Partial Product Perforation Technique for Efficient High Speed and Low Power Approximate Multiplication Circuits

[1]S.Divya, [2]Dr.O.Saraniya

[1]PG Scholar, [2]Assistant Professor
Department of Electronics and Communication Engineering
Government College Of Technology, Coimbatore, India

***Abstract*—Partial product perforation is a technique used for designing approximate multiplication circuits by omitting the generation of some partial products. Product perforation technique is applied on different multiplier circuits, i.e., array multiplier, Wallace tree multiplier, DADDA multiplier and the optimal architecture is exposed. Simulation results shows that the partial product perforation method reduces, area by 45%, power by 50% and critical delay by 35% than accurate multiplier. DADDA multiplier results as the best multiplier and it is applied in 10 TAP FIR filter. In addition, the product perforation method is compared with state-of-the-art approximation circuits, i.e., truncation, voltage overscaling, and logic approximation, showing that it outperforms them in terms of power dissipation and error. This technique is scalable, offering better results as multipliers bit width increases. Multiplication process is often used in digital signal processing system, microprocessors design, communication system, and other application specific integrated circuits. Multipliers have complex units and play an important role in deciding the overall area, speed and power consumption of digital design.**

Keywords—*Partial product perforation, Dadda multiplier, 10 TAP FIR filter.*

I. INTRODUCTION

In today's digital signal processing multipliers plays a major role and various other applications. Essential design targets of multiplier include low power consumption,high speed, regularity of layout and hence has less area or even combination of them in one multiplier are required thereby making them suitable for various VLSI implementations.The straightforward way to implement a multiplication is given by iterative adder-accumulator for the generated partial products. Which is called a serial multiplier. However, solution of this is quite slow as the final result is only available after 'n' clock cycles, where n is the size of the operands. Serial multipliers are used where area and power are of utmost importance and increased delay can be tolerated. A faster version of the iterative multiplier must add partial products at once. This could be achieved by unfolding the iterative multiplier and yielding a combinational circuit consists of several partial product generators together with several adders operates in parallel which is called Parallel Multiplier .

The product is the result of multiplying the multiplicand and the multiplier. The multiplication operation is performed in two main steps. First is the partial product generation, which is done by AND-ing each bit of the multiplier with the multiplicand. Each successive partial product has a relative shift of one bit position to the left of the previous partial product. The second step is the partial product accumulation, where the partial product is combined using adders and compressors to find the result. Most multiplication techniques can be classified as Array multipliers and Tree multipliers. A detailed discussion on different types of multipliers is done in the following sections.

Now a days in modern embedded electronic devices, power consumption is a first-class design concern. A large number of application domains are inherently tolerant to imprecise calculations, e.g., digital signal processing (DSP), data analytics, and data-mining [1], approximate computing appear as a promising solution to reduce their power dissipation. Approximate computing can be applied at both software and hardware levels. Hardware-level approximation mainly targets arithmetic units, such as multipliers and adders, widely used in portable devices to implement multimedia algorithms, e.g., image and video processing. The most commonly used techniques for the generation of approximate arithmetic circuits are truncation [4], [5],simplification of logic complexity (i.e., alteration of the truth table) [7]–[9] and voltage overscaling (VOS) [2], [6]. Extensive research has been conducted on approximate adders [6], [7], [10], [11], providing significant gains in terms of area and power while exposing small error. However, research activities on approximate multipliers are limited. Efficient approximate multipliers introduced in [8], [9], [12], and [13] target the approximation of the partial product accumulation but do not examine approximations on the partial product generation. Approximate hardware circuits, contrary to software approximations, offer transistors reduction, leakage power and lower dynamic, lower circuit delay, and opportunity for downsizing. Motivated by the limited research on approximate multipliers, compared with the extensive research on approximate adders, and explicitly the lack of approximate techniques targeting the partial product generation, we introduce the partial product perforation method for creating approximate multipliers. Inspired from [14], we omit the generation of some partial products, thus reducing the number of partial products that have to be accumulated, we decrease the area, power, and depth of the accumulation tree.

Approximate circuits has been considered for error-tolerant applications that can tolerate some loss of accuracy with improved performance and energy efficiency. Multipliers is a key arithmetic circuits in many such applications such as digital signal processing. In this paper, a novel approximate multiplier with lower power consumption and a shorter critical path than traditional multipliers is proposed for high-performance DSP applications.

