MACACO: Modeling and Analysis of Circuits for Approximate Computing

Rangharajan Venkatesan, Amit Agarwal, Kaushik Roy and Anand Raghunathan
School of Electrical and Computer Engineering, Purdue University
{rvenkate,agarwa19,kaushik,raghunathan}@purdue.edu

***Abstract*—Approximate computing, which refers to a class of techniques that relax the requirement of exact equivalence between the specification and implementation of a computing system, has attracted significant interest in recent years. We propose a systematic methodology, called MACACO, for the Modeling and Analysis of Circuits for Approximate Computing. The proposed methodology can be utilized to analyze how an approximate circuit behaves with reference to a conventional correct implementation, by computing metrics such as worst-case error, average-case error, error probability, and error distribution. The methodology applies to both timing-induced approximations such as voltage over-scaling or over-clocking, and functional approximations based on logic complexity reduction. The first step in MACACO is the construction of an equivalent untimed circuit that represents the behavior of the approximate circuit at a given voltage and clock period. Next, we construct a virtual error circuit that represents the error in the approximate circuit's output for any given input or input sequence. Finally, we apply conventional Boolean analysis techniques (SAT solvers, BDDs) and statistical techniques (Monte-Carlo simulation) in order to compute the various metrics of interest. We have applied the proposed methodology to analyze a range of approximate designs for datapath building blocks. Our results show that MACACO can help a designer to systematically evaluate the impact of approximate circuits, and to choose between different approximate implementations, thereby facilitating the adoption of such circuits for approximate computing.**

I. Introduction

Recent years have witnessed a surge of interest in a group of techniques that could be collectively classified as approximate computing, wherein the requirement of exact numerical or Boolean equivalence between the specification and implementation of a computing platform is relaxed in order to achieve improvements in performance or energy efficiency [1] [2] [3]. Approximate computing is motivated by the large and growing class of applications that demonstrate inherent error resilience, such as DSP, multimedia (images/audio/video), graphics, wireless communications, and emerging workloads such as recognition, mining, and synthesis. These applications usually process large, redundant data sets that contain significant noise, by utilizing statistical or probabilistic computations. The requirement of numerical exactness on their outputs is relaxed due to several factors: (i) the limited perceptual capability of humans (*e.g.*, audio, video, graphics), (ii) a golden result is difficult to define or does not exist (*e.g.*, web search, data analytics), or (iii) users are willing to accept less-than-perfect results.

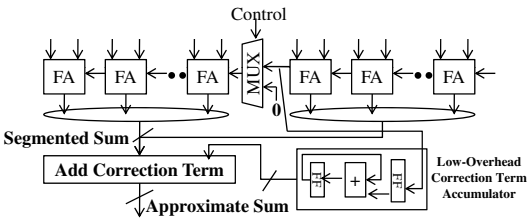
Approximate computing in hardware is based on designs of hardware building blocks whose implementation does not exactly match the specification, either due to the impact of timing-induced errors (*e.g.*, voltage over-scaling or over-clocking), or due to functional approximation (*e.g.*, implement a slightly different Boolean function that has a faster or more power-efficient implementation). Although approximate computing techniques have shown significant promise, moving them to the mainstream will require several issues to be addressed, foremost among which is the issue of modeling and analysis of accuracy. Several approximate designs have been proposed that compromise accuracy in different ways; unfortunately there is no simple and systematic analysis methodology to compare them with conventional designs or with each other. For example, a designer may ask the following questions: "How do I compare between different approximate designs for my circuit?", "How do I ensure that a given approximate implementation meets my accuracy requirements?", or "How do I ensure that there are no bugs in an approximate implementation?". Traditional verification techniques (combinational and sequential equivalence checking) are geared towards verifying exact Boolean equivalence of the specification and implementation, and do not directly address the above questions.

We take a first step towards addressing the aforementioned challenge, by proposing a systematic methodology for the analysis of approximate circuits, and applying it to analyze several approximate implementations of data path building blocks. We start by noting a couple of requirements for such an analysis framework.

- Approximate circuits may be evaluated using multiple metrics. For example, some designers may wish to evaluate the worst-case error, *i.e.*, the largest possible difference between the approximate output and a correct version for all possible inputs. In other scenarios, the error probability, *i.e.*, the probability that the output differs from the correct one, may be of relevance. Finally, the distribution of the errors (for all possible inputs) may be of interest in some cases. Therefore, the analysis framework should be flexible enough to support multiple analysis modes (worst-case error, average-case error, error distribution, *etc.*).
- Several approximate circuit designs utilize voltage over-scaling or over-clocking, resulting in timing errors. Therefore, we need to consider timing information in our analysis framework, *i.e.*, we cannot directly apply techniques from the functional verification domain. Timing-faithful circuit simulation is not an option in many cases since exhaustive enumeration of all inputs or input pairs may be required.

