Middle-East Journal of Scientific Research 23 (6): 1187-1196, 2015
ISSN 1990-9233
© IDOSI Publications, 2015
DOI: 10.5829/idosi.mejsr.2015.23.06.22262

# Low Power Reconfigurable Double Precision Multiplier for DSP Applications

[1]S. Karthick, [2]S. Valarmathy, [3]E. Prabhu and [4]S. Karthick

[1]Department of Electronics and Communication Engineering,
Bannari Amman Institute of Technology, Sathyamangalam, Tamil Nadu, India
[2]Department of Electronics and Communication Engineering,
Bannari Amman Institute of Technology, Sathyamangalam, Tamil Nadu, India
[3]Department of Electronics and Communication Engineering,
[4]VLSI Design and Testing Group, Amrita Vishwa Vidyapeetham University,
Coimbatore, Tamil Nadu, India
[5]Department of Electronics and Communication Engineering,
Bannari Amman Institute of Technology, Sathyamangalam, Tamil Nadu, India

**Abstract:** Floating point arithmetic's are widely used in commercial applications like financial analysis, banking, tax calculation, currency conversion, insurance & accounting and also in large set of scientific, numeric and signal processing computations. In digital signal processors (DSP), IEEE 754 compliant floating point arithmetic is used for large dynamic range or for rapid prototyping. In floating point arithmetic, multiplication is the most commonly used arithmetic. In this brief, low power floating point multiplier is presented (LPFM). LPFM utilizes the 2 partition Karatsuba algorithm and supports run-time reconfigurability with dual single precision or single double precision. LPFM facilitates the design space exploration through the variable pipelining method and low power adder architectures which helps in achieving different performance than the conventional/regular adder architectures. LPFM design was modeled in Verilog HDL and synthesized using Cadence Encounter RTL Compiler by mapping to TSMC 65nm technological library node. Proposed run time reconfigurable mantissa multiplier of the extended double precision floating point multiplier requires 12.18% less area and improves 14.20% performance with 5.14% less power. Proposed ALU improves 16.31% performance and consumes 4.78% less power than the regular ALU design.

**Key words:** DSP · Low Power floating point multiplication · Adder · Low Power VLSI

## INTRODUCTION

Most important operations involved in DSP are convolution, filtering and inner product computation. To implement such operations Adders, Multipliers and Multiply-Accumulate components are essential. DSP operations basically use non-linear functions and they consist of repetitive application of multiplications and additions. Thus these components are responsible for the design constraints like area, speed and power consumption.

Advances in VLSI technology and the use of floating-point operations for business, technical and recreational applications that use floating-point computational logic has been an essential component of high-performance computer systems and mobile applications. Architecture of floating point computational units has greater affect on the area, power consumption and its performance [1]. However the performance of digital circuits has been increased rapidly due to the technological improvements in circuit design and architecture, but the power consumption is the issue [2]. Hence in this brief, low power architectures are explored for floating point multiplication.

Several efficient algorithms and architectures were designed for implementation of floating point units and many were dedicated to particular design constraint like area and performance. In [1], author has demonstrated

**Corresponding Auhor:** S. Karthick, Department of Electronics and Communication Engineering, Bannari Amman Institute of Technology, Sathyamangalam, Tamil Nadu, India.

that how the design constraints can be trade-off as per the design goals or desired applications; floating point unit for data intensive applications and Morphable networked micro-architecture projects are implemented with different design goals. Latency and operational chip area has been improved in [3] while implementing the double precision floating point adder architecture. High speed floating point multiplication is implemented using Dadda algorithm for mantissa multiplication in [4]. Karatsuba partition algorithm has been used in mantissa multiplication to reduce the area by reducing the numbers of smaller bit width multipliers [5].

Floating point multiplications implemented in the past have utilized different concepts at the algorithmic level and reduced the design constraints and some have implemented as per the design constraints/desired applications. But they are limited to performance and area improvement, while the power reduction is dependent on the area reduction. And still due to technological growth the Leakage power has become the major constraint. Hence there is a need for architecture which is specific to the power reduction [6].

Limitations of the existing implementations:

- Limited to area and performance constraints
- Optimizations are at the algorithmic level

Features of this brief:

- Low power architecture
- Variable pipelining for multiplication
- Run-time reconfigurability to Dual single precision or single double precision

This paper explores the low power architectures for the floating point multiplication. Other features explored in this brief are variable pipelining, low power adder architectures to achieve the design space explorations and their impact at different algorithms and at sub-system level. Remaining sections of the paper are organized as follows. Background regarding the floating point multiplication is provided in section 2. Section 3 provides the architectural characterizations. Results are discussed and evaluated in section 4. Paper is concluded in section 5 and references are provided in the last section [7].

**Background:** Single and Double precision Floating point bit representations is shown in Figure 1.

Floating point arithmetic involves separate processing for sign, exponent and mantissa parts between the operands for the intended function. After processing, again they are normalized and framed as per the desired format. To multiply two operands of floating point formats, sign bits of the operands are XOR'ed to calculate the sign, exponents are added and their precision's bias is subtracted from the addition result and finally the significand or the mantissa parts are multiplied.

| Precision | Sign-bit | Exponent | Mantissa |
|---|---|---|---|
| Single | 1 bit | 8 bits | 23 bits |
| Double | 1 bit | 11 bits | 52 bits |

Fig. 1: Single & Double precision Floating point bit representation (IEEE 754 standard)

Floating point multiplication requires a large multiplier for mantissa multiplication and it causes area and power overhead in its implementation. Due to this area overhead truncated multipliers were designed by reducing the block multipliers, but at the cost of precision. To overcome such a trade-off 2 partition karatsuba algorithm was used for mantissa multiplication, where full precision is maintained and area reduction is also achieved [5].

