SABER: Selection of Approximate Bits for the Design of Error Tolerant Circuits

Deepashree Sengupta[1], Farhana Sharmin Snigdha[1], Jiang Hu[2] and Sachin S. Sapatnekar[1],
[1]Department of ECE, University of Minnesota*
[2]Department of ECE, Texas A&M University
[1]{sengu026,sharm304,sachin}@umn.edu, [2]jianghu@ece.tamu.edu

ABSTRACT

A wide variety of error tolerant applications supports the use of approximate circuits that achieve power savings by introducing small errors. This paper proposes a fast and novel algorithm for the design of such circuits with the goal of maximizing power savings, constrained by a fixed error budget, through an *analytical* expression to optimally select the number of bits to be approximated. This algorithm outperforms uniform approximation schemes by over 30% in power savings, with negligible computational overhead.

1. INTRODUCTION

Approximate computing has emerged as a new and promising paradigm [1,2] for low power design. This approach uses circuits that deliberately introduce errors to reduce power dissipation, leveraging the inherent error resilience of certain applications to produce good enough results within a specific error margin. Such applications include (but are not limited to) signal processing, multimedia, data mining and other non-safety-critical domains [3,4].

Approximate computing has been explored using circuit level design [5–8] as well as gate-level synthesis [9]. At higher levels of design, [10] and [11] perform various high level synthesis transformations on abstract syntax trees and directed acyclic graphs (DAGs) representing a circuit, respectively, with consideration of approximate components. These methods use coarse-grained decisions to choose among a few approximation options in conjunction with other high level decisions such as scheduling and binding.

We propose **SABER**, an optimization framework at the register transfer level that is solely focused on the tradeoff between approximation with power, but deployed at a much finer granularity level than [10] and [11]. More specifically, our optimization can continuously decide how many bits to be approximated for each arithmetic operation in a design.

The input to our optimization framework is the dataflow graph of the circuit, represented as a DAG whose nodes correspond to arithmetic units that can potentially be approximated, and whose edges indicate the connections between these units. Our formulation maximizes the number of approximation bits in a circuit (which translates to power/area minimization) so that it uses minimal resources under a specified error budget. This work demonstrates results on fixed-point integer arithmetic operations. For convenience, in our exposition we assume operands to be integers since fractional operands can easily be scaled to integers and back again through simple shift operations. To the best of our knowledge, this is the first work on design optimization through analytical methods considering approximation at bit-level granularity. We develop an error model and a fast heuristic such that the computing cost is very low and highly scalable. This low cost provides the potential for our technique to be frequently called in the inner loop of a high level design-space exploration and optimization such as [10]. Our result serves as an initial solution for further refinement by gate-level synthesis methods [9].

For a fair comparison, instead of comparing against a no-approximation scheme (against which large improvements are easy to show), we compare our approach with methodologies where uniform approximation is used to approximate the circuit [5, 12], and demonstrate that our approach can outperform such methodologies by over 30% in power savings, for similar error specifications. The contributions of this paper are summarized as follows:

- **Precharacterization**: We perform gate level characterization of the error variance of multi-bit adders as a function of the number of approximated bits, starting from the least significant bit (LSB). This step is a one-time effort for a library of approximate gates.
- **Error Formulation**: We propose a computationally efficient, and accurate framework for expressing the error variance at the output of a DAG as a function of the number of approximate LSBs within each of its nodes, and model it as a nonlinear expression.
- **Design Optimization**: We formulate an optimization problem to maximize the total approximation in a circuit, constrained to an error budget. Since this optimization is an integer non-linear programming problem, we propose a heuristic to solve this NP-Hard problem. We generate an accurate starting point, followed by a fast approach to obtain the final solution in a simple, analytical form.

Through our optimization routine, we determine precisely if and how each node of a DAG should be approximated and optimize circuit performance under error specifications.

2. ERROR CHARACTERIZATION

The key ingredient of any methodology based on approximate design is an accurate quantification of the error injected into a computation by the approximation scheme. We

*This work was supported in part by the NSF under awards CCF-1162267, CCF-1525925, and CCF-1525749, and the 2016 Doctoral Dissertation Fellowship from University of Minnesota.

