Power- and Area-Efficient Approximate Wallace Tree Multiplier for Error-Resilient Systems

Kartikeya Bhardwaj
Electrical & Electronics Engg.
BITS Pilani-Goa Campus
Goa – 403 726, India
Email: bhardwajkartikeya@gmail.com

Pravin S. Mane
Electrical & Electronics Engg.
BITS Pilani-Goa Campus
Goa – 403 726, India
Email: pravinm586@gmail.com

Jörg Henkel
Department of Computer Science
Karlsruhe Institute of Technology
Karlsruhe – 76131, Germany
Email: henkel@kit.edu

***Abstract*—Today in sub-nanometer regime, chip/system designers add accuracy as a new constraint to optimize Latency-Power-Area (LPA) metrics. In this paper, we present a new power and area-efficient Approximate Wallace Tree Multiplier (AWTM) for error-tolerant applications. We propose a bit-width aware approximate multiplication algorithm for optimal design of our multiplier. We employ a carry-in prediction method to reduce the critical path. It is further augmented with hardware efficient precomputation of carry-in. We also optimize our multiplier design for latency, power and area using Wallace trees. Accuracy as well as LPA design metrics are used to evaluate our approximate multiplier designs of different bit-widths, $i.e.$ 4×4, 8×8 and 16×16. The simulation results show that we obtain a mean accuracy of 99.85% to 99.965%. Single cycle implementation of AWTM gives almost 24% reduction in latency. We achieve significant reduction in power and area, $i.e.$ up to 41.96% and 34.49% respectively that clearly demonstrates the merits of our proposed AWTM design. Finally, AWTM is used to perform a real time application on a benchmark image. We obtain up to 39% reduction in power and 30% reduction in area without any loss in image quality.**

***Index Terms*—Approximate multiplier; Bit-width aware multiplication algorithm; Wallace tree; Error-resilient systems**

I. Introduction

The International Technology Roadmap for Semiconductors (ITRS) [1] has anticipated imprecise/approximate designs that became a state-of-the art demand for the emerging class of killer applications that manifest inherent error-resilience such as multimedia, graphics, and wireless communications. In the error-resilience systems, adders and multipliers are used as basic building blocks and their approximate designs have attracted significant research interest recently. Conventional wisdom investigated several mechanisms such as truncation [2], over-clocking, and voltage over-scaling(VOS) [3] which could not configure accuracy as well as Latency-Power-Area (LPA) design metrics effectively. Most of the other design techniques rely on functional approximations and a wide spectrum of approximate adders like [4], [5], [6] and [7] have been proposed in the past. However, very few research papers are reported on approximate multipliers in the literature.

Most of the approximate multiplier designs reported shorten the carry-chains in which error is configurable and the algorithms employed in the designs are for smaller numbers and give large magnitude of error as the bit-width of operands increases. In this paper, we present a new Approximate Wallace Tree Multiplier (AWTM) based on a bit-width aware algorithm. We design it specifically to give good results for large operands. Besides accuracy, the AWTM is also optimized for power and area. For single cycle implementation, AWTM gives significant reduction in latency as well. Our contributions are:

- We propose a new power and area-efficient AWTM based on a bit-width aware multiplication algorithm.
- We employ a novel Carry-in Prediction technique which significantly reduces the critical path of our multiplier. We further derive an efficient carry-in precomputation logic to accelerate the carry propagation.
- We obtain a very high mean accuracy of 99.965% (mean error of only 0.035%) when the size of operands are 10 bits or more. However, if there is no lower bound on the size of operands, the mean accuracy varies from 99.85% to 99.9% (a very small mean error of 0.1% to 0.15%).
- We achieve a significant reduction in power and area, *i.e.* up to 41.96%, and 34.49% respectively for the 16-bit accuracy-configurable AWTM design. For single cycle implementation of 16×16 AWTM, we also reduce the latency by around 24%.
- Our proposed AWTM, when used for a real time application on an image, achieved up to 39% reduction in power and up to 30% reduction in area with negligible loss in image quality.

Rest of the paper is organized in various sections. In section 2 we discuss some background and related work reported in literature. We describe some preliminaries in Section 3. An approximate multiplier architecture is explained in Section 4. We propose a bit-width aware approximate multiplication algorithm in Section 5. We present AWTM design based on the proposed methodology and its optimization *w.r.t.* LPA design metrics in Section 6. The experimental results are given in Section 7. Finally, we conclude the paper in Section 8.

II. Background and Related Work

Research on approximate arithmetic circuits mainly reported in the literature is on approximate adders. It is worthwhile to study these approximate adders in order to make research contributions on approximate multipliers. Lu [8] proposed a

k-bit carry look-ahead adder in which only previous k bits are considered to estimate current carry signal. Lu's adder exhibits a low probability of getting correct sum and increases area overhead. Shin et al. [9] reduce data-path delay and re-design the data-path modules. It cuts the critical-path in carry-chain to exploit a given error rate to improve parametric yield.