**Architecture:** This section describes about the multiplier architecture which reduces the block multipliers required in the partitioned multiplication concept. In this brief, mantissa multiplication is carried out for 66bit operands, to provide dual single extended precision or double precision multiplication. It also provides the flexibility of choosing pipelining in several stages of multiplication. Low power adder architectures are also provided in the multiplication stages[8].

The proposed floating point multiplier provides run-time reconfigurability between the dual single-extended precision and single double precision. Design space explorations with different adder architectures are done.

**Karatsuba Partition Algorithm:** In this algorithm, less number of block multipliers is used and no precision loss is occurred. It is one of the fast multiplication algorithms which depends on the divide and conquer paradigm. It reduces the multiplication of two numbers (n-bit X n-bit) from $\mathbf{n^2}$ to $\mathbf{3n^{\log_2 3}}$ [6]. The basic steps involved in Karatsuba algorithm allows to compute the product of two large numbers say **'x'** and **'y'**, using only three multipliers of smaller numbers. Let **'x'** and **'y'** be of n-bits each and are divided by some base **'B'** for any positive integer **'m'** less than **'n'**.

$$x = x_1 B^m + x_0 \quad (1)$$

$$y = y_1 B^m + y_0 \quad (2)$$

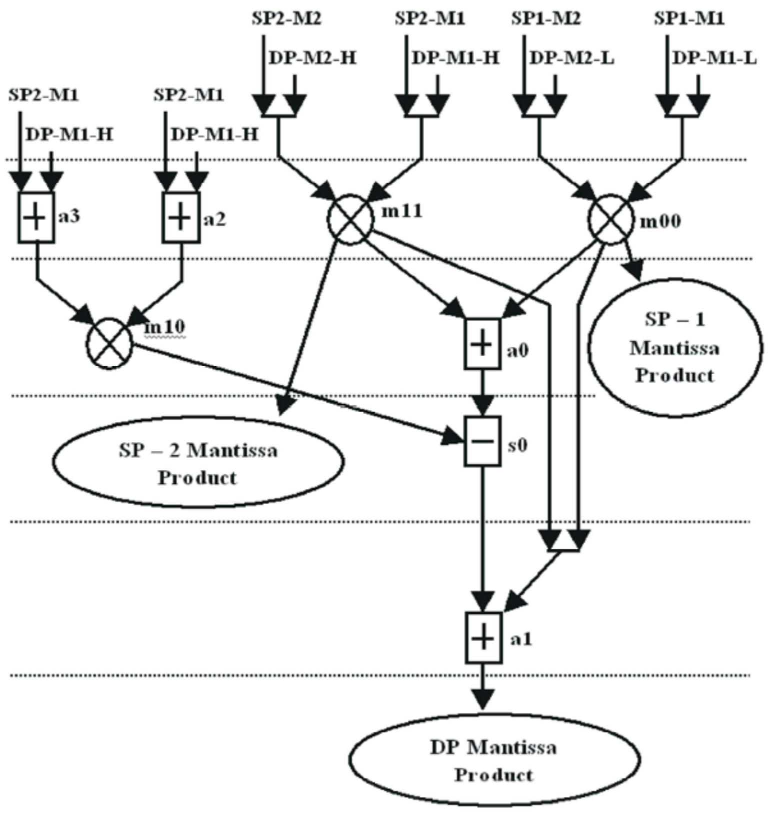

Fig. 2: Dual Single Precision/ Double precision multiplication
DP-M1-L: Double precision Mantissa -1 (LSB half); DP-M1-H : Double precision Mantissa -1 (MSB half);
DP-M1-L: Double precision Mantissa -2 (LSB half); DP-M1-H : Double precision Mantissa -2 (MSB half);
SPX-MY: Single precision, Y is Mantissa -1/2; X is LSB/MSB half

where $x_0$ and $y_0$ are = m digit.

Now $xy = (x_1 B^m + x_0)(y_1 B^m + y_0) = z_2 B^2 m + z_1 B^m + z_0$ (3)

where $z_2 = x_1y_1$, $z_0 = x_0y_0$, $z_1 = x_1y_0 + x_0y_1$

Equation 3 requires four multiplications, but karatsuba observed that **‘$z_1$’** can be written in another way so that only one multiplier is required instead of two [9].

$$z_1 = (x_1 + x_0)(y_1 + y_0) - z_2 - z_0 \quad (4)$$

Thus the number of multiplication is reduced from 4 to 3 at the overhead of some addition and subtraction; which is lesser than the multiplier. It also provides complete and correct multiplication without loss of any precisions. Thus it contains 25% reduction in the block multipliers and tends to increase as the partitioning increases. For example extending the two partition method to three partitions, where it reduces 3 multipliers from 9 to 6 which is 33% reduction [9].

**Mantissa Multiplication:** Karatsuba 2 partition method is adopted in the proposed multiplication methodology. In the proposed LPFM, multiplication can be carried out for extended precision floating point numbers. Run-time reconfigurability is facilitated for dual single extended precision or double precision. Architecture for the dual single precision and double precision multiplication is shown in Figure 2.

Architecture in Figure 2, can be used to implement two 34 bit or one 66 bit multiplications. Hence it can be easily used for the single/double or its extended version multiplication. Architecture in Figure 2 uses the 2-partition karatsuba algorithm, where a 66 bit multiplier is implemented using two 33 bit multipliers (m00 and m11), one 34 bit multiplier (m10) and some adders and subtractions. Again the 34 and 33 bit multipliers are built using 2- partition karatsuba algorithm, where it requires two 17-bit multipliers and one 18-bit multiplier along with some additions and subtractions; for 34-bit multiplication as shown in Figure 3. Thus it requires a total of 9 multipliers [10].

