This can be decoded as $1.2^7 = 128$ as first term and $3.(-1).2^0 = -3$ as second term. This is checked for ample number of random values and almost all are satisfying the matrix given in Table1. This table helps us to generate 11-bit number representation as follows: The multiplicand values {1,3,5,7} in both first and second terms are represented by using 2-bits each {00,01,10,11} respectively. They are denoted as $\{m_1, m_2\}$ as shown in Fig.2. The powers of 2 $\{2^0,2^1,2^2,2^3,2^4,2^5,2^6,2^7\}$ are decoded with 3-bits each {000,001,010,011,100,101,110,111} respectively denoted as $\{n_1, n_2\}$. Finally 1-bit for sign representation (second term only) denoted by $\{s_2\}$. Therefore by accumulating all the bits for first term and second term leads to 11-bit representation i.e.{00_111_01_0_000}. One more example to illustrate 7-bit encoder by taking numeric value 96 and its 11-bit representation will be {00_101_00_1_110}, where "00" represents '1', "101" represents $2^{(101)_2}$ and the first term will be $(1).(2^{(101)_2}) = 32$. In the same way second term will be "00" = '1', "110" = $2^{(110)_2}$ and '1' denotes positive number,deduces to $(1).(2^{(110)_2}) = 64$. The final result will be $32 + 64 = 96$.

The 7-bit encoder is used in 32-bit multiplier where 32-bit is split into four 7-bit data, starting from LSB and the remaining 4-bit is zero padded at MSB, this results in parallel 32x32 multiplier. One 32-bit multiplicand 'X' uses 5 encoders,each encoder converts respective 7-bit data into 11-bit data as shown in Fig.2. The multiplicand 'Y' is used to deduce 1Y,3Y,5Y,7Y by using shift and add operations i.e. $3Y = (Y << 1) + Y$, $5Y = (Y << 2) + Y$ & $7Y = (Y << 3) - Y$ respectively. Correct multipliers