II. PARTIAL PRODUCT REDUCTION

The heart of an efficient digital multiplier implementation is based on the manner in which the PPA bits are added. Were conventional carry adders used to implement these add operations, the delay of all the adders would consume a large amount of time, as each shifted version of the multiplicand would contribute a delay which is proportional to the width of the multiplicand. Instead, the partial product is reduced using a technique called carry-save addition [31]. This procedure allows successive additions to be incorporated into one global addition step. Consider a numerical bit vector representation of the following form: (bn-1,bn-2,...,b1,b0), bi = {0,1}, . If we wish to add two bits from two bit vectors, say a0 and b0, from bit vectors a and b, we can use the full adder ; it takes in three bits and produces a sum bit, and a carry bit. When adding two vectors together, this block can be used to add two bits at a given bit position with the carry-in from the previous bit position.

Consider the case where two bit vectors are to be added. At the lowest bit position, two bits are added, and the carry is propagated to the next bit position. From then on the carry-in and the next two bits at the higher bit position are combined, and a carry-out is generated. Using this rippling technique, we see that adding two n-bit number takes O (n) sequential bit additions, resulting in a delay of O (n). If we have to add three bit vectors, A, B, and C, each of size n, we can use this method to add first A and B, and then to add C to the result of A+B. The number of bit additions is O (2n). We see that if we were to use this technique in the most simple-minded fashion to add n shifted copies of an n-bit multiplicand, the delay will be O (n2).

III. ARCHITECTURE OFARRAY, WALLACE TREE AND DADDA MULTIPLIER

A. Array Multiplier

Array multipliers can also be implemented by directly mapping the manual multiplication into hardware.

The partial products are accumulated by an array of adder circuits. An n x n array multiplier requires n (n-1) adders and n 2 AND gates. 8 X 8 accurate array multiplier and approximate array multiplieris shown in the figure. Array multipliers have a large critical path and is very slow. The main advantage of this multipliers is the regular structure which leads to ease of layout and design.The composition of an array multiplier is shown in the figure.There is a one to one topological correspondence between this hardware structure.

The generation of n partial products requires N*M two bit AND gates. Most of the area of the multiplier is devoted to the adding of n partial products, which requires N-1, M-bit adders. The shifting of the partial products for their proper alignment's performed by simple routing and does not require any logic. The overall structure can be easily being compacted into rectangle, resulting in very efficient layout.