Zhu et al.[4] manifest an error-tolerant adder: ETA-I. ETA-I divides inputs into: 1) Accurate part, and 2) Inaccurate part. In the latter, no carry signal is considered at any bit position. Gupta et al. [10] target low-power and propose five different versions of mirror adder by reducing the number of transistors and internal node capacitance. Verma et al. [6] presented a Variable Latency Speculative Adder (VLSA) which provides approximate/accurate results but gives considerable delay and large area overhead. Kahng et. al [7] proposed an accuracy-configurable adder with reduced critical-path and error rate.

In contrast with the above work, very few researchers have reported work on approximate multipliers. Sullivan et al. [11] used $Truncated\ Error\ Correction$ (TEC) to investigate an iterative approximate multiplier in which some amount of error correcting circuitry is added for each iteration. This circuitry replicates the effects of multiple pipeline iterations for the most problematic inputs quite inexpensively. Kulkarni et al. [12] proposed a 2×2 underdesigned multiplier block and built arbitrarily power aware inaccurate multipliers. Kyaw et al. [13] presented an Error Tolerant Multiplication (ETM) algorithm in which the input operands are split into two parts. a multiplication part consists of higher order bits and a non-multiplication part with the remaining lower order bits. The multiplication begins at the point where the bits split and move simultaneously towards the two opposite directions till all bits are taken care of. The ETM exhibited a significant reduction in delay, power and hardware cost for specific input combinations. Next, we explain the preliminary concepts so as to understand the proposed approximate multiplier.

III. Preliminaries

We make use of a simple recursive multiplication for our approximate multiplier design and use various accuracy design metrics [7], [13] for its evaluation. The recursive multiplication and the accuracy design metrics are described in the following subsections.

A. Recursive Multiplication

A given multiplication can be recursively broken down into several smaller-size multiplications, each of which can be performed in the same clock cycle. Let A be the multiplicand and X be the multiplier and both are of $2b$ bits each. A and X can also be written as $A = A_H A_L$ and $X = X_H X_L$ where A_H, A_L, X_H, and X_L are of b bits each.

The multiplication $A \times X$, which is $2b \times 2b$, can be recursively carried out as shown in Fig. 1(a). In this multiplication, $A_H X_L$, $A_H X_H$, $A_L X_L$, and $A_L X_H$ are partial products, each of which is a $b \times b$ multiplication. Hence, a $2b \times 2b$ multiplication is divided into four $b \times b$ multiplications

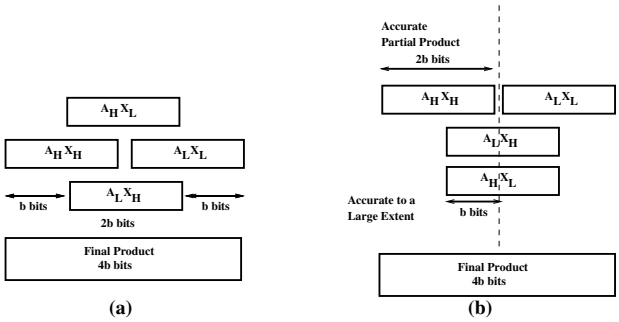

Fig. 1. (a) Recursive Multiplication (b) Approximate Multiplication

followed by additions. Fig. 1(b) is derived from Fig. 1(a) for approximate multiplication.

B. Accuracy Design Metrics

The accuracy design metrics are defined as follows:

1) *Relative Error:* Relative Error can be calculated as $(|R_c - R_e|/R_c) \times 100\%$ for $R_c \neq 0$. Here, R_c is correct result and R_e is approximate result. We denote accuracy as ACC_{amp} where $ACC_{amp} = 1 - Relative\ Error$.
2) *Mean Error:* Mean error is the average of relative errors of all the combinations tested in an algorithm.
3) *Minimum Acceptable Accuracy (MAA):* Minimum Acceptable Accuracy is the minimum level of accuracy that an application can tolerate.
4) *Acceptance Probability (AP):* It is the probability that accuracy of the approximate arithmetic circuit is higher than the minimum acceptable accuracy. Its value is given by $AP = P(ACC_{amp} > MAA)$.

In the next section, we discuss an approximate multiplier architecture to explain the bit-width aware algorithm proposed.

IV. Approximate Multiplier Architecture

In order for the multiplier to exhibit high accuracy, the most significant bits (MSBs) of the final $4b$ bit product ($A \times X$) should be accurate to high extent. Therefore, we make the multiplier $A_H X_H$ as $b \times b$ accurate multiplier and $A_H X_L$, $A_L X_H$, $A_L X_L$ as $b \times b$ approximate multipliers. As we shall see, the $b \times b$ approximate multipliers generate upper b bits as accurate to high extent, which further makes the upper $2b$ bits of final $4b$ bit product achieve high accuracy. The same is illustrated in Fig. 1(b).