DAC '17, June 18 - 22, 2017, Austin, TX, USA
© 2017 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ISBN 978-1-4503-4927-7/17/06…$15.00
DOI: http://dx.doi.org/10.1145/3061639.3062314

use the variance of this error as the error metric to be constrained within a user-specified budget. Here we obtain an analytical expression of this error variance as a function of the total approximation in a circuit.

Let us consider a circuit representing an arithmetic operation with two N-bit operands, X and Y, producing an output, Z. An approximate implementation of the hardware unit yields the benefit of using fewer resources [1] than its exact counterpart. Typically some of the LSBs can be allowed to be erroneous, as this introduces a limited level of approximation. Hence the hardware connected to y LSBs, for example, is approximate, while that connected to the $(N - y)$ most significant bits (MSBs) is accurate. Clearly, the higher the value of y, the greater is the power saving due to the imprecise hardware, although the error is also higher. We use the parameter, y, referred to as the number of approximate LSBs, to quantify the amount of approximation.

We present an approach for characterizing the error variance of a DAG whose nodes are candidates for approximation. We begin by obtaining the error variance of an adder as a function of the number of approximate LSBs, y, in the adder. Using this function, we show how we can compute the error variance of any DAG whose nodes are approximate adders. The results for the adder DAG can be generalized to DAGs whose nodes contain adders, subtractors, multipliers, and dividers since the fundamental element of these operations is an adder [13], with shifters being implemented by appropriately routing the outputs of one DAG node to the inputs of others.

2.1 Error Precharacterization for an Adder

We consider transistor-level approximation where an N-bit approximate adder is implemented as an array of accurate and approximate full adders (FAs). If the error due to y approximate LSBs is e, then e can range from $-(2^y - 1)$ to $(2^y - 1)$, and its exact value depends on the inputs. Typically inputs are assumed to be uniformly distributed random variables [5, 11]. Hence e, being a function of these inputs, can be assumed to be a random variable as well. Let p_x be the probability of e to be x, where $x \in [-(2^y - 1),\ (2^y - 1)]$ is an integer owing to y being an integer. The error means are negligible compared to the variance [5]. Hence we are concerned with the variance, $\sigma_e^2(y)$, given by:

$$\sigma_e^2(y) = \sum_{x=-(2^y-1)}^{(2^y-1)} x^2 p_x \tag{1}$$

Due to the x^2 term in Eq. (1), $\sigma_e^2(y)$ clearly depends on y. If x is uniformly distributed between $-(2^y - 1)$ and $(2^y - 1)$, $p_x = \frac{1}{2^{y+1}-1}, \forall x$, and $\sigma_e^2(y) = (2^{y+1} - 1)^2/12$. We also evaluate $\sigma_e^2(y)$ for normally distributed x in the later part of this section (Fig. 1). In fact, the exponential dependence on y holds for most practical error distribution functions (not just uniform or normal) for $y \le N/2$, N being the word-length of the adder. Additionally, using the fact that $\sigma_e^2(y)$ should be zero for $y = 0$ (no approximation implies zero error), the variance of e is formulated empirically as:

$$\sigma_e^2(y) = a(2^{by} - 1) \tag{2}$$

where a and b are constants, obtained by fitting the error variance for different y, through Monte Carlo simulations.

In this paper we consider the specific transistor-level approximate FAs from [5] and the Lower-part-Or Adder (LOA) from [14] for our analysis. For a particular type of N-bit approximate adder with y approximate LSBs ($N = 10$ considered here), each simulation proceeds by uniformly sampling two inputs, X and Y, from $[0, 2^N - 1]$, to produce an approximate result, Z, and hence the corresponding error, e, can be calculated. Since N is relatively small, we obtain the variance of e for a particular y by exhaustive simulations. This procedure is repeated for $y = 0, \cdots, N - 1$, to obtain a and b in Eq. (2) through regression analysis.

The results are summarized in Table 1. The first column lists the type of adder studied in this work, followed by the respective values of a and b, defined in Eq. (2), in the next two columns, respectively. The fourth and fifth columns list the adjusted R^2 values which refer to the goodness of the fit ($R^2 = 1$ indicates that the fitted model explains all variability) and the root mean square error (RMSE) values of the fitted curve, respectively. Both the quantities indicate that the model is a good fit for the actual data.

Table 1: Fitting parameters for adder error variance.