We propose MACACO, a flexible and efficient methodology for the analysis of approximate circuits that meets both these requirements. First, we convert timing-induced approximations into the functional domain. Given a circuit that is subject to over-clocking or voltage over-scaling at a specific clock frequency and supply voltage, we generate an *equivalent untimed circuit* that represents the functional behavior of the over-scaled or over-clocked circuit. Next, we construct a virtual circuit for error analysis by instantiating the equivalent untimed circuit and the golden circuit, and comparing their outputs to quantify the error. This *virtual error circuit* computes the error between the approximate and correct implementations for any given sequence of inputs. Our methodology then utilizes conventional verification tools — Boolean satisfiability (SAT) solvers and Binary Decision Diagrams (BDDs) — to perform error analysis. To compute the worst-case error, we utilize a SAT solver to maximize the output of the virtual error circuit. To compute the average-case error, error probability or error distribution, we construct a BDD for the output of

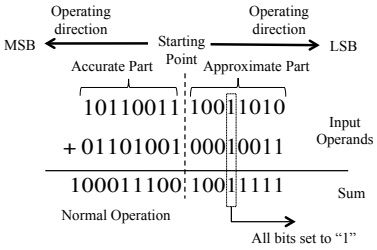
978-1-4577-1400-9/11/$26.00 ©2011 IEEE



(a) Dynamic Segmentation with Error Compensation

A	B	Cin	Sum	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	0	1
0	1	1	0	1
1	0	0	0	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

(b) Truth Table based approximation



(c) Reverse Carry Propagation

Fig. 1: Approximate circuit design techniques, illustrated using adders

the virtual error circuit and utilize a traversal of the BDD to compute the relevant statistics. For cases where BDDs cannot be constructed (*e.g.*, larger bitwidth multipliers), we utilize statistical Monte-Carlo sampling to estimate these metrics. The proposed approach may also be used with other decision diagrams [4]–[7] that are better suited to dealing with pathological cases for BDDs.

We apply the proposed analysis methodology to a range of implementations of common arithmetic units. In the case of adders, which are the most fundamental building block, we evaluate the behavior of three different conventional architectures under voltage over-scaling - ripple carry adder, carry look-ahead adder, and Han-Carlson adder. We also evaluate three architectures that are specifically designed for approximate computing, based on dynamic segmentation with error compensation, reverse carry propagation, and truth table based approximation. Our results reveal interesting insights into the behavior of these approximate arithmetic units and the trade-offs between them. For example, our analysis identifies that functional correlation between the output bits reduces the maximum error introduced by over-scaling, explaining the empirical observation that the average-case error in practice is much smaller than the worst-case. For some designs, we also observe that voltage over-scaling introduces errors of certain specific values much more frequently than others, leading to the possibility of low-complexity error detection and correction. We believe that systematic analysis frameworks such as the one proposed in this paper can significantly facilitate the adoption of approximate computing techniques.

The rest of this paper is organized as follows. In Section II, we present a brief overview of research on approximate computing. Section III focuses on the design of approximate arithmetic units, classifying the techniques into timing-based and functional approximation. Section IV presents the MACACO methodology. Section V evaluates a range of approximate arithmetic circuits using MACACO, and Section VI concludes the paper.

II. RELATED WORK

A number of previous research efforts have demonstrated the benefits of approximate computing through techniques at various levels of design abstraction [8]–[22]. The flexibility to compute an approximate result is translated into improved efficiency (performance or power) in hardware using two common techniques: (i) timing-induced approximation, wherein a circuit is subject to voltage over-scaling or over-clocking, or (ii) functional approximation, wherein the Boolean function is slightly altered to result in a simplified implementation. These techniques have been applied to design approximate versions of various datapath building blocks [13]–[18]. Circuit synthesis techniques [19], [20] based on sizing and multi-V_t libraries make circuits behave more gracefully under voltage over-scaling. More recent work has focused on the systematic synthesis of approximate circuits at the logic level [21], [22].

In summary, approximate circuits sacrifice 100% correctness in return for disproportionate benefits in performance or power consumption. These circuits are often evaluated for limited sets of inputs, or in the limited context of a specific application. The primary contribution of this work is a systematic methodology for the modeling and analysis of approximate circuits. The proposed technique can be used to analyze circuits resulting from any of the design techniques described above. The proposed methodology can also benefit techniques that utilize circuit-level error detection and correction to recover from errors due to elimination of guard bands or voltage over-scaling [23]. In this case, the proposed methodology would compute the error probability that is required to analyze the performance overheads of error correction.

III. APPROXIMATE CIRCUITS: PRELIMINARIES

Design techniques for approximate circuits follow two broad strategies - Over-scaling based approximation (over-clocking or voltage over-scaling, resulting in timing induced errors), and Functional approximation. In over-scaling based approximation, the circuits are designed to operate correctly under normal conditions. Approximations are introduced by voltage over-scaling (without changing the clock frequency) to gain benefits in terms of energy. Equivalently, frequency can be increased (without changing the voltage) for performance gain. Depending on the architecture, the number of paths that fail to meet the delay constraint varies and different circuits behave differently under over-scaled conditions. For instance, in the case of a ripple carry adder, there are fewer long paths and the output fails gradually with over-scaling [9]. On the other hand, a tree adder such as the Han-Carlson adder has a large number of critical paths and fails drastically. In addition, there are circuits specifically designed for better scalability [13]–[18].

Fig. 1 provides illustrative examples of approximate adders. Fig. 1a demonstrates the operation of the Dynamic segmentation with Error Compensation (DSEC) technique, which applies to accumulators such as the ones used in Multiply-Accumulate (MAC) units [16]. Dynamic segmentation involves dividing the accumulator stage into smaller bit width adders depending on the degree of over-scaling. Error compensation keeps track of the carries across sections of the segmented adder that have been ignored in each cycle, and a correction cycle is introduced to adjust the accumulator value in order to compensate for the error due to ignored carries.