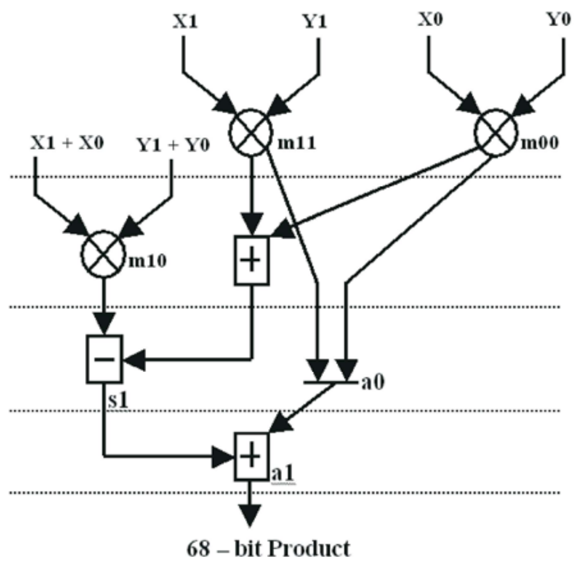

Fig. 3: Architecture of 34-bit multiplier

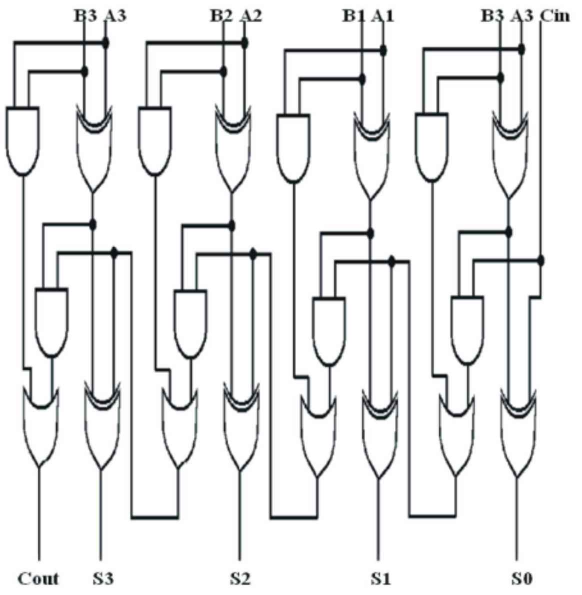

Fig. 4: Regular 4-bit Ripple carry adder [7]

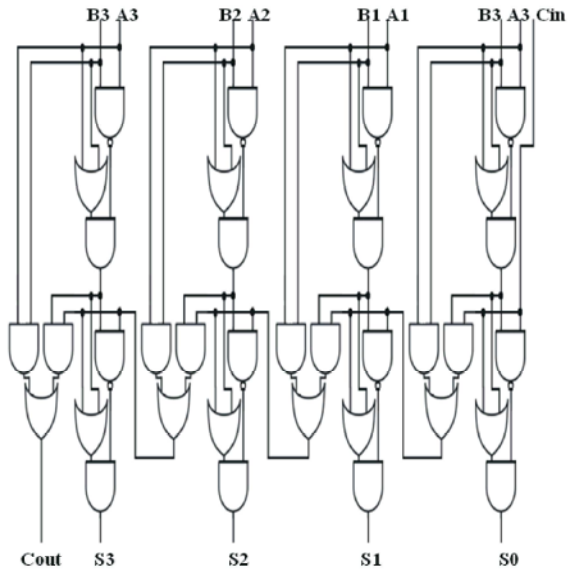

Fig. 5: Proposed 4-bit Ripple Carry Adder

For reconfigurabiity, input operands are multiplexed for the 33 bit multipliers (m00 and m11). Both the 33 bit multipliers are multiplexed between different sets of single and double precision mantissa operands. M00 multiplier is multiplexed with first set of single precision Operand and first half of the double precision operand. Similarly m11 multiplier multiplexes the second set of single precision operand and second half of the double precision operand. Output of the single precision multiplication are taken from the multipliers m00 and m11, whereas the double precision outputs are processed through the multiplier m10 and some adders and subtractions as shown in Figure 2, to get the final mantissa multiplication output. Thus it facilitates the processing of double precision or dual single precision multiplication by efficiently sharing the resources and without any pipeline stall.

**Design and Methodology:** Architectures were modeled using Verilog HDL for the implementation. A multiplier with 66 bits wide was designed using 2-partition karatsuba algorithm to feature as dual single- extended precision or one double precision mantissa multiplication. Architectures in Figure 2 and Figure 3, contains intensive computations and this is major part in the floating point multiplication for the design constraints. Hence low power architectures are modeled to this multiplication part.

Low power adder architecture shown in Figure 4 (RCA – Ripple Carry Adder), is utilized for addition in the multiplication. Conventional adder architecture is shown in Figure 5. Proposed adder architectures avoids the possible inverters in the critical path and also utilizes the complex cells like AND-AND-OR and AND-OR to the larger extent. Complex standard cells have higher stack and lesser area; hence they leak less power than the regularly used standard cells. Use of complex cells also reduces interconnects between the smaller gates and reduces the interconnect delay & power. In Figure 4 & Figure 5, architectures are shown only for 4-bit wide, but the actual design contains as per the requirement.