We have explained the design methodology of these approximate $b \times b$ multipliers in the Carry-in Prediction Logic[14]. We briefly explain this novel technique in the following subsection with the help of an example.

A. The Carry-in Prediction – An Example

Consider the unsigned multiplication of two 16-bit numbers (*i.e.* $b = 8$):

$$A = (AEDB)_{16} = (44763)_{10}$$

$$X = (B6E7)_{16} = (46823)_{10}$$

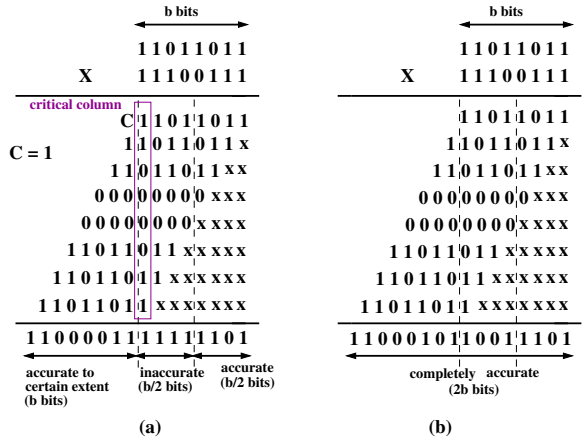

Fig. 2. Carry-in Prediction Example for $b = 8$ (*i.e.* 16×16 multiplication): (a) Approximate $A_L X_L$ (b) Accurate $A_L X_L$

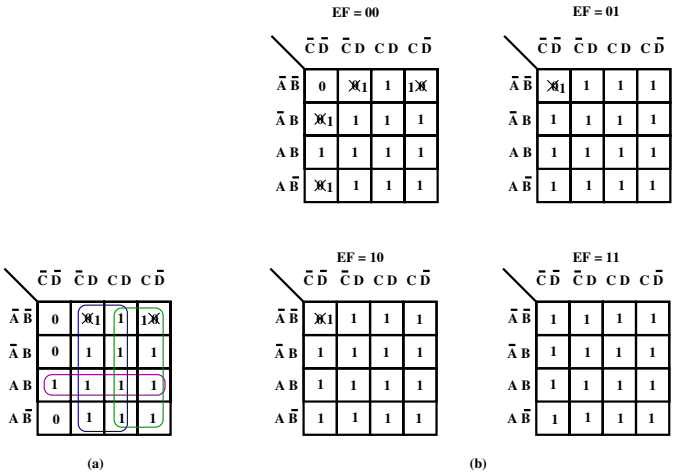

Fig. 3. Carry-in Precomputation for (a) $b = 4$ *i.e.* 8×8 Multiplier (b) $b = 6$ *i.e.* 12×12 Multiplier

Now, let us evaluate one approximate product out of $A_H X_L$, $A_L X_H$ and $A_L X_L$ using our algorithm. Say, we want to evaluate $A_L X_L$ *i.e* $(DB)_{16} \times (E7)_{16}$. As shown in Fig. 2(a), we divide this multiplication in three independent parts: First, accurate computation of $b/2$ least significant bits (LSBs), followed by second part wherein $b/2$ bits are simply set to 1's. The third part is again accurate computation of remaining elements in the multiplication tree with an additional carry 'C' arising from the inaccurate part at least significant position. So, the idea is to precompute 'C' through some mechanism and begin multiplication simultaneously from both first and third part. At the same time, we reduce the number of addition operations involved by directly setting the bits in second part, thus significantly reducing the hardware costs.

Fig. 2(a) further shows a critical column as the column containing maximum number of elements in the multiplication tree. Carry-in Prediction logic exploits the fact that if there are two or more 1's in the critical column, then a carry of at least 1 is definitely propagated to the next column. Next subsection discusses this prediction in more detail. In the second part, we set the $b/2$ bits in the inaccurate part as 1's because for such a large b ($>= 5$), it is very probable that carry propagated from critical column is more than 1. Therefore, setting those bits will reduce the error involved as it is analogous to the difference between 16 ($5'b10000$) and 15 ($5'b01111$) *i.e.* 16 just passes an extra carry.

Fig. 2(b) shows the accurate $A_L X_L$ evaluation. As evident, out of 8 most significant bits (MSBs), 6 are correct in our approximate $A_L X_L$. Evaluating $A_H X_L$ and $A_L X_H$ in a similar fashion and adding all these as indicated in Fig. 1(b) gives approximate result as $(7CEBA7FB)_{16}$. The correct answer is $(7CED799D)_{16}$. The relative error in this case is merely 0.0056%. Precomputation of Carry-in 'C' is described next in detail.