In the case of functional approximation, circuits are designed to be error prone even in the absence of voltage over-scaling. The principle is to accept errors in rare cases for significant reductions in the logic complexity or length of the critical path. Fig. 1b shows the truth table of a functional approximation for a full adder [17]. Changing the truth table at the indicated positions allows for a

simplified implementation with lower power consumption, lower area and higher performance at the cost of approximation. Another kind of functional approximation is reverse carry propagation [13]. As shown in Fig. 1c, approximations are introduced only in the LSBs. The adder is partitioned in such a way that the LSBs fail initially with over-scaling which ensures that error introduced is of small magnitude.

IV. MACACO: Modeling and Analysis of Approximate Circuits

An analysis framework for approximate circuits should provide information regarding

- Worst-case error over all possible inputs.
- Probability of occurrence of errors.
- Error value distribution.
- Conditions for occurrence of undesirable errors *i.e.*, errors of large magnitude or errors with high probability.

In previous work, such information is obtained through simulation on a limited set of vectors or in the context of a specific application. These ad hoc approaches cannot be used to compute worst-case error, which requires exhaustive enumeration, and are not scalable since the number of vectors to be simulated becomes too large. It is desirable to leverage Boolean analysis techniques, which are very efficient at implicit exhaustive enumeration, to the problem of approximate circuit analysis. However, this task is complicated by the fact that formal verification methods are not directly applicable when the timing behavior of the circuit (delays of the gates) needs to be considered. We address this challenge in the proposed methodology.

Fig. 2 presents an overview of the MACACO methodology for the analysis of approximate circuits. The proposed methodology consists of three key steps:

- **Equivalent untimed circuit generation:** We first transform the timing-dependent nature of the approximate circuit into the functional domain, so that Boolean analysis techniques may be used. The Equivalent Untimed ciRcuit gEnerAtor (EUREkA) takes the approximate circuit, the target clock frequency and the delay model of the cell library, and generates a functional equivalent of the original circuit for the given voltage and clock period. This step is described in detail in Section IV-A.
- **Virtual error circuit generation:** In this step, we construct a virtual error circuit that models the error of the approximate circuit, by comparing the output of the equivalent untimed circuit from the previous step with the output of a fully correct reference circuit. This step is further described in Section IV-B.
- **Error analysis:** In this step, we apply conventional verification tools (SAT, BDDs) as well as statistical Monte-Carlo simulation, to perform error analysis using the virtual error circuit. This step is further described in Section IV-B.

A. Modeling Approximate Circuits using Equivalent Untimed Circuits

In this section, we first describe the impact of timing-induced approximation on a circuit's output. We then describe how this effect may be captured using equivalent untimed circuit, describe our algorithm for the generation of equivalent untimed circuit, and demonstrate it using a suitable example.

Under timing-induced approximation, the output of a combinational circuit becomes a function of the current as well as previous input values. The current inputs would propagate to the output through non-critical paths (paths that are shorter than the clock period), while the previous inputs would affect the output through

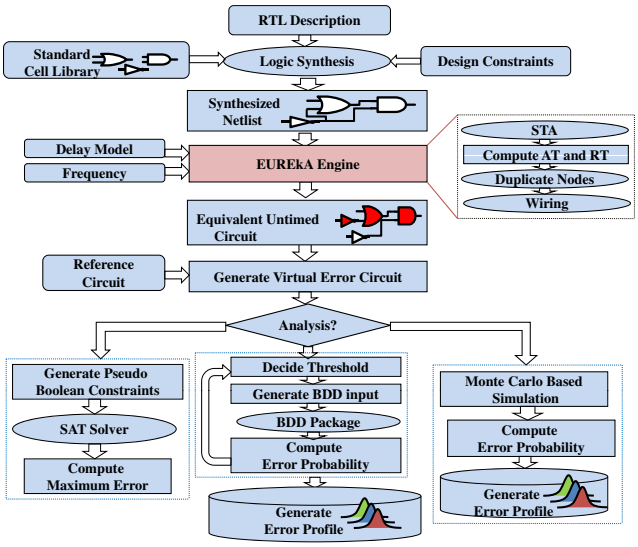


Fig. 2: Overview of MACACO

critical paths (paths that are longer than the clock period). For example, consider the circuit shown in Fig. 3a. The delays of the individual gates (normalized with respect to an inverter's delay) are shown inside the gates. The arrival time for the output of each gate is shown inside a circle. The maximum delay of the circuit is 10. If the clock period is 8, Path 1 and Path 2 are not critical and the values of inputs *IN0* and *IN1* from the current cycle can propagate to the output. On the other hand, the current cycle values of inputs *IN3* and *IN4* do not have sufficient time to propagate to the output. Therefore, the previous cycle's values of *IN3* and *IN4* would influence the current output. Also, note that inputs *IN5* and *IN6* can propagate to the output through Path 7 but cannot propagate through Path 6. In this case, both the current and previous values of the inputs would affect the current output. In general, if I_t represents the current inputs to the circuit, and D represents the net delay of the circuit, and T represents the clock period, then the outputs $Out_{(t+T)}$ depend on inputs $I_t, I_{(t-T)}, ..., I_{(t-rT)}$ where $r = \lceil (D/T) \rceil - 1$.