Adder type	a	b	adjusted R^2	RMSE
appx1	0.05	1.98	1.00	1.04
appx2	0.11	2.01	1.00	1.17
appx3	0.14	2.00	1.00	0.25
appx4	0.10	2.00	1.00	0.11
appx5	0.08	2.00	1.00	0.00
LOA	0.06	2.00	1.00	0.00

The simulations shown above assumed the two inputs, X and Y, of the adder node to be statistically independent. In a general scenario, the two inputs of an adder node within a DAG may have some correlation. Furthermore their distribution may not be uniform, as assumed in the above experiment. To observe the effect of a different input distribution that is correlated, we perform 5000 Monte Carlo simulations on 10-bit adders implemented using the FAs from Table 1, first with two independent 10-bit Gaussian inputs, and then with two correlated Gaussian inputs ($\rho = 0.5$), altering the number of approximate LSBs, y. We compare $\sigma_e^2(y)$, for both cases with that obtained through our model, for different values of y, as shown in Fig. 1.

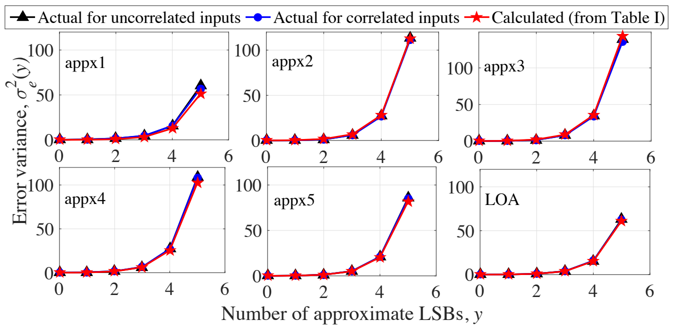

Figure 1: Error variance due to the uncorrelated and correlated adder inputs as a function of y.

In spite of the correlation among the inputs, which are also from a different distribution than what was used for precharacterization, variances of the error generated by the adder using a and b from Table 1, show an excellent match with those obtained from Monte Carlo simulations. Intuitively, this effect arises because the change in the distribution and correlation is more likely to affect the higher order bits, which are not approximated, and the distribution of the lower order bits is close to uniform regardless of correlation and for any reasonable distribution. Hence we consider the generated adder errors as independent random variables to compute the total error variance of a DAG (in Eq. (3)).

2.2 Error Computation of a DAG

Let us consider a DAG consisting of adders and multipliers as shown in Fig. 2(a) with multiple primary inputs (PIs) and outputs (POs). A pair of adder and multiplier from Fig. 2(a) is highlighted in Fig. 2(b) to depict the implementation of the multiplier by add and shift operations. Overall, the DAG has T nodes, each representing an adder, and each edge is associated with a shift operation, denoted by $<<$.

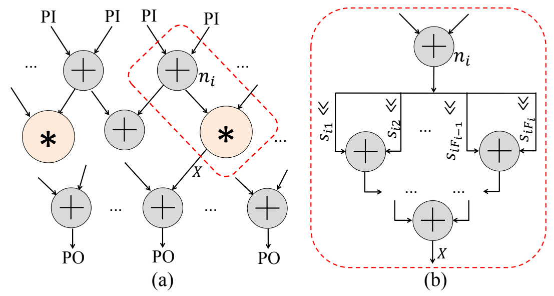

Figure 2: (a) A DAG consisting of adders and multipliers, (b) Representation of multipliers as shift and add operations.

Each node, n_i, is indexed by the subscript, $i \in [1, T]$, and the fanout of n_i is represented by F_i. Each of the F_i fanout edges of n_i is associated with a weight resulting from a shift operation of s_{ij} bits, such that j ranges from 1 to F_i. This nomenclature is depicted in Fig. 2(b).

Let the number of approximate LSBs in n_i be represented by y_i, hence the generated error variance in n_i is $\sigma_e^2(y_i)$, and is obtained by substituting $y = y_i$ in Eq. (2). The error generation among different approximate operations can be assumed to be independent for all practical purposes. However, error propagation exhibits structural correlation since the approximation in n_i not only affects its immediate fanouts, but also those in its fanout cone, through the edges (and the associated weights) connecting n_i to a PO via the transitive fanouts. We use the error sensitivity, β_i, of n_i, to a PO, to capture the structural correlations within the DAG. For a single PO, if an error, e, at n_i results in an error, E_i, at the PO, then $\beta_i = E_i/e$ is computed by a depth-first search of the DAG [11]. The total error variance, σ_t^2, which is a commonly used error metric [8] in the DAG, is obtained as:

$$\sigma_t^2 = \sum_{i=1}^{T} \sigma_e^2(y_i)\beta_i = \sum_{i=1}^{T} a(2^{by_i} - 1)\beta_i \tag{3}$$

where β_i is alternatively called the β value of the node, n_i.