B. Efficient Carry-in Precomputation

Carry-in Prediction necessitates the precomputation of carry-in. Since we are dealing with error resilient systems, we can further simplify and approximate the evaluation of carry-in so that reduction in latency is not achieved at the cost of power. We consider the cases of $b = 4$ and $b = 6$ in order to better explain carry-in precomputation procedure.

The precomputation is made hardware efficient by making minor changes in the K-Maps of the carry-in expressions as shown in Fig. 3. Here $A, B, \ldots, F$ are the elements in critical column. The original K-Maps are obtained from the statement of Carry-in Prediction Logic *i.e.* $C_{in} = 1$ if 2 or more elements of critical column are 1. Fig. 3 further derives

- for $b = 4$: By making changes in 2 cases out of 16, we can simplify the Carry-in expression to

$$C_{in} = A.B + C + D$$

  Similar results can also be derived for $b < 4$.
- for $b = 6$: We make changes in 6 cases out of 64 and get

$$C_{in} = A + B + C + D + E + F$$

  This is same as OR operation of all the elements present in critical column. Therefore, in general, we can state that for large b (greater than 4), one should take the OR of all the elements present in critical column to get C_{in}.

Next, we propose various accuracy configurations and a bit-width aware approximate multiplication algorithm.

V. Bit-Width Aware Approximate Multiplication

In the approximate multiplication, we divide the $b \times b$ accurate multiplier $A_H X_H$ into 4 smaller components, each being a $b/2 \times b/2$ multiplier. This is because, when accurate $A_H X_H$ is performed in parallel with approximate $A_H X_L$, $A_L X_H$ and $A_L X_L$, the critical path will still be determined by the accurate multiplier. Therefore, recursively reducing it to smaller multipliers will make approximate $b \times b$ multipliers as deciding factors of critical path as they are more critical than accurate $b/2 \times b/2$ multipliers.

In other words, the stage 1 of the pipelined approximate multiplier effectively consists of 7 multipliers. The designation

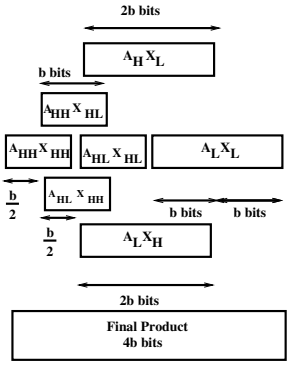

Fig. 4. Latency-Driven Pipelined Approximate Multiplier

TABLE I
MODES OF OPERATION OF ACCURACY CONFIGURABLE MULTIPLIER

Mode	$A_{HH}X_{HH}$	$A_{HH}X_{HL}$	$A_{HL}X_{HH}$	$A_{HL}X_{HL}$
1	A	I	I	I
2	A	A	I	I
3	A	A	A	I
4	A	A	A	A

of each of these multipliers and their respective arrangement for addition in second stage is depicted in Fig. 4. Note that this kind of arrangement will not lead to any change in latency of second pipeline stage as we perform the addition of $A_H X_L$ and $A_L X_H$ in parallel with rest of the smaller multipliers. The latter additions generate a net sum of $A_H X_H$ in almost the same time as the former takes to complete its addition. Now let us see how we can exploit this property in the proposed multiplication to configure its accuracy.

A. Accuracy Configuration Modes

Since now we have 7 multipliers in stage 1, we can vary the accuracy level of the proposed multiplier by varying the number of multipliers that are accurate. In any case, we keep the $A_{HH}X_{HH}$ as always accurate, so that the accuracy level does not fall below a certain level. Therefore, we obtain an accuracy configurable multiplier whose accuracy can be adjusted according to error tolerance of the application. The number of inaccurate multipliers used will directly determine the amount of power saved by the multiplier.

We propose 4 different modes of operations of our approximate multiplier based on accuracy levels. The proposed modes are given in Table I. Here 'A' stands for an accurate multiplier and 'I' stands for an inaccurate multiplier. We explain the bit-width aware algorithm next.

B. Proposed Bit-Width Aware Algorithm

We propose a bit-width aware algorithm for generalized $2b\times 2b$ multiplication which is configurable at run-time according to bit-width of operands *i.e.* if $b \times b$ multiplication is with smaller operands, say $(1011)_2 \times (1101)_2$, it will configure b at run-time as 4, not as 8 (used previously for $b \times b$). The size of inaccurate part (k) in the approximate partial products will always be equal to $b/2$ bits. Therefore, k will be automatically set to 2 and not as 4. Further, the positions at which middle $b/2$ bits are set to 1 also changes with operand bit-width. This has already been indicated in Fig. 2. This is plausible because at a time, we will use a multiplier of fixed size depending on application and hence can program it accordingly.

We present our algorithm (see Algorithm 1) of approximate $b \times b$ partial product computation (*e.g.* $A_H X_L$) in its most general form. This algorithm is coded later as a C-Program for simulation purposes. It should be noted that the index 0 is the Most Significant Bit in the Algorithm 1.