Proposed adder architecture is utilized in the design wherever it is possible, example in Figure 3; input additions for the 18-bit multiplier, 34-bit addition for the outputs of 17-bit multipliers to deduce addition of z0 and

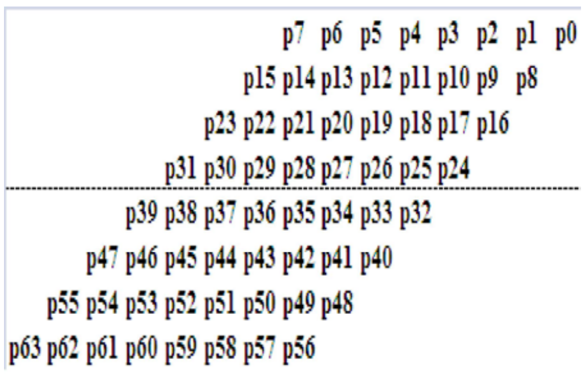
p7 p6 p5 p4 p3 p2 p1 p0
p15 p14 p13 p12 p11 p10 p9 p8
p23 p22 p21 p20 p19 p18 p17 p16
p31 p30 p29 p28 p27 p26 p25 p24
p39 p38 p37 p36 p35 p34 p33 p32
p47 p46 p45 p44 p43 p42 p41 p40
p55 p54 p53 p52 p51 p50 p49 p48
p63 p62 p61 p60 p59 p58 p57 p56

Fig. 6: Partial product partitioned to provide pipelining

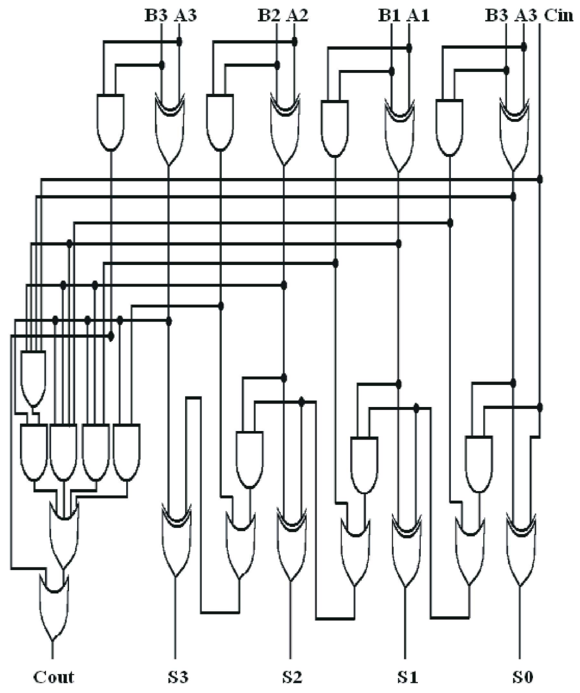

Fig. 7: Regular 4-bit Carry Look Ahead Adder

z2 of karatsuba algorithm and also utilized in subtraction part also. Similarly proposed adder architectures are also utilized in the architecture of Figure 2. ‘

As mentioned above regarding the advantage of proposed adder architecture, the multiplier design provides flexibility in the design constraints (like area, delay and power) and enables new corners for analysis. Several corners resulting from such optimizations are discussed in the results section of the paper.

In this brief, like low power adder architectures discussed in the previous paragraphs, improves the performance of the 17-bit and 18-bit mutlipliers by providing the variable pipelining (featuring the choice of pipelining at reduction and propagate addition stage of the multiplier) and performance improvement in the

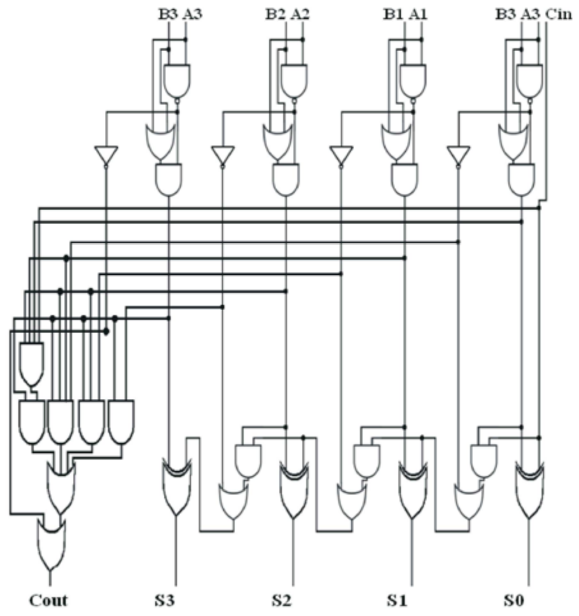

Fig. 8: Proposed 4-bit Carry Look-ahead Adder architecture

propagate addition with carry predictive method. As per the known fact that the pipelining improves the performance, the proposed multiplier provides pipelining in the partial product reduction stage, which computes the blocks parallely as shown in Figure 6. Partial products of the 8-bit multiplier are partitioned into two blocks to process them parallely, similar partitions are done for 18-bit and 17-bit multipliers. This provides the two-pipelined registers and another one is provided before the propagate addition stage of the multipliers. Pipelining stage can be selected as per the requirement; which enables the design to achieve variable performance.

Performance of the propagate addition is improved by the carry prediction method, where carry look ahead adder (CLA) is utilized due to its predictive behavior. Here also low power concept is adopted for the carry look-ahead adder, so that it can improve the performance as well as decrease the power consumption; which again leads to the new corner for analysis. Thus the design can have the extended range of performance and power results for design space exploration. Regular and proposed Carry look ahead architectures are shown in Figure 7 and Figure 8 respectively.