When there are multiple POs in a DAG, to minimize error on all the POs, we simply add a dummy (but accurate) adder node with all the POs. This node is not a part of the design but conceptually indicates the summation of error variances of all nodes to compute the total error variance, σ_t^2 for a multioutput circuit. The addition of this dummy node is a simple device that enables the depth-first traversal of the DAG to compute the β values of the *real* nodes.

3. OPTIMIZATION THROUGH SABER

In this section, we outline the SABER algorithm, which yields the number of approximate LSBs in each node of the DAG, maximizing the total power and area savings, while satisfying a specified error budget.

We first explain our proposed optimization problem, which is NP-Hard, and obtain a feasible solution by relaxing some of the constraints, to make it tractable. Next we propose a heuristic to solve the original problem, using the solution of the relaxed problem.

3.1 The Optimization Problem

Let us consider a DAG with T adder nodes. The power savings increase with increasing levels of approximation in the DAG, and all components of the power savings (dynamic and leakage) are proportional to the number of approximate bits, i.e., the number of approximate FAs. For adders, the proportionality of power savings to the number of approximate FAs has been empirically observed in previous work ([5] and [12]). Even for a multiplier node, the linear proportionality holds true, because as discussed in Sec. 2.2, we decompose it into its constituent FAs, and hence the power savings are linear in the number of FAs in the decomposed graph. Therefore, the total number of approximate LSBs in the DAG, $\sum_{i=1}^{T} y_i$, is a good surrogate objective function that captures the essential trend of power savings, which we aim to maximize. The total error variance, σ_t^2, accumulated as a result of this approximation is given by Eq. (3). If the specified error variance budget is m, then σ_t^2 must be less than m. We thus formulate the optimization problem as:

$$\max \sum_{i=1}^{T} y_i, \quad \text{s.t} \sum_{i=1}^{T} a\left(2^{by_i} - 1\right)\beta_i \leq m, \quad y_i \in \mathbb{Z}^+ \tag{4}$$

where $\mathbb{Z}^+$ represents the set of non-negative integers. The constraint, $y_i \in \mathbb{Z}^+$, arises because the number of approximate LSBs in a node cannot be negative or fractional. A feasible solution to the problem, which satisfies the error budget, always exists: it is the zero approximation solution. However, generating the optimal solution is NP-Hard since (4) is an integer non-linear problem. Hence we *relax* the optimization problem, to make it tractable, and obtain a feasible solution. For this, we first remove the constraint, $y_i \in \mathbb{Z}^+$ in (4), and then convert the inequality constraint into an equality, since the optimal solution for the new maximization problem, will lie on the constraint surface. We obtain the solution through Theorem 1.

Theorem 1 *If the relaxed optimization from* (4) *is,*

$$\max \sum_{i=1}^{T} y_i, \quad s.t \sum_{i=1}^{T} a(2^{by_i} - 1)\beta_i = m \tag{5}$$

with β_i being the error sensitivity used in Eq. (3)*, then the solution, $\widetilde{y}_i$, is obtained as:*

$$\widetilde{y}_i = Y - \frac{1}{b}\log_2(\beta_i) \tag{6}$$

$$\text{where } Y = \frac{1}{b}\log_2\left[\left(m + a\sum_{i=1}^{T}\beta_i\right)(aT)^{-1}\right] \tag{7}$$

Proof: We rewrite the optimization problem from (5) as:

$$\max \ S = y_1 + \sum_{i=2}^{T} y_i \tag{8}$$

$$\text{s.t } a(2^{by_1} - 1)\beta_1 + \sum_{i=2}^{T} a(2^{by_i} - 1)\beta_i = m \tag{9}$$

Hence using Eq. (9), we obtain y_1 as:

$$y_1 = \frac{1}{b}\log_2\theta \tag{10}$$

$$\text{where } \theta = [m + a\beta_1 - a\sum_{i=2}^{T}(2^{by_i} - 1)\beta_i]/a\beta_1 \tag{11}$$