Algorithm 1 Approximate Partial Product Evaluation

procedure APPROXIMATE_PRODUCT(p, a, x)
  $Product \leftarrow p[0, 1, ..., 2b-1]$ /* Say $A_H X_L$ */
  $Multiplicand \leftarrow a[0, 1, ..., b-1]$
  $Multiplier \leftarrow x[0, 1, ..., b-1]$
  $c \leftarrow 0$ /* Temporary Carry */
  $q, r \leftarrow 2b-1$; $d \leftarrow b-(k/2)$
  for $i \leftarrow b-1, b-(k/2)$ **do** /* Inaccurate Part */
    for $j \leftarrow b-1, d$ **do**
      $[p(q), c] \leftarrow$ add-bits($p(q)$, $a(j)\&x(i)$, c);
      $q \leftarrow q-1$
    end for
    $q, r \leftarrow r-1$; $d \leftarrow d+1$; $c \leftarrow 0$
  end for
  for $r \leftarrow 2b-(k/2)-1, 2b-k$ **do**
    $p(r) \leftarrow 1$;
  end for
  $C_{in} \leftarrow$ Carry-in-Pre /* Switch Case for various b */
  $p(r) \leftarrow C_{in}$; $q, r \leftarrow 2b-k-1$; $d \leftarrow b-k-1$; $C_{in} \leftarrow 0$
  for $i \leftarrow b-1, b-k$ **do** /* Accurate Part */
    for $j \leftarrow d, 0$ **do**
      $[p(q), C_{in}] \leftarrow$ add-bits($p(q)$, $a(j)\&x(i)$, C_{in});
      if $j == 0$ and $C_{in} == 1$ and $q \neq 0$ **then**
        $p(q-1) \leftarrow 1$
      end if
      $q \leftarrow q-1$
    end for
    $q \leftarrow r$; $d \leftarrow d+1$; $C_{in} \leftarrow 0$
  end for
  for $i \leftarrow b-k-1, 0$ **do**
    for $j \leftarrow b-1, 0$ **do**
      $[p(q), C_{in}] \leftarrow$ add-bits($p(q)$, $a(j)\&x(i)$, C_{in});
      if $j == 0$ and $C_{in} == 1$ and $q \neq 0$ **then**
        $p(q-1) \leftarrow 1$
      end if
      $q \leftarrow q-1$
    end for
    $q, r \leftarrow r-1$; $C_{in} \leftarrow 0$
  end for
end procedure

VI. AWTM DESIGN AND ITS LPA OPTIMIZATION

In our accuracy-configurable design, we have reduced the horizontal critical path to just half. In order to reduce the

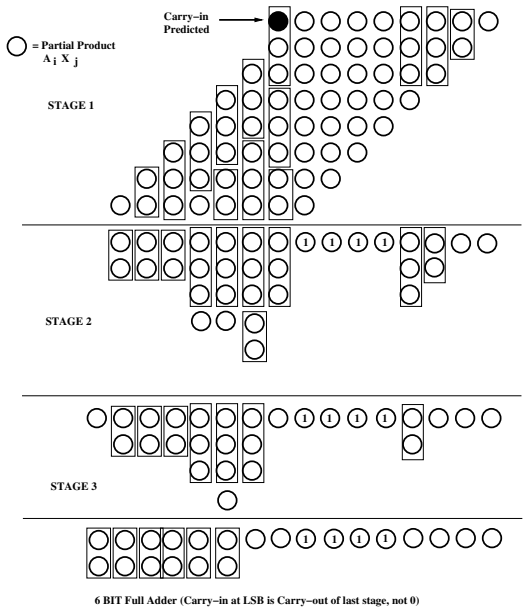

Fig. 5. Wallace Tree for approximate 8×8 partial product evaluation

vertical critical path as well (of $A_H X_L$, $A_L X_H$ and $A_L X_L$), we use Wallace Tree Reduction [15]. The Wallace Trees are fast and hardware efficient for multiplication of more than 16 bits. Wallace tree height also grows as $log_{3/2}(N/2)$. We call this Wallace tree based design as Approximate Wallace Tree Multiplier (AWTM). For an accurate 8×8 Wallace multiplication, it takes a total of 4 stages of reduction (each of which has a delay of 1 full adder) and then uses a 11-bit full adder to compute the final product. We use these 8×8 partial products to evaluate a 16×16 multiplication.

Fig. 5 shows that approximate partial product multipliers ($A_H X_L$, $A_L X_H$ and $A_L X_L$), which are 8×8, take a total of 3 stages of reduction and further use a 6-bit full adder for final product evaluation. When compared in terms of critical paths (of stage 1 of pipelined 16×16 multiplier), an accurate 8×8 multiplier uses a delay of 15 full adders (4 stages and 11-bit full adder) and its approximate counterpart uses a delay of 9 full adders (3 stages and a 6-bit full adder). Theoretically, this leads to an improvement of 40% in latency of stage 1.