Thus in this brief, with optimizations mentioned below will yield better results for the mantissa multiplication in the floating point multiplier.

- Low power adder architecture for karatsuba algorithm
- Variable pipelining in partitioned multipliers

- Low power carry prediction logic in propagate addition stage of the partitioned multipliers to reduce the delay
- Low power – same/high performance adders

Since the optimizations are done at the datapath architectural level, the proposed architectures can be extended to any bit-width levels and can achieve the low power results. It's also possible to utilize the proposed architectures at any abstract hierarchical levels in the design cycles.

## RESULTS

The mantissa part of floating point multiplication based on 2-partition karatsuba algorithm is designed and modeled using Verilog HDL. Functionality of the design was verified in waveform editor by simulating with the Modelsim simulator. Standard ASIC design methodology was considered for design synthesis and results were benchmarked. Synopsys Design Compiler was used for synthesis and was mapped to 65nm technology node. Results of the building blocks of multiplier architectures are tabulated in the Tables 1 & 2. Table 1 shows the results of the regular and proposed ripple carry adder architectures having widths of 33-bits and 66-bits; used in the 2-partition karatsuba algorithm based mantissa multiplier architecture of floating point multiplication.

From Table 1, it can be observed that the proposed architectures results are better than the regular adder architectures in all the design quality parameters (area, timing and power). And it also proves that the proposed architecture holds good and its impact will be similar for any bit-widths. Proposed architectures have obtained 14% improvement in performance with 12% power efficiency. This enables the designed architecture to be dealt as the low power – high performance architecture and it explores into new technological corner.

Proposed adder architecture shown in Figure 5, was applied to the partitioned mutlipliers (m00, m11 and m10) of Figure 3 and its impact are observed at the multiplier level. This design was again compared by applying the regular adder architecture of Figure 4 in the partitioned multipliers of Figure 3. Apart the ripple carry adder architectures, the carry look ahead logics are incorporated in the propagate addition stage of the partitioned multipliers to reduce the delay. Similar to proposed ripple carry architecture, low power carry look ahead adder architecture is also proposed, which is shown in Figure 8. The results of the regular and proposed partitioned multiplier architectures of 16, 17 & 18-bits are tabulated in Table 2.

Results of Table 2 (a), Table 2 (b) and Table 2 (c), suggests that the proposed partitioned multiplier architectures with proposed adder architectures provides better results than the regular architectures.

Table 1: Existing and Proposed Ripple carry adder architecture

| | 33-bits | | | 66-bits | | |
|---|---|---|---|---|---|---|
| Design bit width | Existing | Proposed | % Change | Existing | Proposed | % Change |
| Area (Sq. microns) | 443.52 | 362.16 | 18.34 | 894.96 | 730.44 | 18.38 |
| Delay (ns) | 3.76 | 3.2 | 14.89 | 7.56 | 6.44 | 14.81 |
| Dp (micro watt) | 86.926 | 76.905 | 11.53 | 176.529 | 156.149 | 11.54 |
| Lp (micro watt) | 4.619 | 3.527 | 23.64 | 9.321 | 7.112 | 23.70 |
| Tp (micro watt) | 91.545 | 80.432 | 12.14 | 185.85 | 163.261 | 12.15 |

Note: "Dp" is Dynamic power; "Lp" is Leakage power and "Tp" is Total power

Table 2(a): Existing and Proposed 16-bits partitioned multiplier architecture

| | RCA | | | CLA | | |
|---|---|---|---|---|---|---|
| Adder variant | [5] | Proposed | % Change | [5] | Proposed | % Change |
| Area (Sq. microns) | 5334.12 | 4715.28 | 11.60 | 5379.84 | 4798.8 | 10.80 |
| Delay (ns) | 3.37 | 2.93 | 13.06 | 2.59 | 2.29 | 11.58 |
| Dp (micro watt) | 204.906 | 201.198 | 1.81 | 206.112 | 201.932 | 2.03 |
| Lp (micro watt) | 51.231 | 42.74 | 16.57 | 51.602 | 43.727 | 15.26 |
| Tp (micro watt) | 256.137 | 243.938 | 4.76 | 257.714 | 245.659 | 4.68 |

Note: "Dp" is Dynamic power; "Lp" is Leakage power and "Tp" is Total power

Table 2(b): Existing and Proposed 17-bits partitioned multiplier architecture

| | RCA | | | CLA | | |
|---|---|---|---|---|---|---|
| Adder variant | [5] | Proposed | % Change | [5] | Proposed | % Change |
| Area (Sq. microns) | 5938.2 | 5231.52 | 11.90 | 6013.8 | 5368.32 | 10.73 |
| Delay (ns) | 3.46 | 3.02 | 12.72 | 2.77 | 2.56 | 7.58 |
| Dp (micro watt) | 224.733 | 220.261 | 1.99 | 226.743 | 221.583 | 2.28 |
| Lp (micro watt) | 57.208 | 47.546 | 16.89 | 57.817 | 49.164 | 14.97 |
| Tp (micro watt) | 281.941 | 267.807 | 5.01 | 284.56 | 270.747 | 4.85 |

Note: “Dp” is Dynamic power; “Lp” is Leakage power and “Tp” is Total power

Table 2(c): Existing and Proposed 18-bits partitioned multiplier architecture

| | RCA | | | CLA | | |
|---|---|---|---|---|---|---|
| Adder variant | [5] | Proposed | % Change | [5] | Proposed | % Change |
| Area (Sq. microns) | 6589.8 | 5787 | 12.18 | 6667.2 | 5931 | 11.04 |
| Delay (ns) | 3.52 | 3.02 | 14.20 | 3.01 | 2.75 | 8.64 |
| Dp (micro watt) | 244.856 | 239.989 | 1.99 | 247.156 | 241.552 | 2.27 |
| Lp (micro watt) | 63.663 | 52.677 | 17.26 | 64.296 | 54.357 | 15.46 |
| Tp (micro watt) | 308.519 | 292.666 | 5.14 | 311.452 | 295.909 | 4.99 |