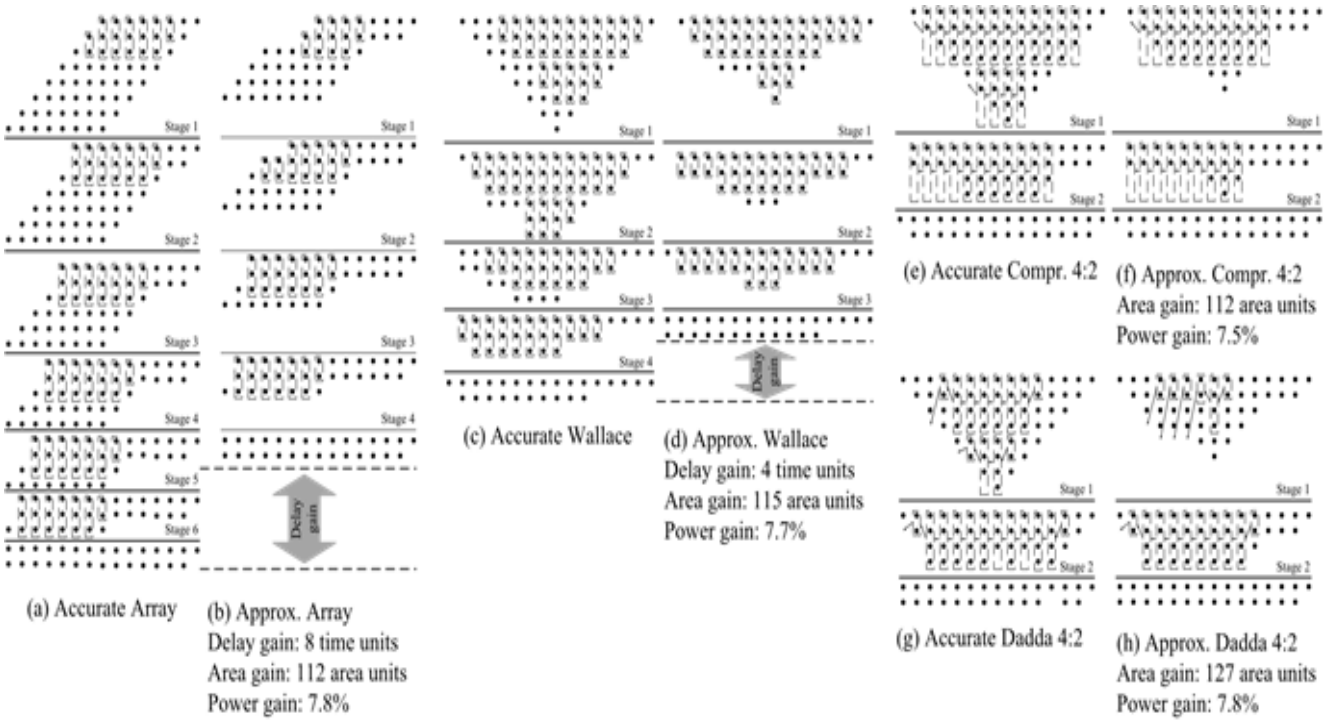
B. Wallace Tree Multiplier

A fast technique to perform multiplication is Propounded by C.S. Wallace for large operands Wallace tree multiplier offers faster performance. Unlike an array multiplier the partial product matrix for a tree multiplier is rearranged in a tree-like format, by reducing both the critical path and the number of adder cells needed.

The Wallace tree multiplier comes under a family of multipliers called column compression multipliers. The underlying principle used in this family of multipliers is to achieve partial product accumulation by successively reducing the number of information bits in each column using full adders or half adders. The full adder is known as a (3:2) compressor which has a ability to add 3 bits from a single column of the partial product matrix and output 2 bits, from the output 2 bits 1 bit is placed the same column and 1 bit in the next column of the output matrix. The half adder is known as a (2:2) compressor which has a ability to take 2 bits from a single column of the partial product matrix and output 2 bits, from the output 2 bits 1 bit is placed in the same column and 1 bit in the next column of the output matrix.

C. DADDA Multiplier

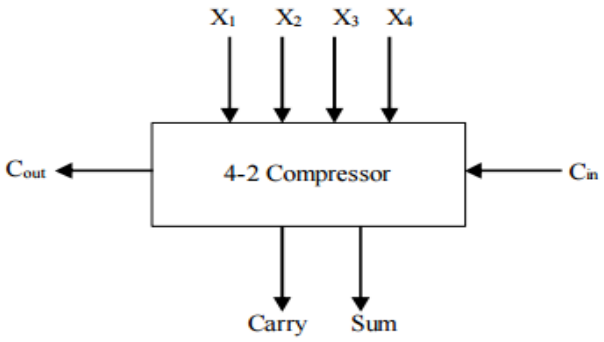
DADDA multiplier is a hardware multiplier designed similar to Wallace multiplier. Unlike Wallace multipliers that perform reductions as much as possible on each layer, Dadda multipliers do as few reductions as possible. Due to this, Dadda multipliers have less expensive reduction phase, but the numbers may be a few bit longer, thus requiring slightly bigger adders. This implies that fewer columns are compressed in the initial stages of the column compression tree, and more columns in the later levels of the multiplier. In this figure, the plain diagonal line connecting the two dots represents the outputs of the full adder while, the outputs of the half adder are represented by crossed diagonal lines connecting two dots.4:2 Compressor is a high speed compressor structures which perform more than three bit addition in parallel reducing number of stages in partial product reduction tree, minimizing the critical delay.



The 4:2 compressor has four inputs and two outputs. The architecture of 4:2 compressor is composed of two serially connected full adders. The four inputs X1, X2, X3 and X4 and the output Sum are from same column of the partial product. Cin is the output carry of the preceding module and Cout is the carry output of the current compressor fed as Cin to the next compressor module.

The compressor is governed by Equation (4.1).