Let us now generalize the discussion from the above example. Consider a circuit C with only one critical path P having delay D that is greater than clock period T. Assume that every gate along P has a fanout of exactly one. Clearly, the current primary input values cannot propagate to the output through path P. The output would be determined by the previous primary input value that had propagated through path P and the current primary input values that propagate to the output through other non-critical paths. Therefore, by substituting the current primary input value with the previous primary input value along path P and retaining the current primary input values along other paths, we obtain a circuit C' whose output is equal to the output of circuit C under scaled conditions. We refer to C' as the equivalent untimed circuit of C, since the untimed simulation of circuit C' would be equivalent to timed simulation of the original circuit under the specified clock period T. Next, consider the case when the gates along path P in circuit C have fanout greater than one. In this case, there are nodes (primary inputs or output of a gate) along path P such that the value of the node corresponding to the current input values can propagate to the output through certain paths but cannot propagate through path P. In this case, we duplicate the gates with fanout greater than one such that every gate along P has a fanout of one [24] and then obtain the equivalent untimed circuit as described earlier.

Algorithm 1 EUREkA Engine

Input: C: Synthesized Netlist
T: Clock Period
Library with Delay models
Output: C': Equivalent untimed Circuit
Begin
$Parse\_circuit()$;
$Topological\_sort()$;
$STA()$; {Compute arrival and required Times}
for each $out_i \in$ outputs such that $AT_i > T$ **do**
duplicate out_i as out_i';
$DUPLICATE\_DFS(out_i, 0, out_i^d)$;
end for
End

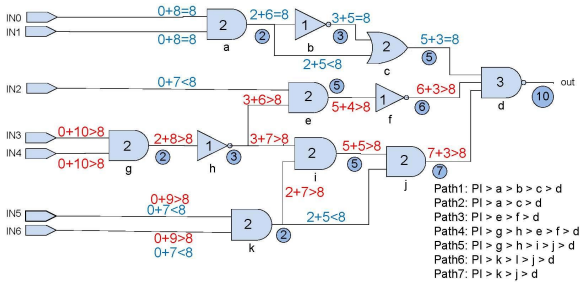
DUPLICATE_DFS(N_l, D_{il}, P_l) {N_l = Node in original circuit}
for each fanin N_j of N_l **do** {P_l = reproduced or duplicated Node of N_l}
$D_{ij} \leftarrow D_{il}$ + gate_delay(N_l) {D_{il} = Depth of N_l from out_i}
if ($AT_i + D_{ij} > T$) **then** {condition for node to be critical}
if ($markd_j$ is set) **then** {checking if the node is already duplicated}
Set P_l as fanout of N_j^d
Return;
else
Duplicate N_j as N_j^d;
Rename the duplicated node;
Set P_l as fanout of N_j^d
Set $markd_j$;
$DUPLICATE\_DFS(N_j, D_{ij}, N_j^d)$
end if
else
if ($markc_j$ is set) **then** {checking if the node is already reproduced}
Set P_l as fanout of N_j^r
Return;
else
Reproduce N_j as N_j^r;
Set P_l as fanout of N_j^r
Set $markc_j$;
$DUPLICATE\_DFS(N_j, D_{ij}, N_j^r)$
end if
end if
end for

Let us now consider a circuit C with multiple critical paths. Assume that $D < 2T$, implying that output $Out_{(t+T)}$ of the approximate circuit would depend only on I_t, $I_{(t-T)}$. We can divide the paths in the circuit into critical paths whose delay is greater than T and non-critical paths whose delay is less than T. In this case, we duplicate the gates which are part of a critical path, but also have a fanout to a gate in a non-critical path, to obtain a circuit C_d. This ensures that the value of any node in the circuit C_d can either propagate through all the paths to the output or cannot propagate through any path connecting the node to the output. By applying previous input values to the primary inputs in the critical paths and current input values to the primary inputs along the non-critical paths, we obtain the required equivalent untimed circuit.

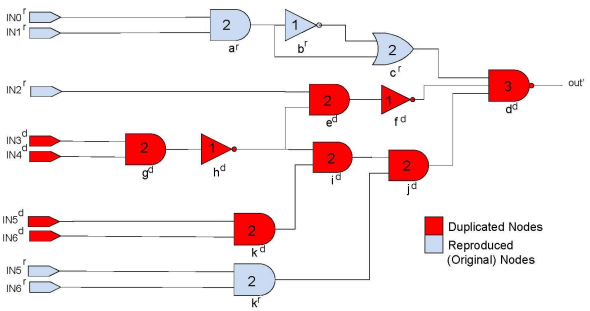
In the general case when $(r-1)T < D < rT$ for some integer r, gates need to be duplicated such that the gates that are part of a path with delay d such that $(i-1)T < D < iT$ where $2 < i < r$, does not fanout to a gate that is part of a path with delay d' such that $(j-1)T < d' < jT$ where $0 < j < i$.

Algorithm 1 describes the procedure for generating the equivalent untimed circuit. This algorithm was inspired from the algorithm presented in [24] for redundancy removal. For ease of explanation, we assume that $D < 2T$ (the algorithm directly extends to general case). In order to generate the equivalent untimed circuit, we employ a modified Depth First Search (DFS) to find and duplicate the portion of the circuit which violates the time constraint and then connect it to the original circuit for further analysis. In our algorithm, each node of a circuit can be visited through a critical path or a non-critical path. Hence we associate two marker variables, *markc* and *markd*, with each node. Each node is visited at most twice in the entire process *i.e.*, our algorithm implicitly considers all paths in the circuit without enumerating them. A node is duplicated and *markd* is set if it is visited for the first time through a critical path. If a node is visited through a non-critical path for the first time, the node is simply *reproduced* and *markc* is set. If a marked node is visited again through a critical (non-critical) path, then if *markd* (*markc*) is set, the node is not duplicated (reproduced) and the parent node (fanout of the current node through the current path) is linked to the duplicated (reproduced) node.