Note: “Dp” is Dynamic power; “Lp” is Leakage power and “Tp” is Total power

Table 3: Existing and Proposed 66-bits 2-partitioned karatsuba algorithm based mantissa multiplier architecture

| | RCA | | | CLA | | |
|---|---|---|---|---|---|---|
| Adder variant | [5] | Proposed | % Change | [5] | Proposed | % Change |
| Area (Sq. microns) | 68087.16 | 59883.48 | 12.05 | 69951.6 | 63333.72 | 9.46 |
| Delay (ns) | 10.43 | 8.98 | 13.90 | 7.3 | 6.82 | 6.58 |
| Dp (milli watt) | 1.471 | 1.453 | 1.22 | 1.492 | 1.469 | 1.54 |
| Lp (milli watt) | 0.654 | 0.543 | 16.97 | 0.669 | 0.582 | 13.00 |
| Tp (milli watt) | 2.125 | 1.996 | 6.07 | 2.161 | 2.051 | 5.09 |

Note: “Dp” is Dynamic power; “Lp” is Leakage power and “Tp” is Total power

Results of Table 2 (a), Table 2 (b) and Table 2 (c), suggests that the proposed partitioned multiplier architectures with proposed adder architectures provides better results than the regular architectures. Tables 2 (a), (b) and (c), proves that the proposed architectures have less area – high performance and low power characteristics. This shows that the impact of adder architectures at the multiplier level and proves that the proposed architectures can be utilized at higher hierarchical levels.

From Tables 2 (a), (b) and (c), it can be observed that the existing CLA multiplier architecture (carry prediction logic) has reduced the delay of the multiplier architecture, but trades-off in area and power. This limits the use of CLA architectures at the area and power constraint design. Even though the proposed RCA based multiplier architecture has lesser area and low power but it can’t provide the performance of CLA. This resembles the low area – low power technological corner. The proposed CLA based multiplier architecture provides better results in all the aspects and when it is compared with the existing RCA based multiplier architecture; it results in the low power and high performance technological corner. This suggests that the proposed CLA based partitioned multiplier architecture provides high performance with low power consumption than the existing partitioned multiplier architecture. Thus from Tables 2 (a), (b) and (c), it can be summarized that the proposed architectures can explore new technological corners for the analysis.

Both the proposed adder (RCA & CLA) architectures are implemented as individual adder architectures in the mantissa multiplier of 66bit wide for extended double precision floating point multiplication. Results of this are tabulated in Table 3 and compared with the existing mantissa multiplier architecture having regular adder architectures.

Similar to the results of the Tables 2 (a), (b) & (c), Table 3 also provides the same results but at the higher bit-width level. This proves that the proposed adder architecture can be incorporated for any bit-width and at any abstract hierarchical in the design cycle (algorithm/sub-system/system). Here also the proposed

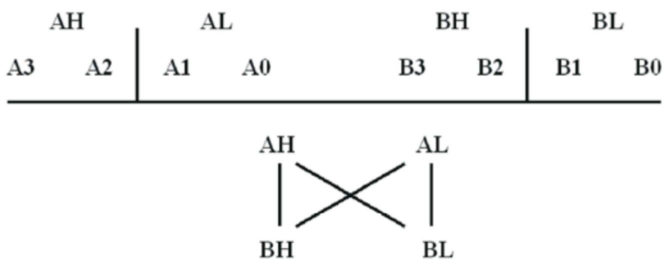

Fig. 9: Representation of Urdhava Tiryagbhyam algorithm for 4-bit multiplication

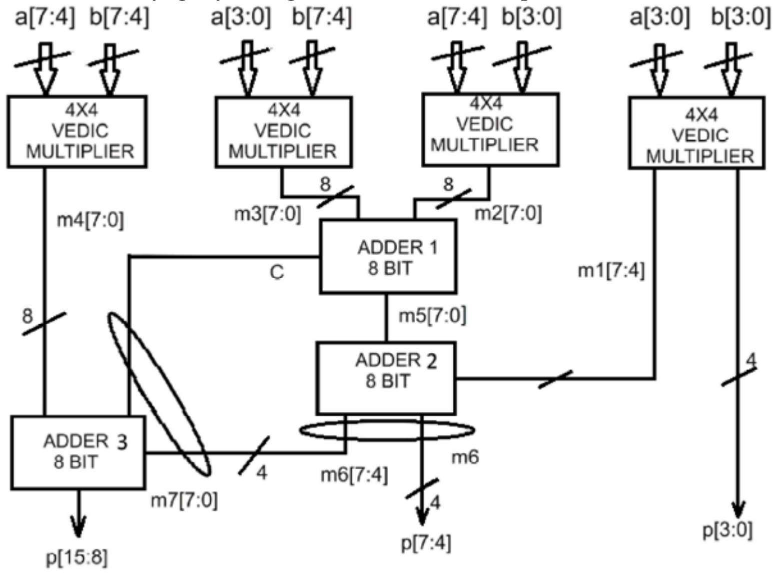

Fig. 10: 8-bit Vedic multiplier based on Urdhava Tiryagbhyam algorithm [9].