X1+X2+X3+X4+Cin=Sum+2(Carry+Cout) (4.1)



$$Sum=(X_1\oplus X_2).(X_3\oplus X_4)'+(X_1\oplus X_2)'.(X_3\oplus X_4).C_{in}' + [(X_1\oplus X_2).(X_3\oplus X_4)'+(X_1\oplus X_2)'.(X_3\oplus X_4)]'.C_{in} \quad (4.2)$$

$$C_{out}=(X_1\oplus X_2).X_3+(X_1\oplus X_2)'.X_1 \quad (4.3)$$

$$Carry=(X_1\oplus X_2\oplus X_3\oplus X_4).C_{in}+(X_1\oplus X_2\oplus X_3\oplus X_4)'. \quad (4.4)$$

Fig.1 *Block diagram of 4:2 compressor*

IV. SIMULATION RESULTS

To summarize the performance measure of different multiplier architectures i.e, array multiplier, wallace tree multiplier , DADDA multiplier, it is simulated using XILINX ISE Design Suite 14.2. The area, power and delay of different multiplier architectures were analysed and from the simulation results it is shown that DADDA multiplier outperforms best than other multipliers.

TABLE 1:AREA, POWER AND DELAY SUMMARIZATION TABLE

	ACCURATE MULTIPLIERS			APPROXIMATE MULTIPLIERS		
	ARRAY	WALLACE TREE	DADDA	ARRAY	WALLACE TREE	DADDA
INPUT SIZE(N)	8	8	8	8	8	8
STAGES(S)	6	4	2	4	3	2

AREA (LUT'S)	122 OUT OF 12480	103 OUT OF 12480	124 OUT OF 12480	87 OUT OF 12480	82 OUT OF 12480	70 OUT OF 12480
POWER(W)	0.326	0.325	0.463	0.324	0.324	0.323
DELAY(ns)	10.976	11.951	10.747	10.142	11.130	10.194

To calculate area, power and delay Run synthesize-XST followed by this Run implement design once it is simulated open synthesis report to calculate area and delay, to calculate power click tools and select Xpower Analyzer update signal rate and update power analysis now total power is given by summary table.

```
Slice Logic Utilization:
 Number of Slice LUTs:                  124  out of  12480     0%
    Number used as Logic:               124  out of  12480     0%

Slice Logic Distribution:
 Number of LUT Flip Flop pairs used:    124
   Number with an unused Flip Flop:     124  out of    124   100%
   Number with an unused LUT:             0  out of    124     0%
   Number of fully used LUT-FF pairs:     0  out of    124     0%
   Number of unique control sets:         0

IO Utilization:
 Number of IOs:                          32
 Number of bonded IOBs:                  32  out of    172    18%
```

```
Delay:               10.747ns (Levels of Logic = 12)
  Source:            a<5> (PAD)
  Destination:       m<14> (PAD)

 Data Path: a<5> to m<14>
                                    Gate     Net
   Cell:in->out      fanout        Delay    Delay  Logical Name (Net Name)
   ----------------------------------------  -----------
    IBUF:I->O             19       0.694    0.571  a_5_IBUF (a_5_IBUF)
    LUT2:I0->O             3       0.086    0.496  n1/n1/p1_5_and00001 (n1/w1<5>)
    LUT6:I4->O             2       0.086    0.905  n1/n2/cout1 (n1/c<1>)
    LUT6:I0->O             3       0.086    0.496  n1/n3/Mxor_s_Result1 (w1<6>)
    LUT5:I3->O             2       0.086    0.823  n6/cout1 (c<4>)
    LUT5:I0->O             2       0.086    0.823  n7/Mxor_s_Result1 (p1<7>)
    LUT5:I0->O             3       0.086    0.609  h7/cout1 (c2<6>)
    LUT5:I2->O             3       0.086    0.609  h9/cout1 (c2<8>)
    LUT5:I2->O             3       0.086    0.609  h11/cout1 (c2<10>)
    LUT3:I0->O             2       0.086    0.823  h12/cout1 (c2<11>)
    LUT6:I1->O             1       0.086    0.286  h14/Mxor_s_xo<0>1 (m_14_OBUF)
    OBUF:I->O                      2.144            m_14_OBUF (m<14>)
   ----------------------------------------
   Total                          10.747ns (3.698ns logic, 7.049ns route)
                                           (34.4% logic, 65.6% route)
```

Fig.2 *Simulation result ofAccurate dadda area*Fig.3 *Simulation result ofAccurate dadda delay*