In general, when $(r-1)T < D < rT$ for some $r > 1$, we would require r *markd* variables, 1 *markc* variable and each node would be visited at most $r+1$ times. The equivalent untimed circuit obtained using our algorithm would have utmost $r * n$ nodes leading to space and run time complexity of *O(n+w)* , where n is the number of gates and inputs in the circuit, and w is the number of wires in the circuit.



(a) Approximate circuit



(b) Equivalent untimed circuit

Fig. 3: Generation of the equivalent untimed circuit

Let us consider the generation of the equivalent untimed circuit for the circuit shown in Fig. 3a. Let AT_l represent the arrival time of a node N_l. Depth of a node along a particular path is defined as the time required for the value at a node to propagate to the output. D_{il} represents the depth of N_l from the output, out_i, under consideration. Along a path, a node N_l is said to be critical if $AT_l + D_{il} > T$. It may be noted that reproducing a node simply regenerates the original circuit. Only the critical nodes are duplicated. Thus, if there is no critical path, all the nodes will be marked as *markc* and the original circuit will be reproduced. Fig. 3b shows the equivalent untimed circuit generated using our algorithm. We start from the output node *out*. This node satisfies the constraint for duplication. Then we visit the nodes along Path 1. After reaching the input *IN0*, the algorithm returns to node *a* and visits *IN1*, which is also reproduced. When the algorithm returns to node *c* from node *b*, only a link is created between node *a* and *c*, without revisiting the nodes as the *markc* label is already set. Our algorithm continues to visit all the nodes in depth-first manner. Consider the point when the algorithm reaches node *IN2* from node *e*. All the nodes from *e* to *out* are duplicated but

IN2 is reproduced which means that the current cycle's value of input *IN2* will affect the output. Next, consider the point when we reach node *i*. The algorithm then visits node *k* and inputs *IN5* and *IN6*, which are duplicated. Then, the algorithm returns to node *j* and then revisits node *k*, *IN5* and *IN6*, but they are reproduced this time. In this way, we obtain the equivalent untimed circuit efficiently without enumerating the paths in the circuit.

Given a circuit C with maximum delay D operating at clock period T ($T < D$), the output becomes a function of the current and previous input values. Our algorithm generates an equivalent untimed circuit that captures the functional relationship between the current and previous input values such that the output of untimed simulation of the equivalent untimed circuit is equal to that observed from timed simulation of the original circuit C.

B. Analysis of approximate circuit behavior

In this section, we describe how the untimed equivalent circuit is analyzed to compute the various metrics of interest, such as maximum error, error distribution, error probability, *etc*. We propose three different techniques to analyze an approximate circuit - a SAT Solver based approach to compute the maximum or worst-case error, a BDD based approach to compute the complete error distribution or error probability, and a statistical approach based on Monte-Carlo sampling to enable scalability. We note that other Boolean analysis techniques (*e.g.*, different decision diagrams) or statistical techniques (*e.g.*, stratified sampling) may also be employed within our framework.

1) SAT based analysis: For computing the worst-case error, we use a pseudo-Boolean SAT solver, which is basically a combination of a SAT solver and an ILP solver. We use part of the circuit shown in Fig. 4 and generate CNF formula corresponding to the reference circuit, the equivalent untimed circuit and the subtractor and an objective function that maximizes the error. The input to the SAT solver is given by:

$$\begin{aligned} Objective: &\quad Maximize\ Error \\ Constraints: &\quad CNF_{ref.} \wedge CNF_{approx.} \wedge CNF_{subt.} \end{aligned} \tag{1}$$

Note that the construction of the objective function should take into account the number representation scheme that is used in the circuit (sign-magnitude, 2's complement, *etc.*).

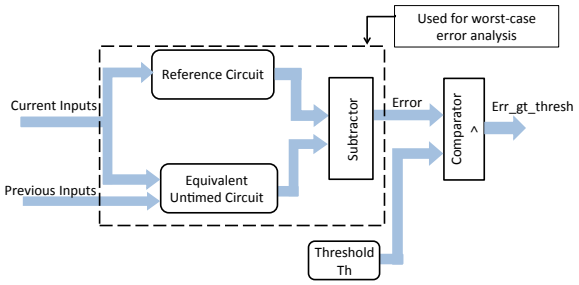


Fig. 4: Virtual error circuit

2) BDD based analysis: In most applications, knowledge of the worst-case error alone will not suffice, since its probability of occurrence could be negligible. For such a scenario, we propose a BDD based technique to generate an error distribution that would provide information about probability of occurrence of errors of different magnitudes. To compute the error distribution efficiently, we use the virtual error circuit shown in Figure 4, which checks if the error introduced by approximation is greater than a given threshold. The BDD representation is obtained for the output of the virtual error circuit after fixing the threshold to a constant value. We, then, traverse the BDD and count the number of paths from the root to the leaf 1 and obtain the number of input vectors that satisfy the comparator output. In other words, we obtain the number of input vectors corresponding to error of magnitude less than a chosen threshold. By repeating this procedure for different thresholds, we obtain the cumulative distribution function (CDF) of the error. The derivative of the CDF gives us the Probability Distribution Function (PDF). BDDs are known to be very large for pathological classes of circuits. There has been considerable research effort like BMDs [4], PBDDs [5], ZBDDs [6], and OKFDDs [7] that are efficient at representing many circuits for which BDDs are not feasible. They can also be used within our framework to obtain the error distribution of approximate circuits.