CLA based architecture provides high performance with low power consumption. The proposed RCA based mantissa multiplier architecture requires 12.18% less area; improves 14.20% performance and consumes 5.14% less power than the RCA based architecture of [5]; when implemented as per ASIC domain methodology. Similarly proposed CLA based mantissa multiplier architecture results in 11.04% lesser area, 8.64% higher performance and 4.99% low power consumption than the regular CLA based mantissa multiplier architecture of [5] ; when implemented as per ASIC domain methodology.

**Application: Low Power Architectures in Vedic Multiplier:** Proposed RCA architecture was applied to the vedic multiplier and its impact is observed for another algorithm. Vedic multiplier uses the Urdhava Tiryagbhyam sutra of Ancient Indian mathematics, which reduces computational time of the time of multiplier. It is also called Vertical and cross wise algorithm. Figure 9 shows the 4bit multiplier representation using vertical cross wise algorithm. Here the input bits (A[3:0] and B[3:0]) are divided to two equal parts and are called as MSB(H) and LSB(L) parts. Then the LSBs of both inputs (A & B) are multiplied to get the First part of product. Second part contains the multiplication of LSBs to MSBs and multiplication of MSBs to LSBs followed by the addition of both products. Next MSBs of both the inputs are multiplied to get the third part of the product. Finally through proper bit positioning all three parts are added to arrive at final product [9].

Hence to implement 4-bit multiplication using Vedic sutra four 2-bit multipliers are required. Extended the concept to 8-bit multiplication requires four 4-bit multipliers as shown in Figure 10, where each 4-bit multiplier is deduced from four 2-bit multipliers.

In vedic multiplication also, adders are required to compute and hence the proposed adder architecture of Figure 5, is incorporated in 8-bit multiplier and its impact is observed. Results of this analysis are tabulated in Table 4.

Table 4 gives the results for the 2, 4 & 8-bits existing and proposed vedic multiplier architectures. From Table 4, it proves that the proposed adder architectures behave similarly and yields better results than the existing

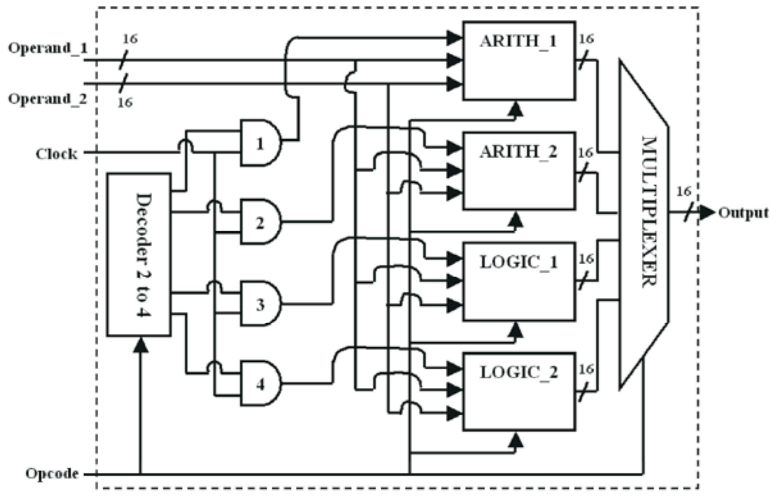

Fig. 11: 16-bit ALU architecture performing 16-operations [10]

Table 4: Existing and Proposed 2, 4 & 8-bits Urdhava Tiryagbhyam algorithm based Vedic multiplier architecture

| | 2-bits | | | 4-bits | | | 8-bits | | |
|---|---|---|---|---|---|---|---|---|---|
| Design bit-width | Regular | Proposed | % Change | Regular | Proposed | % Change | Regular | Proposed | % Change |
| Area (Sq. microns) | 20.16 | 18.72 | 7.14 | 185.76 | 161.64 | 12.98 | 969.12 | 832.68 | 14.08 |
| Delay (ns) | 0.23 | 0.2 | 13.04 | 0.99 | 0.82 | 17.17 | 2.24 | 1.86 | 16.96 |
| Dp (micro watt) | 1.887 | 1.759 | 6.78 | 24.268 | 22.895 | 5.66 | 173.689 | 165.403 | 4.77 |
| Lp (micro watt) | 0.185 | 0.169 | 8.65 | 1.81 | 1.486 | 17.90 | 9.565 | 7.724 | 19.25 |
| Tp (micro watt) | 2.072 | 1.928 | 6.95 | 26.078 | 24.381 | 6.51 | 183.254 | 173.127 | 5.53 |

Note: "Dp" is Dynamic power; "Lp" is Leakage power and "Tp" is Total power

architectures in vedic multiplier. Table 4 suggests that the impact of the proposed architectures will be higher as the number of input bits increases.

**Application: Low Power Architectures in Arithmetic & Logic Unit (ALU):** ALU is one of the important datapath elements of microprocessors and they perform arithmetic and logical operations. It is the sub-system of the microprocessor and it satisfies particular performance or the price constraints of the micro-process design constraint. For implementation and to illustrate the behavior of the proposed adder architectures at the sub-system level, ALU of 16-bit performing 16 operations are considered. Figure 11 shows the architecture of the 16-bit ALU and Table 5 shows the operation involved in the ALU. Of the 16 operations, 4 blocks (ARITH_1, ARITH_2, LOGIC_1and LOGIC_2) are created; each performing 4 operations, to switch off the un-computing blocks completely during the run-time through clock-gating; so that enormous amount of power is saved [10] 8-bit Vedic multiplier with regular and proposed adder architectures is incorporated in the ALU design to perform the 8bit-multiplication operation. A result of this analysis is provided in Table 6.