On-Chip	Power (W)	Used	Available	Utilization (%)
Logic	0.002	111	12480	1
Signals	0.005	140	---	---
IOs	0.133	32	172	19
Leakage	0.322			
Total	0.463			

Thermal Properties	Effective TJA (C/W)	Max Ambient (C)	Junction Temp (C)
	2.8	83.7	51.3

```
Slice Logic Utilization:
 Number of Slice LUTs:                   70  out of  12480     0%
    Number used as Logic:                70  out of  12480     0%

Slice Logic Distribution:
 Number of LUT Flip Flop pairs used:     70
   Number with an unused Flip Flop:      70  out of     70   100%
   Number with an unused LUT:             0  out of     70     0%
   Number of fully used LUT-FF pairs:     0  out of     70     0%
   Number of unique control sets:         0

IO Utilization:
 Number of IOs:                          32
 Number of bonded IOBs:                  32  out of    172    18%
```

Fig.4 *Simulation result ofAccurate dadda power*Fig.5 *Simulation result ofApproximatedadda area*

```
Delay:               10.194ns (Levels of Logic = 11)
  Source:            a<3> (PAD)
  Destination:       m<14> (PAD)

 Data Path: a<3> to m<14>
                                    Gate     Net
   Cell:in->out      fanout        Delay    Delay  Logical Name (Net Name)
   ----------------------------------------  -----------
    IBUF:I->O             13       0.694    0.543  a_3_IBUF (a_3_IBUF)
    LUT2:I0->O             2       0.086    0.905  n1/n1/p1_3_and00001 (w1<3>)
    LUT6:I0->O             2       0.086    0.905  h3/cout1 (c2<2>)
    LUT6:I0->O             2       0.086    0.905  h4/cout1 (c2<3>)
    LUT6:I0->O             3       0.086    0.609  h5/cout1 (c2<4>)
    LUT5:I2->O             3       0.086    0.609  h7/cout1 (c2<6>)
    LUT5:I2->O             3       0.086    0.609  h9/cout1 (c2<8>)
    LUT5:I2->O             3       0.086    0.609  h11/cout1 (c2<10>)
    LUT5:I2->O             2       0.086    0.604  h13/cout1 (c2<12>)
    LUT5:I2->O             1       0.086    0.286  h14/Mxor_s_xo<0>1 (m_14_OBU
    OBUF:I->O                      2.144            m_14_OBUF (m<14>)
   ----------------------------------------
   Total                          10.194ns (3.612ns logic, 6.583ns route)
                                           (35.4% logic, 64.6% route)
```

On-Chip	Power (W)	Used	Available	Utilization (%)
Logic	0.001	60	12480	0
Signals	0.002	86	---	---
IOs	0.000	32	172	19
Leakage	0.321			
Total	0.323			

Thermal Properties	Effective TJA (C/W)	Max Ambient (C)	Junction Temp (C)
	2.8	84.1	50.9

Fig.6 *Simulation result ofApproximatedadda delay*Fig.7*Simulation result ofApproximatedadda power*

V. EXPERIMENTAL STUDY

In this section, the Partial product perforation technique is applied on different multiplier architectures i.e, array, wallace tree and dadda multiplier to explore thier power consumption, area and delay. The resultant output of area, power and delay is plotted in a pareto graph shown in the fig