3) Monte-Carlo analysis: In our Monte-Carlo analysis, we perform RTL simulations to compute the error profile of the given circuit at different frequencies of operation. For a given frequency, we choose different thresholds for error to perform our analysis. Since there are exponentially many test vectors, we used statistical techniques to compute the number of vectors required for a given confidence level and margin of error using the following equation:

$$n = \frac{[(z^2 * p * q) + ME^2]}{ME^2 + \frac{(z^2 * p * q)}{N}} \tag{2}$$

where n is the number of samples required, z is the critical standard score computed based on the confidence level, p is the population proportion which depends on the threshold, $q = 1 - p$, ME is the margin of error, and N is the population size given by $N = 2^{2I}$ where I is the number of inputs to the circuit. For the number of samples computed above, Monte-Carlo simulations are performed for different thresholds to obtain a error distribution for a given operating voltage and frequency

V. Experimental Results

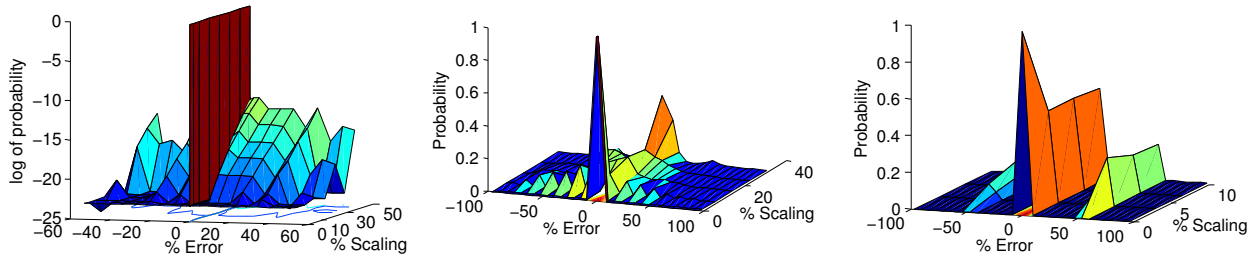
In this section, we describe our experimental setup and present the results of applying the MACACO methodology to different architectures of 32-bit adders and 8-bit Wallace Tree multiplier (WTM) under different levels of over-scaling. We also analyze functionally approximate circuits using different truth table based approximate adders (TTA) [17] and reverse carry propagation (RCP) based adder [13]. DSEC [16] was evaluated using 16-bit accumulator. We compare the run times for using different analysis techniques to analyze these circuits.

A. Experimental Setup

In our implementation, we synthesized our circuits using Synopsys Design Compiler [25]. Our algorithm for generating the untimed equivalent circuit was implemented in C++. We used minisat [26] SAT solver in our maximum error analysis. For generating the error distribution of various approximate circuits, we used CUDD BDD package [27] and gate level Monte-Carlo simulation using Modelsim [28]. All our simulations were carried out on a Linux PC with a 2.66 GHz Intel CoreTM 2 Quad Processor and 4GB RAM.

B. Error Profiling

In this section, we generate the error distribution using the proposed framework and hence study the performance of various approximate circuits. We assume that all the inputs are equi-probable and there is no correlation between the current and the previous input values. Fig. 5 plots the probability of occurrence of various error values with scaling of frequency for different types of adders- Ripple



(a) RCA32 (b) CLA32 (c) HC32

Fig. 5: Error distribution for various kinds of 32 bit adders

Carry Adder (RCA), Carry Look-ahead Adder (CLA) and Han-Carlson Adder (HCA). The error probability was found to increase with scaling as the number of critical paths that fail to meet the delay constraint increases. The error distribution of the circuit depends on the path delay distribution, which determines the number of outputs failing for a particular delay constraint, the probability of activation of individual paths and the functional correlation between different outputs. An interesting insight from our analysis is that although the MSB fails first,(suggesting large errors), the errors observed are frequently much smaller. This is because, when more than one output bit fails, the erroneous values are correlated to the correct values (as the current input can propagate through shorter paths) and there is a correlation between the erroneous bits as they may share certain paths. For instance, when 2 MSB bits fail, there could be an error compensation effect which reduces the overall error introduced.

In the case of RCA, the probability of error increases very slowly as the number of outputs that fail with over-scaling is small. The occurrence of peak near zero error, even at 50% over-scaling, indicates that the error introduced by over-scaling is of very small magnitude for a large number of input combinations. This is due to the functional correlation between the output bits which causes error compensation as explained above. In the case of CLA, the graph flattens out with over-scaling, and it was found that 90% of the time, the errors were found to be less than 30% at all levels of scaling as shown in Fig. 5b. It may also be noted that the probability of positive errors is different from that of negative errors due to assymetry introduced by over-scaling. In the case of Han-Carlson adder, carry merge operations are performed on the even bits only. The generate and propagate signals of the odd bits are transmitted down the prefix tree. As a result, most of the odd bits start failing even when the clock period is reduced by a smaller amount. This leads to occurrence of peaks near the 50% error point and probability of zero error reduces drastically. HCA, thus, shows very poor scalability. Also note that most of the probability is lumped around the 3 peaks shown in Fig. 5c. These regions together account for more than 90% probability. This knowledge can help in the design of suitable error correction schemes to improve scalability.