We rewrite, $S = \frac{1}{b}\log_2\theta + \sum_{i=2}^{T} y_i$, by substituting y_1 from Eq. (10) in Eq. (8). To maximize S, we set $\frac{\partial S}{\partial y_i} = 0$.

$$\therefore \quad 2^{by_i} = \frac{\theta\beta_1}{\beta_i} \tag{12}$$

Substituting 2^{by_i} in Eq. (11) and simplifying, we obtain:

$$\theta\beta_1 = \left(m + a\sum_{i=1}^{T}\beta_i\right)(aT)^{-1} \tag{13}$$

We obtain the result by substituting $\theta\beta_1$ in Eq. (12). $\square$

Next we impose the constraint on y_is to be integers. Since the $\widetilde{y}_i$s from Eq. (6), are not guaranteed to be integers, we use Lemma 1 to obtain a feasible solution.

Lemma 1 *A feasible solution of* (4) *is given by* $\lfloor \widetilde{y}_i \rfloor$, *where* $\lfloor . \rfloor$ *represents the* floor *function, and* $\widetilde{y}_i$ *is the optimal solution of the relaxed problem,* (5)*, and is defined in Eq.* (6)*.*

Proof: The left hand side (LHS) of the constraint in (4) is a monotonically increasing function of the state variables. Since, $\lfloor \widetilde{y}_i \rfloor \le \widetilde{y}_i$, if $\widetilde{y}_i$ is a feasible solution (i.e., ensures the LHS to be less than m), then so is $\lfloor \widetilde{y}_i \rfloor$. □

Since, the solution from Lemma 1 may be suboptimal, or even negative, we propose heuristics to address these issues.

3.2 Heuristics to Solve the Original Problem

We attempt to obtain the number of approximate LSBs, $\hat{y}_i$, in node, n_i, of the DAG in Fig. 2, by pushing the $\lfloor \widetilde{y}_i \rfloor$s towards the constraint surface while ensuring non-negativity.

For this, we first define $X = \lfloor Y \rfloor$ (with Y defined in Eq. (7)), so that each node now has $X - \frac{1}{b}\log_2 \beta_i$ approximate LSBs. This expression arises out of Eq. (6), where we apply the *floor* function only to a part of the solution, $\widetilde{y}_i$. Next we use Theorem 2 to obtain a parameter, K, denoting the number of nodes to which we add one more approximate LSB while satisfying the error constraints, thus further increasing the objective function in (4), while keeping the solution, feasible.

Theorem 2 *If the nodes of a DAG are indexed in the increasing order of their* β *values, such that each node,* n_i*, has* $(X - \frac{1}{b}\log_2 \beta_i)$ *approximate LSBs, where* $X = \lfloor Y \rfloor$*, and* Y *is defined in Eq.* (7)*, then the number of first* K *nodes to which one more approximate LSB can be added to satisfy the error constraint,* m*, is given by:*

$$K = \left\lfloor \left(m + a\sum_{i=1}^{T} \beta_i - a2^{bX}T \right) \Big/ \left(a2^{bX}(2^b - 1) \right) \right\rfloor \quad (14)$$

where T *is the total number of adder nodes, and* $K < T$.

Proof: Comparing the impact of adding one more approximate LSB to two nodes, n_1 and n_2, with β values, β_1 and β_2, respectively, where $\beta_1 < \beta_2$, we observe that n_1 introduces lower error in the DAG compared to n_2. In other words, for the same increase in approximation, the total error incorporated is lower if we start increasing the number of approximate LSBs in the nodes in the order of their increasing β values. Hence we renumber the nodes in increasing order of the β values in Theorem 2, so that $\hat{y}_i = \lfloor X - \frac{1}{b}\log_2 \beta_i + 1 \rfloor$ for the first K nodes, while for the rest, $\hat{y}_i = \lfloor X - \frac{1}{b}\log_2 \beta_i \rfloor$, $i \in [1, T]$. The new error variance with the increased number of approximate LSBs should satisfy the error constraint, m, in (5), such that,

$$\sum_{i=1}^{K}(2^{b(X-\frac{1}{b}\log_2 \beta_i + 1)} - 1)\beta_i + \sum_{i=K+1}^{T}(2^{b(X-\frac{1}{b}\log_2 \beta_i)} - 1)\beta_i = \frac{m}{a}$$