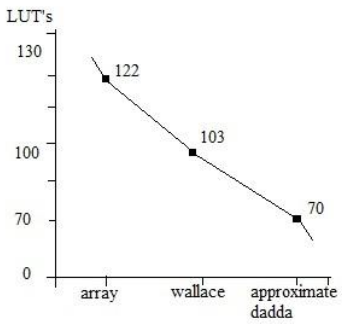


Fig.8 *plots of area in LUT's*

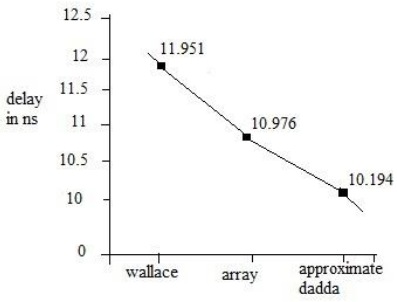


Fig.9 *plots of power in watts*

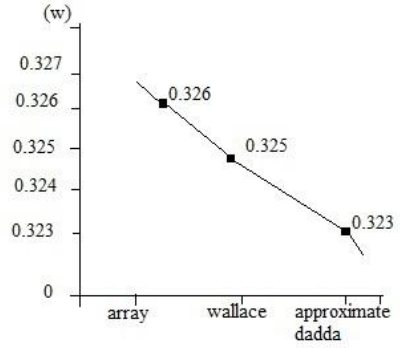


Fig.10 *plots of delay in ns*

Multiplier is one of the DSP and image processing applications etc., Further dadda multiplier is applied on 10Tap fir filter replacing normal multiplier and comparative study is made between ordinary 10Tap fir filter and modified 10Tap fir filter. The parameters that are used for comparisons include area, power and delay. The comparison table is given below

TABLE 2:COMPARITIVE TABLE BETWEEN ORDINARY 10TAP AND MODIFIED 10TAP FIR FILTER

NORMAL 10TAP FIR FILTER			MODIFIED 10TAP FIR FILTER(WITH DADDA MULTIPLIER)		
AREA IN LUT's	POWER IN WATT	DELAY IN ns	AREA IN LUT's	POWER IN WATT	DELAY IN ns
101 OUT OF 126800	0.048	2.284	101 OUT OF 126800	0.049	1.959

From the table we can infer that probably there will be a trade off between area, power and delay but here on replacing normal multiplier with a dadda multiplier results in very less delay. Thus this modified filters can be used for high speed DSP applications with less area and power. The following figure shows the simulation results of modified 10Tap fir filter.

```
Device utilization summary:
---------------------------

Selected Device : 7a100tcsg324-3


Slice Logic Utilization:
 Number of Slice Registers:             101  out of  126800     0%
 Number of Slice LUTs:                  254  out of  63400      0%
    Number used as Logic:               254  out of  63400      0%

Slice Logic Distribution:
 Number of LUT Flip Flop pairs used:    261
   Number with an unused Flip Flop:     160  out of    261     61%
   Number with an unused LUT:             7  out of    261      2%
   Number of fully used LUT-FF pairs:    94  out of    261     36%
   Number of unique control sets:         1

IO Utilization:
 Number of IOs:                          25
 Number of bonded IOBs:                  25  out of    210     11%

Specific Feature Utilization:
 Number of BUFG/BUFGCTRLs:               1  out of     32      3%
```

On-Chip	Power (W)	Used	Available	Utilization (%)
Clocks	0.000	1	---	---
Logic	0.005	214	63400	0
Signals	0.002	260	---	---
IOs	0.000	25	210	12
Leakage	0.042			
Total	0.049			

Thermal Properties	Effective TJA (C/W)	Max Ambient (C)	Junction Temp (C)
	3.3	84.8	25.2

Fig.11 *area of modified 10tap fir filter*

Fig.12 *power of modified 10tap fir filter*

```
Delay:               1.959ns (Levels of Logic = 3)
  Source:            k6/n8/q (FF)
  Destination:       k7/n12/q (FF)
  Source Clock:      clk rising
  Destination Clock: clk rising

  Data Path: k6/n8/q to k7/n12/q
                              Gate     Net
    Cell:in->out      fanout   Delay   Delay  Logical Name (Net Name)
    ----------------------------------------  ------------
     FD:C->Q              3   0.361   0.521  k6/n8/q (k6/n8/q)
     LUT6:I3->O           5   0.097   0.398  g6/n1/x6/c1_SW1 (N19)
     LUT6:I4->O           1   0.097   0.379  g6/n1/x8/c1_SW6 (N105)
     LUT6:I4->O           1   0.097   0.000  g6/n2/x4/Mxor_s_xo<0>1 (y6<11>)
     FD:D                     0.008          k7/n12/q
    ----------------------------------------
    Total                     1.959ns (0.660ns logic, 1.299ns route)
                                      (33.7% logic, 66.3% route)
```

Fig.13 *delay of modified 10tap fir filter*

VI. CONCULSION AND FUTURE WORK

In this design strategy, the multiplier circuit is redesigned into two different parts.Accurate part, which is implemented using standard partial multiplier to achieve greater accuracy.Approximate part, which generates small amount of error but this is acceptable error value to circuit designer/customer because with small sacrifice of accuracy the design provides noticeable saving in power and high speed operation.such multipliers are widely used in wireless communication and multimedia. It offers better results as the multiplier's bit width increases.Comparing all multipliers DADDA multiplier is best in reducing the area, power and delay.Multiplication process is often used in digital signal processing systems, microprocessors designs, communication systems, and other application specific integrated circuits. Here the improved multiplier is applied on 10Tap fir filter. Further, it can be applied on higher tap fir filters for high speed low power DSP applications.