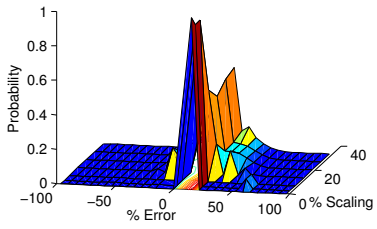


Fig. 6: Error distribution for 8-bit Wallace tree multiplier

Fig. 6 shows the error distribution for the 8-bit WTM with 16-bit RCA at the last stage. The critical path of the WTM goes through the RCA which makes it more scalable. When the frequency was scaled by 13%, the error was found to be less than 12.5% for 94% of the input vectors.

In addition to these circuits, where approximation is achieved through over-scaling, there are other circuits which are functionally approximate. Fig. 7 presents the error distribution for various truth table based approximate adders. In these adders, the errors are introduced in the design of the full adder circuit. The error distribution is mainly decided by the number of inputs for which the errors are introduced, and error compensation between the *sum* and the *carry* bits of the full adder. Adder1 has a much better distribution because the errors are introduced in the *sum* bit only for 2 inputs and one of them is compensated by the errors introduced in the *carry* bit. On the other hand, in the case of Adder3, the *sum* bit is erroneous for 3 inputs and the error introduced in the *carry* bit exacerbate the error due to the *sum* bit. Adder2 has error introduced in the *sum* bit for 2 inputs without any compensation in the *carry* bit while in Adder4, the errors are introduced in the *sum* bit for 3 inputs with compensation in the *carry* bit for 2 of them.

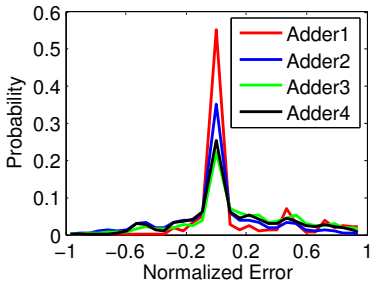


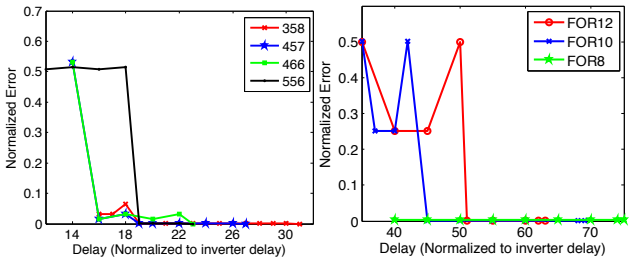
Fig. 7: Truth table based approximation

C. Maximum Error Computation

For certain approximate circuits, we need to verify that the error introduced is always bounded. In the case of RCA and other conventional adders, the MSB fails initially thereby introducing an error of 50% of the maximum value. There are other kinds of approximate circuits designed to ensure that the error is bounded. In this section, we evaluate the scalability of DSEC and RCP. Fig. 8 shows the variation in maximum error for different DSEC and RCP designs. In Fig. 8a, 358 segmentation is one in which the accumulator is divided into 3 stages - 3 bit adder for the MSBs, 5 bit adder for next 5 bits and 8 bit adder for the LSBs. As shown in Fig. 8a, different segmentations allow for different levels of over-scaling. For instance, 358 segmentation has maximum delay but the error introduced by over-scaling is bounded even when the delay is decreased to 50%. On the other hand, 556 segmentation, which has a shorter critical path, fails even with a small amount of over-scaling.

Another kind of functionally approximate adder is the one based on reverse carry propagation. In this case, errors are introduced even in the absence of over-scaling. With over-scaling, the lower order bits fail initially and the overall magnitude of error introduced is very

small. Fig. 8b shows the results for different partitions of 32bit RCA based adder. In FOR12, the length of the forward propagation path corresponds to 12 MSBs. Different partitions allow different levels of over-scaling. In certain cases, it was found that the failing output bits were functionally correlated, which reduced the maximum error with over-scaling. Also, the initial error introduced by RCP was found to vary depending upon the partition.



(a) DSEC (b) RCP

Fig. 8: Variation of maximum error with delay

D. Runtime Comparison

TABLE I: Runtime comparison

Circuit	Logic Gates	BDD Analysis (Sec)	MC Analysis (Sec)	SAT Analysis (Sec)
RCA	62	4	5.43	0.6
CLA	203	5	5.55	0.9
HCA	362	7	5.35	1
WTM	401	17	6.40	1.2
TTA[3]	160	0.48	4.34	0.1

[3]Average over different Truth Table Approximations

Table 1 compares the runtime of the BDD, Monte-Carlo (MC) and the SAT based techniques for circuits of different logical complexity. The Monte-Carlo analysis was carried out with sufficient number of samples for a confidence level of 95%. In the BDD based analysis, the runtime is decided by time taken to generate the BDD representation of the virtual error circuit. Note that the BDD based framework is much faster for analysis of approximate circuits where the BDD is generated efficiently. However, in the case of multiplier, the run time of BDD Analysis is higher than Monte-Carlo based technique. The SAT based approach is much faster but has the limitation that it can produce only the worst-case error.

In summary, the MACACO methodology can be used to analyze

- The trade-offs between various arithmetic unit architectures.
- The error magnitudes that occur with large probability.
- The error distribution for different levels of over-scaling.
- The effect of functional correlation between different failing bits on the error due to approximation.