Table 5: ALU functions

| Opcode | Operation | Active block |
|---|---|---|
| 0000 | Addition | ARITH_1 |
| 0001 | Subtraction | |
| 0010 | Increment | |
| 0011 | Decrement | |
| 0100 | Multiplication | ARITH_2 |
| 0101 | Add with Carry | |
| 0110 | Clear register | |
| 0111 | Set register | |
| 1000 | NOT | LOGIC_1 |
| 1001 | AND | |
| 1010 | OR | |
| 1011 | EX-OR | |
| 1100 | Shift left | LOGIC_2 |
| 1101 | Shift right | |
| 1110 | Rotate left | |
| 1111 | Rotate right | |

Table 6: Existing and Proposed 16-bit ALU design based on regular & proposed adder architecture in vedic multiplier

| | 16-bits | | |
|---|---|---|---|
| Design bit-width | Regular | Proposed | % Change |
| Area (Sq. microns) | 1974.24 | 1837.8 | 6.91 |
| Delay (ns) | 2.33 | 1.95 | 16.31 |
| Dp (micro watt) | 37.045 | 36.229 | 2.20 |
| Lp (micro watt) | 18.529 | 16.686 | 9.95 |
| Tp (micro watt) | 55.574 | 52.915 | 4.78 |

Note: "Dp" is Dynamic power; "Lp" is Leakage power and "Tp" is Total power

Table 6 gives the results for the 16-bit ALU design. Here multiplier architecture is implemented using Vedic sutra and proposed adder architectures are incorporated only to this multiplier. Table 6 shows the impact of proposed adder is higher and can be further increased by incorporating the proposed adder architectures wherever possible in the ALU design (ARITH_1 and ARITH_2 blocks). Such a fully optimized ALU design creates new corners for analysis and increases the range of design space exploration. Proposed ALU design gains 6.91% area, provides 16.31% more performance and consumes 4.78% less power than the regular ALU design.

From all the above discussions in results section, it can be summarized that the proposed architectures will create new technological corners and provides flexible design constraints to choose the architectures as per the applications. It also increases the range of design space exploration.

Mantissa multiplication in double precision floating point representation is implemented in this brief. Low power architectures are proposed for the karatsuba algorithm utilized in the mantissa multiplication. Variable pipelining and carry prediction are proposed in the partitioned multipliers of karatsuba algorithm for performance improvement. Proposed architectures yield similar results in the datapath and sub-system level; as in the multiplier and ALU designs. Hence the proposed architectures can be utilized at any hierarchical level and for any bit-widths in the design cycle. Proposed architectures creates new technological corners like less area - low power, high performance, low area – low power – high performance and etc for analysis and selection. It increases the range of design space exploration to choose the architectures as per the applications. Proposed run time reconfigurable mantissa multiplier of the extended double precision floating point multiplier requires 12.18% less area and improves 14.20% performance with 5.14% less power.

## CONCLUSION

Floating point arithmetic's are widely used in commercial applications like financial analysis, banking, tax calculation, currency conversion, insurance & accounting and also in large set of scientific, numeric and signal processing computations. In digital signal processors (DSP), IEEE 754 compliant floating point arithmetic is used for large dynamic range or for rapid prototyping. In floating point arithmetic, multiplication is the most commonly used arithmetic. In this brief, low power floating point multiplier is presented (LPFM). LPFM utilizes the 2 partition Karatsuba algorithm and supports run-time reconfigurability with dual single precision or single double precision.

## REFERENCES

1. Kwon, Taek-Jun, Jeff Sondeen and Jeff Draper, 2005. Design trade-offs in floating-point unit implementation for embedded and processing-in-memory systems, Circuits and Systems, 2005. ISCAS 2005. IEEE International Symposium on. IEEE.
2. Galal, S. and M. Horowitz, 2011. Energy-Efficient Floating-Point Unit Design, Computers, IEEE Transactions, 60(7): 913-922.
3. Ghosh, Somsubhra, Prarthana Bhattacharyya and Arka Dutta, 2013. FPGA based implementation of a double precision IEEE floating-point adder, Intelligent Systems and Control (ISCO), 2013 7th International Conference, pp: 271-275.
4. Jeevan, B., S. Narender, C.V.K.Reddy and K. Sivani, *2013.* A high speed binary floating point multiplier using Dadda algorithm, Automation, Computing, Communication, Control and Compressed Sensing (iMac4s), International Multi-Conference, 455(460): 22-23.
5. Jaiswal, Manish Kumar and Ray C.C. Cheung, 2013. Area-efficient architectures for double precision multiplier on FPGA, with run-time-reconfigurable dual single precision support, Microelectronics Journal, 44(5): 421-430.
6. Karatsuba, A. and Y. Ofman, 1962. Multiplication of many-digital numbers by automatic computers, in: Proceedings of the USSR Academy of Sciences, 145: 293-294.
7. Weste, N., 2008. CMOS VLSI Design- A Circuits & System Perspective, Pearson Education.
8. ChandraMohan, U., 2004. High Speed Squarer, Proceedings of the 8th VLSI Design and Test Workshops, VDAT.
9. Premananda, B.S., S. Pai Samarth, B. Shashank and S. Bhat Shashank, 2013. Design and Implementation of 8-bit Vedic Multiplier, International Journal of Advanced Research in Electrical, Electronics and Instrumentation, Engineering, 2(12): 5877-5882.
10. Kamaraju, M., M. Kishore Lal and A.V.N. Tilak, 2010. Power Optimized ALU for Efficient Datapath, International Journal of Computer Applications, 11(11): 0975-8887.