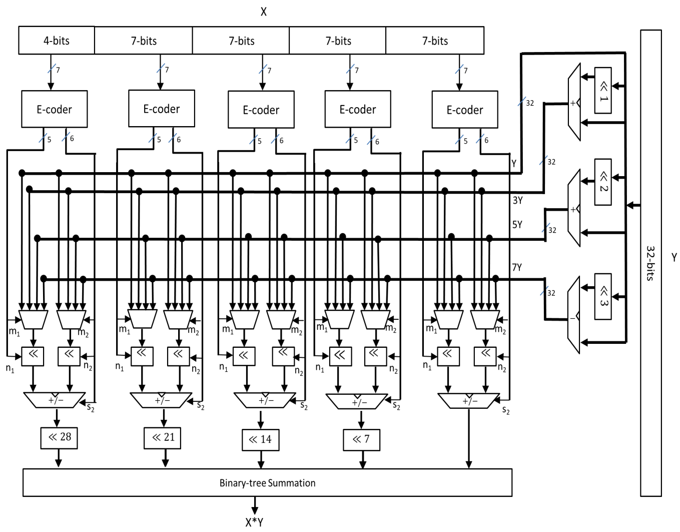

Figure 2: Radix 2^r multiplier architecture

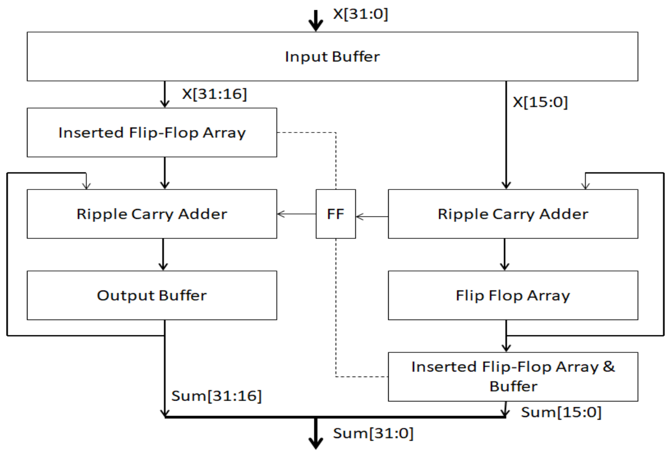

Figure 2a: Conventional Accumulator

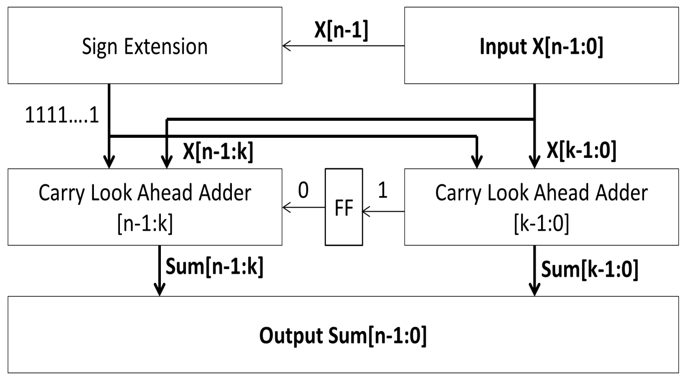

Figure 2b: 2-stage PFCF Accumulator

of Y is selected by multiplexers using m_1, m_2 and barrel shifters are triggered by n_1, n_2; s_2 decides to add or subtract intermediate results to obtain partial results. These partial products are shifted in multiples of 7 i.e.7k,where k= {0,1,2,3,4} from LSB to form final partial products. Then they are fed to the binary-tree summation which gives the desired final product(X*Y).

Figure 2 uses only 12 adders/subtractors to get multiplier operation.This architecture not only uses less number of operations but also very quick to produce final product i.e (X*Y). By replacing traditional multiplier with radix 2^r multiplier optimizes area and increases the performance of PFFDP unit as shown in Fig.1.

3.1. PFFDP using Radix-2^r Floating-point Multiplication Algorithm

1. Both multiplicand and multiplier should be in IEEE-754 format, either in single precision represents(1_ 8_ 23) or double precision represents (1_11_ 52). Where the representation defines (sign bit_exponent bits_mantissa bits).

2. Biasing Operation: Subtract Bias value from exponent value, i.e bias value $2^7 - 1 = 127$ in single precision and $2^{10} - 1 = 1023$ in double precision operations.

3. Concatenate operation: Concatenating single and double precision operations with '1' results in 24-bit and 53-bit representations respectively. They are represented as {1 & 23-bit mantissa} and {1 & 52-bit mantissa}.

4. Performing radix-2^r multiplication operation: Fig.2 is used to perform this operation. We use four 7-bit windows and seven 7-bit windows to represent 24-bit and 53-bit concatenated mantissa representations respectively i.e.{ 3'd0 &4}_7_7_7}; {{3'd0 &4} _7_7_7_7_7_7_7}.

5. Exponent compare operations: Alignment for the resultant partial products will be done. This refers to the amount of zero padding and shift operations.

6. Perform compression technique by using carry save adder for internal addition.

7. Normalization and rounding use LZA and leading zero detect(LZD): It also detects catastrophic cancellation by using OR logic.

8. Sign Operation: Perform XOR operation on sign bits.

9. The final output is concatenation of 1-bit from sign logic, 8-bits from exponent adjuster and final 23-bits are from rounding & post normalize unit results in desired floating-point fused dot product operation.

3.2. Pipeline Feedfarword Cutset Free(PFCF) Accumulator

Pipelined conventional accumulator of N-bit given in Fig.2a needs (N+1)*(M-1) additional flip flops for M stage operations. This additional flip-flops count significantly increases as number of pipeline stages increases. In order to reduce usage of additional flip flops we use feedforward cutset free(FCF)[16] concept that uses only one flip flop between two stages as shown in Fig.2b. For m-pipeline stages, FCF needs only m-1 additional flip flops. Comparison between pipelined conventional adder & PFCF adder is given in Fig.2c. The conventional sum output takes two clock cycles after the inputs are stored in the X. The PFCF produces output with only one clock cycle delay. The generated carry bit of lower 16-bit data is placed between