Furthermore, for each of the $A_H X_L$, $A_L X_H$ and $A_L X_L$, number of adders reduced is around 51.24% (48 full adders and 25 half adders are required for their accurate evaluation which is in contrast with 23 full adders and 13 half adders required for approximate evaluation) and hence for complete pipelined multiplier, total power reduction is expected to be around 38.42% (because $A_H X_H$ is accurate and other 3 are inaccurate, therefore three-fourth of 51.24%). We validate these theoretical estimates by running actual simulations. The experimental results and their analysis are discussed next.

VII. Experimental Results and Analysis

In this section, we present power and area results obtained experimentally. All results have been produced using Cadence RTL Compiler for $45nm$ Nangate Opencell Library. We also generate results on real time application by computing Discrete Cosine Transform (DCT) and Inverse Discrete Cosine Transform (iDCT) of a benchmark image.

TABLE II
Results: Mean Error and Acceptance Probability (AP)

Mode	Parameter	For Operands > 1 (in %)	For Operands > 1000 (in %)
1	Mean Error	5.26	4.59
	AP	27.34	28.18
2	Mean Error	3.42	3.16
	AP	46.18	46.04
3	Mean Error	0.46	0.29
	AP	91.58	94.28
4	Mean Error	0.13	0.035
	AP	98.44	99.72

A. Accuracy and Acceptance Probability

We simulate our 16×16 bit-width aware multiplier using a C Program by generating 5000 random numbers to compute accurate and approximate products for all possible combinations for different accuracy modes. For each case, MAA is set as 99%. It generates results like ACC_{amp}, mean error and Acceptance Probability. The mean error and Acceptance Probability (AP) results are tabulated in Table II for various modes. Note that for unbounded operand size (operands > 1 in Table II), bit-width aware algorithm is employed. Whereas when operand size is constrained to be 10 bit or more (operands > 1000), it is found that a simple 16×16 approximate multiplier without bit-width awareness produces almost the same results.

Table II shows spectacular results for accuracy levels. Acceptance Probability of more than 98% for a minimum acceptable accuracy of 99% signifies that for all possible combinations of the random numbers generated, more than 98% cases give an accuracy greater than 99%. Also, as explained earlier, for larger numbers (operand size > 1000), the accuracy level shoots up to as high as 99.965%. A 16×16 multiplier proposed in [12] generates a mean error of 3.32%. Clearly, our multiplier performs better than this for modes 3 and 4. The error is comparable for mode 2.

We further investigate the relationship between MAA and acceptance probability. Fig. 6 shows a plot of AP *vs.* MAA for various modes and for ETM (proposed in [13]). It is evident that our multiplier (when used in mode 3 and 4) outperforms ETM easily as far as accuracy is concerned. The accuracy results for mode 4 of 16×16 AWTM were confirmed by inputting $10,000$ random test vectors in the RTL netlist. Note that for the sake of simplicity, we do not make hardware implementation (HDL codes) of AWTM as bit-width aware. For such a design, we obtained a mean error of 0.16% and AP of 98.56% for MAA of 99%, quite in agreement with Table II. Hence, mode 4 of 16×16 multiplier gives almost the same results (for all operands) when employed with or without bit-width awareness. Next, we present Power and Area results.

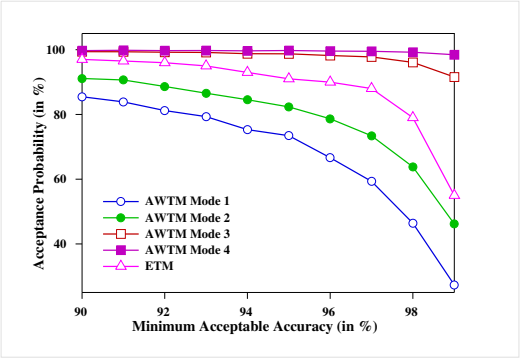

Fig. 6. Acceptance Probability *vs.* Minimum Acceptable Accuracy

TABLE III
REDUCTION IN AREA AND POWER (HIGHER IS BETTER)

Approximate Multiplier		Area (%)	Power (%)
4×4	AWTM (Proposed)	55.76	53.16
	ETM [13]	53.85	49.60
	Kulkarni [12]	35.75	36.3
	Truncation [11]	43.44	43.08
8×8	AWTM (Proposed)	51.93	57.19
	ETM [13]	50.02	39.25
	Kulkarni [12]	22.03	41.5
	Truncation [11]	47.90	15.30
16×16	AWTM (Proposed)	34.49	41.96
	ETM [13]	30.27	31.49
	Kulkarni [12]	17.89	31.8
	Truncation [11]	15.17	9.69

B. Power and Area Analysis

We obtain power and area results for our 4×4, 8×8 and 16×16 AWTM designs. Table III shows these results along with power and area results for the comparable designs reported in the literature. Here, power and area reduction of various approximate multipliers are computed with respect to their accurate counterparts. Fig. 7 and Fig. 8 display the scalability results from which it is evident that AWTM performs better than all other corresponding multipliers reported in the literature *w.r.t* power and area.