VI. Conclusion

Approximate computing techniques, which exploit the inherent error-resilient nature of applications, have gained popularity in recent years. However, there does not exist any methodology for systematic analysis and verification of approximate circuits. The proposed MACACO methodology is an initial step in this direction. Future extensions of MACACO could include analysis of complete hardware and software implementations, *i.e.*, analysis of the effects of approximation at the output of the algorithm. One possible approach to this problem would be to use a divide-and-conquer strategy, *i.e.*, consider the circuit as a network of approximate building blocks, analyze the building blocks using the techniques proposed in this work, and propagate error characteristics through the network. We believe that such systematic analysis would significantly facilitate future research efforts in this area, as well as the adoption of approximate computing techniques.

References

[1] M. Breuer, "Hardware that produces bounded rather than exact results," in *DAC*, 2010, pp. 871–876.

[2] N. Shanbhag, R. Abdallah, R. Kumar, and D. L. Jones, "Stochastic computation," in *DAC*, 2010, pp. 859–864.

[3] S. Chakradhar and A.Raghunathan, "Best-effort computing: re-thinking parallel software and hardware," in *DAC*, 2010, pp. 865–870.

[4] R. Bryant and Y. Chen, "Verification of Arithmetic Circuits with Binary Moment Diagrams," in *DAC*, 1995, pp. 535–541.

[5] D. Sahoo, S. Iyer, J. Jain, C. Stangier, A. Narayan, D. Dill, and E. Emerson, "A Partitioning Methodology for BDD-Based Verification," in *FMCAD*, 2004, pp. 399–413.

[6] F. Aloul, M. Mneimneh, and K. Sakallah, "ZBDD-Based Backtrack Search SAT Solver." in *IWLS*, 2002, pp. 131–136.

[7] R. Drechsler and B. Becker, "Ordered Kronecker functional decision diagrams-a data structure for representation and manipulation of Boolean functions," *TCAD*, vol. 17, no. 10, pp. 965 –973, 1998.

[8] R. Hegde and N. Shanbhag, "Energy-efficient signal processing via algorithmic noise-tolerance," in *ISLPED*, 1999, pp. 30–35.

[9] R. Hegde and N. R. Shanbhag, "Soft digital signal processing," *TVLSI*, vol. 9, no. 6, pp. 813–823, 2001.

[10] K. V. Palem, "Energy aware algorithm design via probabilistic computing: From algorithms and models to Moore's law and novel (semiconductor) devices," in *CASES*, 2003, pp. 113–116.

[11] L. Leem, H. Cho, J. Bau, Q. A. Jacobson, and S. Mitra, "ERSA: Error Resilient System Architecture for probabilistic applications," in *DATE*, 2010, pp. 1560–1565.

[12] V. Chippa, D. Mohapatra, A. Raghunathan, K. Roy, and S. Chakradhar, "Scalable effort hardware design: exploiting algorithmic resilience for energy efficiency," in *DAC*, 2010, pp. 555–560.

[13] N.Zhu, W. L. Goh, W. Zhang, K. Yeo, , and Z. Kong, "Design of Low-Power High-Speed Truncation-Error-Tolerant Adder and Its Application in Digital Signal Processing," *TVLSI*, vol. 18, no. 8, pp. 1225–1229, 2010.

[14] A. K. Verma, P. Brisk, and P. Ienne, "Variable latency speculative addition: a new paradigm for arithmetic circuit design," in *DATE*, 2008, pp. 1250–1255.

[15] P. Kulkarni, P. Gupta, and M. Ercegovac, "Trading accuracy for power with an underdesigned multiplier architecture," in *VLSI Design*, 2011, pp. 346–351.

[16] D. Mohapatra, V. Chippa, A. Raghunathan, and K. Roy, "Design of voltage-scalable meta functions for approximate computing," in *DATE*, 2011.

[17] V. Gupta, D. Mohapatra, S. P. Park, A. Raghunathan, and K. Roy, "IMPACT: Imprecise adders for low-power approximate computing," in *ISLPED*, 2011, pp. 409–414.

[18] J. Huang and J. Lach, "Exploring the fidelity-efficiency design space using imprecise arithmetic," in *ASPDAC*, 2011, pp. 579 –584.

[19] A. B. Kahng, S. Kang, R. Kumar, and J. Sartori, "Slack redistribution for graceful degradation under voltage overscaling," in *ASPDAC*, Jan 2010, pp. 825–831.

[20] L. Wan and D. Chen, "Dynatune: Circuit-level optimization for timing speculation considering dynamic path behavior," in *ICCAD*, 2009, pp. 172–179.

[21] M. R. Choudhury and F. Mohanram, "Approximate logic circuits for low overhead, non-intrusive concurrent error detection," in *DATE*, 2008, pp. 903 –908.

[22] D. Shin and S. K. Gupta, "Approximate logic synthesis for error tolerant applications," in *DATE*, 2010, pp. 957 –960.

[23] D. Ernst, N. S. Kim, S. Das, S. Pant, R. Rao, T. Pham, C. Ziesler, D. Blaauw, T. Austin, K. Flautner, and T.Mudge, "Razor: A Low-Power Pipeline Based on Circuit-Level Timing Speculation," in *MICRO*, 2003, pp. 7–19.

[24] K. Keutzer, S. Malik, and A. Saldanha, "Is redundancy necessary to reduce delay?" *TCAD*, vol. 10, no. 4, pp. 427–435, 1991.

[25] Design Compiler Synopsys Inc. .

[26] Minisat, "http://minisat.se/."

[27] CUDD, "http://vlsi.colorado.edu/ fabio/cudd/."

[28] MODELSIM, "http://www.model.com/."