Simplifying $\left(2^{b(X-\frac{1}{b}\log_2 \beta_i)} - 1\right)\beta_i$ as $2^{bX} - \beta_i$, we obtain:

$$\frac{m}{a} = \sum_{i=1}^{K}(2^{b(X+1)} - \beta_i) + \sum_{i=K+1}^{T}(2^{bX} - \beta_i) \quad (15)$$

Expanding the right hand side of Eq. (15), we obtain:

$$m = a2^{bX}(2^b - 1)K - a\sum_{i=1}^{T}\beta_i + a2^{bX}T \quad (16)$$

Hence K is obtained by simplifying the above equation. Additionally, $K < T$, since starting with $\lfloor \widetilde{y}_i \rfloor$, we can never increase all the $\lfloor \widetilde{y}_i \rfloor$s by one and remain in the feasible region, because such an increment will lead to $\lceil \widetilde{y}_i \rceil$ ($\lceil . \rceil$ being the *ceiling* function), and if it were in the feasible region, Theorem 1 would have chosen that solution over $\widetilde{y}_i$. □

This analysis till now holds true even if some of the $\hat{y}_i$s are negative. However, since a negative $\hat{y}_i$ does not have any physical meaning, we need to reassign these values to zero, which in turn may violate the error constraint in (4). To address this non-trivial issue we propose Algorithm 1 to obtain non-negative approximate LSBs while satisfying the error budget, m. Algorithm 1 first resets the negative $\hat{y}_i$s to zero, and then, by decreasing the other $\hat{y}_i$s, computes the error to be compensated for. This compensation is performed by reducing $\hat{y}_i$ of the nodes corresponding to the highest β_i value as much as possible, before proceeding to the next with the second highest β value, and so on. The complexity of our algorithm is dominated by the sorting of nodes in term of their β values, so that the overall complexity is $O(T \log T)$.

Algorithm 1 Algorithm to ensure non-negativity of $\hat{y}_i$.

```
Input: $\hat{y}_1, \cdots, \hat{y}_T$ values with decreasing order of $\beta$ values.
Input: Error constraint, $m$, of the optimization problem
    in (4).
Output: Updated non-negative $\hat{y}_1, \cdots, \hat{y}_T$ values.
 1: Initialize $C \leftarrow 0$ //Need to compensate error by $C$
 2: for each $j$ from 1 to $T$ do
 3:     if $\hat{y}_j < 0$ then
 4:         $\hat{y}_j \leftarrow 0$ //Setting negative $\hat{y}_j$ to 0
 5:     end if
 6:     $C \leftarrow C + a(2^{b\hat{y}_j} - 1)\beta_j$ //Error from positive $\hat{y}_j$s
 7: end for
 8: if $C - m < 0$ then
 9:     exit //Resetting $\hat{y}_j$ to 0 did not violate error
10: end if
11: Set $C \leftarrow C - m$ //Compensate by $C - m$, $\because C \ge m$
12: for each $i$ from 1 to $T$ with decreasing order of $\beta_i$ do
13:     if $C \le a(2^{b\hat{y}_i} - 1)\beta_i$ then
14:         Find $Z$ where $(2^{b\hat{y}_i} - 1) - (2^{b(\hat{y}_i - Z)} - 1) = \frac{C}{a\beta_i}$
15:         $\hat{y}_i \leftarrow \hat{y}_i - \lceil Z \rceil$
16:         exit
17:     else
18:         $C \leftarrow C - a(2^{b\hat{y}_i} - 1)\beta_i$
19:         $\hat{y}_i \leftarrow 0$
20:     end if
21: end for
```

4. EXPERIMENTAL RESULTS

We implement SABER in MATLAB R2015b on a 64-bit Ubuntu server with a 3GHz Intel® Core™2 Duo CPU E8400 processor. We consider the appx5 approximate adder which uses transistor-level approximation [5] for our analysis, and consider two examples to demonstrate SABER.

4.1 Optimization Results on an Example DAG

First we demonstrate the results of using SABER through an example structure, DAG10, with ten nodes, as shown in Fig. 3. Each node, n_i, can be represented by a 20-bit adder, where the approximation is introduced by replacing y_i LSB FAs with approximate FAs. A dummy node has been added as explained in Sec. 2.2, to obtain the error sensitivity (i.e., the β value) of the other nodes to the output. Each β_i is obtained by finding the number of paths from n_i through a depth-first search of the DAG considering the edge weights, and is depicted in Fig. 3.