REFERENCES

[1] V. K. Chippa, S. T. Chakradhar, K. Roy, and A. Raghunathan, "Analysis and characterization of inherent application resilience for approximate computing," in *Proc. 50th ACM/EDAC/IEEE Design Autom. Conf.*, May/Jun. 2013, pp. 1–9.

[2] R. Venkatesan, A. Agarwal, K. Roy, and A. Raghunathan, "MACACO: Modeling and analysis of circuits for approximate computing," in *Proc.IEEE/ACM Int. Conf. Comput.-Aided Design*, Nov. 2011, pp. 667–673.

[3] S. T. Chakradhar and A. Raghunathan, "Best-effort computing: Re-thinking parallel software and hardware," in *Proc. 47th ACM/IEEEDesign Autom. Conf.*, Jun. 2010, pp. 865–870.

[4] V. Gupta, D. Mohapatra, S. P. Park, A. Raghunathan, and K. Roy, "IMPACT: Imprecise adders for low-power approximate computing," in *Proc. Int. Symp. Low Power Electron. Design*, Aug. 2011, pp. 409–414.

[5] C. Liu, J. Han, and F. Lombardi, "A low-power, high-performance approximate multiplier with configurable partial error recovery," in *Proc.Conf. Design, Autom. Test Eur.*, Mar. 2014, Art. no. 95.

[6] A. Momeni, J. Han, P. Montuschi, and F. Lombardi, "Design and analysis of approximate compressors for multiplication," *IEEE Trans.Comput.*, Vol. 64, No. 4, pp. 984–994, Apr. 2015.

[7] P. Kulkarni, P. Gupta, and M. Ercegovac, "Trading accuracy for power with an underdesigned multiplier architecture," in *Proc. 24th Annu. Conf.VLSI Design*, Jan. 2011, pp. 346–351.

[8] S. Narayanamoorthy, H. A. Moghaddam, Z. Liu, T. Park, and N. S. Kim, "Energy-efficient approximate multiplication for digital signal processing and classification applications," *IEEE Trans. Very Large ScaleIntegr. (VLSI) Syst.*, Vol. 23, No. 6, pp. 1180–1184, Jun. 2015.

[9] C. Lee, M. Potkonjak, and W. H. Mangione-Smith, "MediaBench: A tool for evaluating and synthesizing multimedia and communications systems," in *Proc. 13th Annu. IEEE/ACM Int. Symp. Microarchitecture*, Dec. 1997, pp. 330–335.

[10] G. Zervakis, K. Tsoumanis, S. Xydis, N. Axelos, and K. Pekmestzi, "Approximate multiplier architectures through partial product perforation: Power-area tradeoffs analysis," in *Proc. 25th Great Lakes Symp.VLSI*, 2015, pp. 229–232.

[11] D. Soudris, C. Piguet, and C. Goutis, *Designing CMOS Circuits forLow Power*, ser. European low-power initiative for electronic systemdesign. Springer, 2002.

[12] R. Zimmermann and W. Fichtner, "Low-power logic styles: Cmos versus pass-transistor logic," *Solid-State Circuits, IEEE Journal of*, vol. 32, no. 7, pp. 1079–1090, Jul 1997.

[13] A. Salz and M. Horowitz, "Irsim: An incremental mos switch-levelsimulator," in *Design Automation, 1989. 26th Conference on*, June 1989, pp. 173–178.

[14] D. Mohapatra, "Approximate computing: Enabling voltage over-scaling in multimedia applications," Ph.D. dissertation, Purdue University, 2011.

[15] S. Venkataramani, A. Sabne, V. J. Kozhikkottu, K. Roy, and A. Raghunathan, "Salsa: systematic logic synthesis of approximate circuits," in *The 49th Annual Design Automation Conference 2012, DAC '12*. ACM, 2012, pp. 796–801.