Since, we have not optimized the second stage of pipeline (*i.e.* the addition stage), the latency results were same for both accurate and approximate 16×16 pipelined multipliers. As Wallace trees are very fast, the minimum clock period in RTL Synthesis was decided by the addition stage itself. Nevertheless, when latency of single cycle implementation of AWTM (complete multiplier, not just stage 1) was compared with that of single cycle accurate multiplier, it was found that AWTM reduces the latency by 23.91%. Similarly for 8×8 multiplier, we achieved 32% reduction in latency, which was theoretically predicted to be around 40%.

A net reduction in total power and area of 57.19% and 51.93% respectively is also obtained for 8×8 AWTM. Both of these values were expected to be around 52% theoretically as mentioned in the previous section. Therefore, the experimental results confirm to the theoretical results to a large extent. The area and power reduction results of 16×16 AWTM are given in Table IV for various modes of operation.

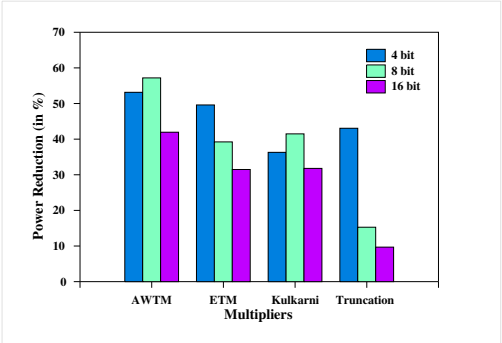

Fig. 7. Power Reduction plot for Scalability of approximate multipliers (Higher is better)

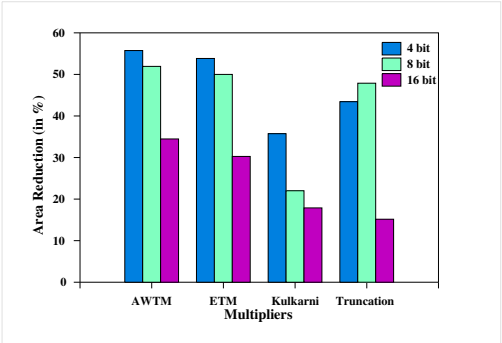

Fig. 8. Area Reduction for various scalable approximate multipliers (Higher is better)

TABLE IV
RTL COMPILER RESULTS OF 16×16 AWTM

Mode	Area (in %)	Leakage Power(in %)	Dynamic Power(in %)	Total Power (in %)
1	34.49	34.06	43.4	41.96
2	31.91	32.52	41.46	40.63
3	29.89	30.97	39.68	38.86
4	27.90	29.43	37.90	37.10

Key Observations: First as expected, leakage power and area follow almost the same trends (even in the values of percentage reduction). Second, the percentage power reduction goes as high as 41.96%. Therefore, we have larger power savings for applications that can tolerate relatively more error. Finally, for mode 4, net power saving obtained from RTL Compiler is 37.10% which was estimated to be around 38% theoretically, thus confirming the validity of these results.

Furthermore, Fig. 9 compares percentage reduction in power and area against the mean error involved in 16×16 product. The figure clearly shows that with increase in error tolerance of application, power (dynamic as well as leakage) and area savings also increase. In the next subsection, we evaluate our multiplier on a real time application.

C. Real Time Application: DCT and iDCT

We make use of the AWTM to demonstrate its effectiveness in computing DCT and iDCT of benchmark image 'Lena' We

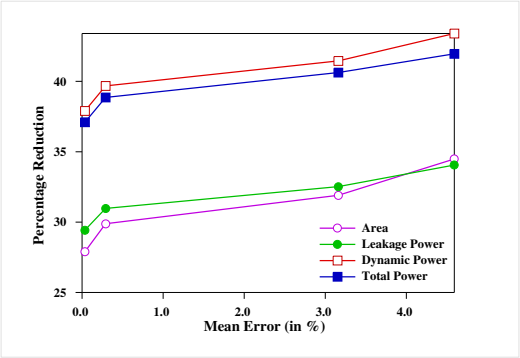

Fig. 9. Mean Error *vs.* Area and Power Reduction

Fig. 10. DCT and iDCT of image using accurate multiplier and AWTM

have used this application because it involves multiplication of floating point numbers. Floating point multiplication uses large unsigned multipliers, making them an ideal application area of AWTM. Fig. 10 shows our results on this image. The same image is generated back when the DCT and iDCT operations are performed on it.