We demonstrate our results for three error variance budgets, m = 1K, 10K, 100K, corresponding to the allowable error variance at the output of the dummy node. For comparison purposes, we consider the commonly-used uniform approximation case (e.g., in [5, 12]), when the number of approximate bits in each node is identical.

Using SABER, we compute the number of approximate LSBs in each of the ten nodes, whose distributions among

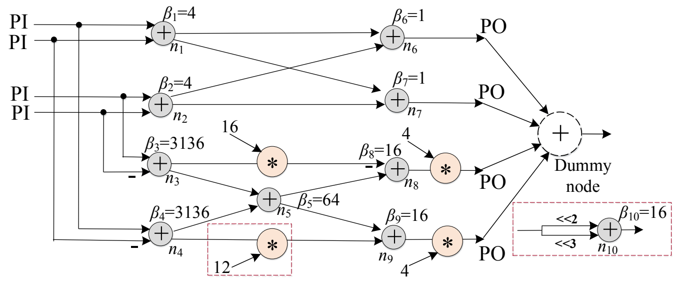

Figure 3: Structure of DAG10 with ten nodes.

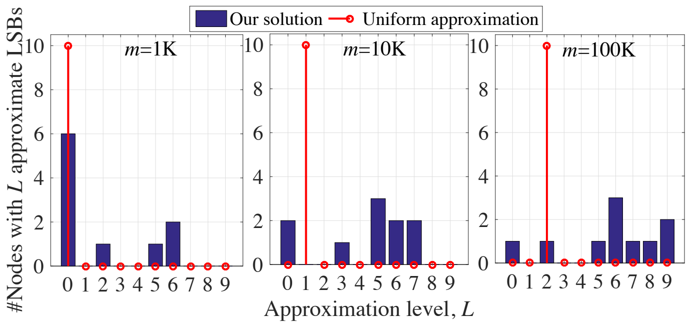

Figure 4: Distribution of the number of approximate LSBs over the ten nodes of DAG10.

the nodes are depicted in Fig. 4 for the three error variance budgets, m. Since the uniform approximation results in the same number of approximate LSBs in each node, it is depicted by the stem plot, with a single stem of height ten, the rest being zero, in the same figure. Due to the dependence of the number of approximate LSBs in a node on its β value through Eq. (6), the distributions in Fig. 4 are also determined by the distribution of β values. As m increases, more approximation is possible, and this is seen by the rightward horizontal shift in the bar charts.

Table 2: Total number of approximate FAs by SABER and the uniform approximation case for different error budgets.

Target, m ($\times 10^3$)	Achieved, σ_t^2 ($\times 10^3$)	Total approximate LSBs		Power savings (%)	
		SABER	Uniform	SABER	Uniform
1.00	0.98	19 LSBs	0 LSB	9.50	0.00
10.00	9.49	44 LSBs	10 LSBs	22.00	5.00
100.00	92.88	58 LSBs	20 LSBs	29.00	10.00

The target, m, and the actual error variance, σ_t^2, arising out of the three approximate configurations, are listed in the first two columns of Table. 2. The difference between them arises due to the relaxation of the original problem in (4), and application of the proposed heuristics to make our solution feasible. However, this difference is less than 8% for all three cases, indicating the effectiveness of our methodology. The total number of approximate LSBs by SABER and the uniform approximation case, are listed in the next two columns of Table. 2, indicating that SABER clearly outperforms the uniform approximation for DAG10, from which power savings are calculated and listed in the last two columns. The proportionality constant between the number of approximate FAs and percentage power savings is 0.5%, as evident from the last four columns of the table, since each adder node is 20-bit, and 10 such adder nodes in the DAG lead to a total of 1/200×100=0.5% approximate FAs for each approximate LSB within the DAG. Since we use appx5 version of FA for approximation, the power savings for the entire DAG is also scaled by the same factor (0.5%) as appx5 has negligible power consumption compared to the exact FA [5]. Hence approximating k LSB FAs in the DAG leads to 0.5k% power savings, which is accurate to a first order to indicate the advantage of using SABER to maximize power savings over the uniform approximation, without performing logic synthesis.