2-stage		**Pipelined Conventional Adder**				**PFCF Adder**		
		[31:16]		[15:0]		[31:16]		[15:0]
Cycle 1	X	1511		B9AD	X	1511		B9AD
	Sum	0000		0000	Sum	0000		0000
Cycle 2	X	0502		B9B9	X	0502		B9B9
	Sum	0000		0000	Sum	1511		B9AD
Cycle 3	X	1606		B9BA	X	1606		B9BA
	Sum	1511		B9AD	Sum	1A13	**1**	7366
Cycle 4	X	0000	**1**	0000	X	0000		0000
	Sum	1A14		7366	Sum	3019	**1**	2D20
Cycle 5	X	0000	**1**	0000	X	0000		0000
	Sum	**301B**		**2D20**	**Sum**	**301B**		**2D20**

Figure 2c: 2-stage Pipelined conventional and PFCF operations

lower & higher 16-bit representation. The carry generated by lower 16-bits is involved in the higher 16-bit operation only after one clock cycle as in cycles 3 & 4. This operation takes only one flip-flop to store & process carry bit. However conventional operation takes N+1 flip-flops to hold and process the data & carry bit as shown in cycles 4,5. This requires huge flip-flop array to process the entire operation in one clock cycle. However, both accumulators produce same final output in cycle 5 only.

Ripple carry adders have highest critical path delay that leads to slower operations. An alternative to reduce critical path delay is by replacing with carry look ahead adder (CLA). But, The carry prediction logic adder significantly increase area and power consumption. However, usage of PFCF logic accumulator removes this drawback in CLA adders. Replacing adder in Fig.1 & binary tree summation in Fig.2 with PFCF-CLA adder significantly reduces critical path delay, area and power consumption. This enhancement in binary-tree summation block clearly increases the performance of radix-2^r multiplier architecture. This in turn increases the overall performance of PFFDP operations.

4. Performance Evaluation and Results

The system is simulated using verilog HDL simulator and implemented using a 60nm technology library. The operating conditions of the processor are 1.2V at 40°C (min) to 0.95V at 125°C (max). This processor is excited by 1GHz global clock with low clock slew ($<$ 10% of the global clock), considering good positive slack and skew. Setting maximum clock uncertainty between 1-2%. Maximum Input and output delays are 10ns.

Area comparison of all the sub-blocks in FDP units are shown in Table 2. The proposed design consumes 15% less area as compare to that of Sohn's FDP unit[18]. The major variation in the area is observed in multiplier that is designed with the help of radix-2^r multiplier and addition is designed with the help of PFCF accumulator based CLA.

Table 2: Area Break-down FDP units(Single Precision)

Design	Kim's [7]	Tao's [17]	Sohn's [18]	Proposed
Multiplier	16,400	16,400	16,400	14,592
Exponent Compare	4,600	4,300	3,400	3,382
Alignment	5,200	11,200	3,600	3,512
Reduction	1,200	1,200	2,400	1,214
Addition	9,100	9,100	1,200	730
LZA & Normalization	1,300	1,300	2,200	1,254
Rounding	1,900	1,900	1,600	1,600
Rest of Control Logic	1,300	1,800	1,600	1,200
Total (μm^2)	41,000	47,200	32,400	27,484

Table 3: Latency Break-down of FDP units(Single Precision)

Design	Kim's [7]	Tao's [17]	Sohn's [18]	Proposed
Multiplier	0.64	0.64	0.64	0.51
Alignment	0.96	0.82	0.40	0.42
Reduction	0.32	0.32	0.16	0.16
Addition	0.52	0.52	0.48	0.24
LZA & Normalization	0.32	0.32	0.16	0.16
Rounding	0.34	0.34	0.42	0.42
Toal (ns)	3.10	2.96	2.26	1.91

Latency of all the sub-blocks in FDP units are tabulated in Table 3.From the table, it is observed that radix-2^r multiplier is 20% more faster than Sohn's multiplier[18] and PFCF accumulator based CLA adder is almost 50% faster.

Individual Comparison of area, delay & power calculations will not give exact performance of the system. Architectures occupying high area or power with low execution time & low area or power with high execution are technically impossible to compare directly. To get equilibrium in comparison process, area-delay-product(ADP) and power-delay-product(PDP) parameters are used to measure the exact performance of the overall architecture. Table 4 displays single precision and double precision operational parameters of different FDP unit architectures. The proposed PFFDP architecture is compared with the referred journal's data to validate performance w.r.to area, latency, total delay, ADP & PDP parameters. Area and power parameters are normalized to 60nm technology for fair comparison.