As expected, the results for mode 1 (Fig. 10(e)) and mode 2 (Fig. 10(f)) are not good as they give a mean error of $5-6\%$. It means that this application can't tolerate such a magnitude of error. Obviously there are applications which can do, and hence mode 1 and 2 can be easily employed there. On the other hand, it is hardly possible to distinguish between results of accurate multiplier (Fig. 10(b)) and those of AWTM mode 4 (Fig. 10(c)) and mode 3 (Fig. 10(d)). The multiplier used to produce image in Fig. 10(c) saves 37% power and 28% area. The one used for Fig. 10(d) reduces power by 38.86% and area by 29.89%. Therefore, using AWTM, we can save up to 39% power and 30% area with negligible loss in image quality.

VIII. Conclusion

We proposed a power and area-efficient Approximate Wallace Tree Multiplier (AWTM) for error-resilient systems. A new bit-width aware approximate multiplication algorithm is also presented. The AWTM design is further empowered with a Carry-in Prediction logic and its efficient precomputation to increase overall throughput. Our power-area efficient AWTM is fast, particularly for operands size of 16-bit or more, and optimized *w.r.t.* LPA design metrics. Single cycle implementation of AWTM showed a 23.91% reduction in latency. We obtained the mean accuracy of 99.85% to 99.965% for 16-bit multiplication of different sized operands. We also achieved significant reduction in power and area of our multiplier design, up to 41.96% and 34.49% respectively for 16-bit multiplication which clearly demonstrates efficiency and effectiveness of AWTM. Finally, we demonstrated that AWTM produced images of almost the same quality as obtained by operations using accurate multipliers but with power and area savings of around 39% and 30% respectively.

References

[1] "International technology roadmap for semiconductors, http://www.itrs.net."

[2] E. J. Swartzlander, "Truncated multiplication with approximate rounding," in *Signals, Systems, and Computers, 1999. Conference Record of the Thirty-Third Asilomar Conference on*, vol. 2, oct. 1999, pp. 1480–1483 vol.2.

[3] L. N. Chakrapani, K. K. Muntimadugu, L. Avinash, J. George, and K. V.Palem, "Highly energy and performance efficient embedded computing through approximately correct arithmetic: a mathematical foundation and preliminary experimental validation," *CASES*, pp. 187–196, 2008.

[4] N. Zhu, W. L. Goh, W. Zhang, K. S. Yeo, and Z. H. Kong, "Design of low-power high-speed truncation-error-tolerant adder and its application in digital signal processing," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 18, no. 8, pp. 1225–1229, aug. 2010.

[5] N. Zhu, W. L. Goh, G. Wang, and K. S. Yeo, "Enhanced low-power high-speed adder for error-tolerant application," in *SoC Design Conference (ISOCC), 2010 International*, nov. 2010, pp. 323–327.

[6] A. Verma, P. Brisk, and P. Ienne, "Variable latency speculative addition: A new paradigm for arithmetic circuit design," in *Design, Automation and Test in Europe, 2008. DATE '08*, march 2008, pp. 1250–1255.

[7] A. Kahng and S. Kang, "Accuracy-configurable adder for approximate arithmetic designs," in *Design Automation Conference (DAC), 2012 49th ACM/EDAC/IEEE*, june 2012, pp. 820–825.

[8] S. L. Lu, "Speeding up processing with approximation circuits," *Computer*, vol. 37, no. 3, pp. 67–73, mar 2004.

[9] D. Shin and S. Gupta, "A re-design technique for datapath modules in error tolerant applications," in *Asian Test Symposium, 2008. ATS '08. 17th*, nov. 2008, pp. 431–437.

[10] V. Gupta, D. Mohapatra, A. Raghunathan, and K. Roy, "Low-power digital signal processing using approximate adders," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 32, no. 1, pp. 124–137, jan. 2013.

[11] M. B. Sullivan and E. E. Swartzlander, "Truncated error correction for flexible approximate multiplication," in *Signals, Systems and Computers (ASILOMAR), 2012 Conference Record of the Forty Sixth Asilomar Conference on*, 2012, pp. 355–359.

[12] P. Kulkarni, P. Gupta, and M. Ercegovac, "Trading accuracy for power with an underdesigned multiplier architecture," in *VLSI Design (VLSI Design), 2011 24th International Conference on*, 2011, pp. 346–351.

[13] K. Y. Kyaw, W.-L. Goh, and K.-S. Yeo, "Low-power high-speed multiplier for error-tolerant application," in *Electron Devices and Solid-State Circuits (EDSSC), 2010 IEEE International Conference of*, 2010, pp. 1–4.

[14] K. Bhardwaj and P. S. Mane, "Acma: Accuracy-configurable multiplier architecture for error-resilient system-on-chip," in *Reconfigurable and Communication-Centric Systems-on-Chip (ReCoSoC), 2013 8th International Workshop on*, July 2013, pp. 1–6.

[15] C. S. Wallace, "A suggestion for a fast multiplier," in *Electronic Computers, 1964 IEEE Transactions on*, 1964, pp. 14–17.