The proposed single precision PFFDP unit consumes 45% & 15% less area when compare to that of Lang's discrete FDP unit [19] & Sohn's FDP unit [18] respectively. Although the proposed PFFDP unit consumes 6% more area when compare to that of Xing's FDP unit [12] in single precision implementation, it dominates Xing's by occupies 11% less area in double precision implementation.

Table 4: Comparison of IEEE-754 Single & Double Precision Floating-point FDP units

Single Precision							
	Lang's [19]	Sohn's [2]	Kim's [7]	Tao's[17]	Sohn's [18]	Xing' s [12]	Proposed
Norm.Area (μm^2)	50,050	41,400	41,000	47,200	32,400	25,870	27,484
Latency (ns)	3.90	3.14	3.10	2.9	2.26	2.05	1.91
Total Delay (ns)	4.49	4.71	5.43	4.96	2.73	2.46	2.14
Norm.Power (mW)	54.78	44.30	43.30	49.67	32.74	2.66	2.24
ADP($\mu m^2.ns$)	224.47	194.99	222.43	234.06	88.60	63.64	58.79
PDP ($mW.ns$)	245.69	208.65	234.90	246.31	89.53	6.54	4.79
Double Precision							
	Lang's [19]	Sohn's [2]	Kim's [7]	Tao's[17]	Sohn's [18]	Xing' s [12]	Proposed
Norm. Area(μm^2)	119,100	101,000	99,350	114,700	76,350	52,670	46,720
Latency (ns)	4.90	4.26	4.18	4.02	2.96	2.21	2.07
Total Delay (ns)	7.60	7.67	7.32	6.87	3.58	2.65	2.32
Norm. Power (mW)	122.17	99.7	98.07	112.58	74.42	3.94	3.67
ADP ($\mu m^2.ns$)	904.56	774.47	726.75	788.47	273.46	139.68	108.32
PDP ($mW.ns$)	927.88	764.50	717.38	773.90	266.54	10.44	8.52

The proposed PFFDP unit's optimized area consumption has an upper hand on Lang's [19] by 64%, Kim's [7] by 43%, Toa's [17] by 59% & Sohn's [18] by 39%.

Xing's FDP unit [12] consumes 12% & 7% higher power consumption than the proposed PFFDP unit in single and double precision implementations respectively. It also beats rest of the FDP units with significant margin.

ADP values of the proposed unit are very impressive and they differ by 18% & 22% with Xing's FDP unit [12] in single & double precision implementations. It also differs by 34% & 60% with Sohn's FDP unit [18] in single & double precision implementations. The other important parameter PDP values are also compared with Xing's FDP unit [12] that differs by 27% & 18% in single & double precision implementations. Lower ADP & PDP values the more balanced system performance. Hence, the proposed PFFDP unit is well balanced architecture with respect to area, power & delay aspects.

5. Conclusion and Future Scope

This paper presents implementation of PFFDP unit that plays vital role in DSP application. Multiplier tree in traditional FDP operation is replaced with radix-2^r multiplier that increases performance by reducing operation delay and die area. In binary tree simulator, three stage adder operations and traditional adders are replaced with PFCF-CLA adder that reduce additional memory elements by maintaining almost same latency. This architecture is powered with 1GHz clock that delivers $58.79\mu m^2.ns$, 108.32 $\mu m^2.ns$ ADP & $4.79mW.ns$ & $8.52mW.ns$ PDP values in single and double precision implementations. These ADP and PDP values prove that the implemented architecture is well balanced. This PFFDP architecture can also used in machine language algorithms for optimized operations.

Acknowledgements

We thank our Parents, Guru & the Almighty who trusted and supported us all the time.

References

[1] E. E. Swartzlander, H. H. M. Saleh, Fft implementation with fused floating-point operations, IEEE Transactions on Computers 61 (2) (2012) 284–288.

[2] J. Sohn, E. E. Swartzlander, Improved architectures for a floating-point fused dot product unit, in: 2013 IEEE 21st Symposium on Computer Arithmetic, 2013, pp. 41–48.

[3] S. Yun, G. E. Sobelman, X. Zhou, A low complexity floating-point complex multiplier with a three-term dot-product unit, in: 2014 IEEE International Conference on Signal Processing, Communications and Computing (ICSPCC), 2014, pp. 549–552.

[4] Ieee standard for floating-point arithmetic, IEEE Std 754-2019 (Revision of IEEE 754-2008) (2019) 1–84`doi:10.1109/IEEESTD.2019.8766229`.

[5] H. H. Saleh, E. E. Swartzlander, A floating-point fused dot-product unit, in: 2008 IEEE International Conference on Computer Design, 2008, pp. 427–431.

[6] J. Sohn, E. E. Swartzlander, Improved architectures for a fused floating-point add-subtract unit, IEEE Transactions on Circuits and Systems I: Regular Papers 59 (10) (2012) 2285–2291.

[7] D. Kim, L. Kim, A floating-point unit for 4d vector inner product with reduced latency, IEEE Transactions on Computers 58 (7) (2009) 890–901.

[8] H. Sam, A. Gupta, A generalized multibit recoding of two's complement binary numbers and its proof with application in multiplier implementations, IEEE Transactions on Computers 39 (8) (1990) 1006–1015. `doi:10.1109/12.57039`.

[9] P. . Seidel, L. D. McFearin, D. W. Matula, Secondary radix recodings for higher radix multipliers, IEEE Transactions on Computers 54 (2) (2005) 111–123. `doi:10.1109/TC.2005.32`.

[10] A. K. Oudjida, N. Chaillet, M. L. Berrandjia, Radix-2r arithmetic for multiplication by a constant: Further results and improvements, IEEE Transactions on Circuits and Systems II: Express Briefs 62 (4) (2015) 372–376. `doi:10.1109/TCSII.2014.2387620`.

[11] A. K. Oudjida, N. Chaillet, Radix- 2^r arithmetic for multiplication by a constant, IEEE Transactions on Circuits and Systems II: Express Briefs 61 (5) (2014) 349–353. `doi:10.1109/TCSII.2014.2312799`.

[12] X. Wei, H. Yang, W. Li, Z. Huang, T. Yin, L. Yu, A reconfigurable 4-gs/s power-efficient floating-point fft processor design and implementation based on single-sided binary-tree decomposition, Integration 66 (2019) 164 – 172. doi:https://doi.org/10.1016/j.vlsi.2019.02.008.

[13] A. A. Wahba, H. A. H. Fahmy, Area efficient and fast combined binary/decimal floating point fused multiply add unit, IEEE Transactions on Computers 66 (2) (2017) 226–239.

[14] A. K. Oudjida, N. Chaillet, M. L. Berrandjia, Radix-2r arithmetic for multiplication by a constant: Further results and improvements, IEEE Transactions on Circuits and Systems II: Express Briefs 62 (4) (2015) 372–376.

[15] V. S. Dimitrov, K. U. Jarvinen, J. Adikari, Area-efficient multipliers based on multiple-radix representations, IEEE Transactions on Computers 60 (2) (2011) 189–201.

[16] S. Ryu, N. Park, J. Kim, Feedforward-cutset-free pipelined multiplyaccumulate unit for the machine learning accelerator, IEEE Transactions on Very Large Scale Integration (VLSI) Systems 27 (1) (2019) 138–146.

[17] Y. Tao, G. Deyuan, F. Xiaoya, J. Nurmi, Correctly rounded architectures for floating-point multi-operand addition and dot-product computation, in: 2013 IEEE 24th International Conference on Application-Specific Systems, Architectures and Processors, 2013, pp. 346–355.

[18] J. Sohn, E. E. Swartzlander, A fused floating-point four-term dot product unit, IEEE Transactions on Circuits and Systems I: Regular Papers 63 (3) (2016) 370–378.

[19] T. Lang, J. D. Bruguera, Floating-point fused multiply-add with reduced latency, in: Proceedings. IEEE International Conference on Computer Design: VLSI in Computers and Processors, 2002, pp. 145–150.