4.2 Optimization Results on FIR Filters

We evaluate our algorithm on a real-world example, by checking the sound quality of filtered signals from an approximate finite impulse response (FIR) filter. The results of filtering by approximate filters designed through SABER have been summarized within a compressed folder and uploaded to http://conservancy.umn.edu/handle/11299/185544. The signals under study comprise of 150K samples of eight different genres of audio clips [15] sampled at their prespecified frequency of 22.05KHz [16] and mixed with a high frequency noise. We constrain the signal to noise ratio (SNR) degradation between an exact filter and an approximate filter to be 50dB, to ensure comfortable loudness and clarity.

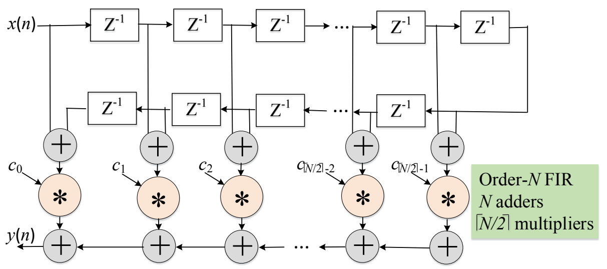
$x(n)$
$y(n)$
Z^{-1}
c_0
c_1
c_2
$c_{\lceil N/2 \rceil-2}$
$c_{\lceil N/2 \rceil-1}$
Order-N FIR
N adders
$\lceil N/2 \rceil$ multipliers

Figure 5: An FIR filter with symmetric coefficients [17].

The normalized pass band and stop band frequencies[1] of the FIR filter are 0.50 and 0.65, respectively, and the minimum order filter that MATLAB generated, had order=33.

The filter coefficients have been scaled by 1024 to facilitate integer arithmetic. All adders have word length of 20 bits, and the multiplications are implemented by array multipliers with add and shift operations. Since the coefficients are symmetric in an order-N FIR filter, we can reuse multipliers [17], resulting in only $\lceil N/2 \rceil$ multipliers and N adders, as shown in Fig. 5. In our order-33 FIR filter, the first 16 coefficients could be implemented simply by 30 adders (and shifters) based on their binary decomposition. Additionally the filter requires 33 adders. Hence the resulting DAG for the optimization problem in (4) has $T = 63$ nodes.

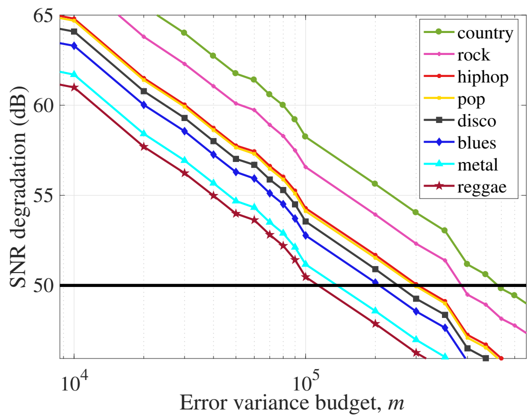

Figure 6: Tradeoff plot of SNR degradation with error variance in the FIR filter used to select the error budget.

To formulate the optimization problem, we need the error variance budget for each audio clip. Since the different

[1]Normalized pass (stop) band frequency is twice the ratio of the actual pass (stop) band frequency to the sampling frequency [18].

genres of music are differently sensitive to approximation in the FIR filter, we select the respective error budgets from the tradeoff plot of SNR degradation versus error variance for each clip as depicted in Fig. 6. We obtain this plot by sweeping the error variance, m, to first obtain different configurations of approximate filters using SABER. We then filter the noisy signal using each such filter and compute the SNR degradation from the accurately filtered signal. Using this plot, we can select the target error variance, m, for various target SNR degradation values, an example of which is shown for 50dB by the line in Fig. 6. This plot can be generated very quickly since SABER takes less than a second to generate one configuration of the approximate filter.

For demonstration purposes, we select three values of m ($m = 100\text{K}, 200\text{K}, 400\text{K}$) from Fig. 6, to obtain three different approximate filters. The number of approximate LSBs in each node of the FIR filter for each m is then obtained by SABER, whose distribution among the 63 adder nodes is depicted in Fig. 7. The stem plots depicting the corresponding uniform approximation case are also provided in the same figures as reference.

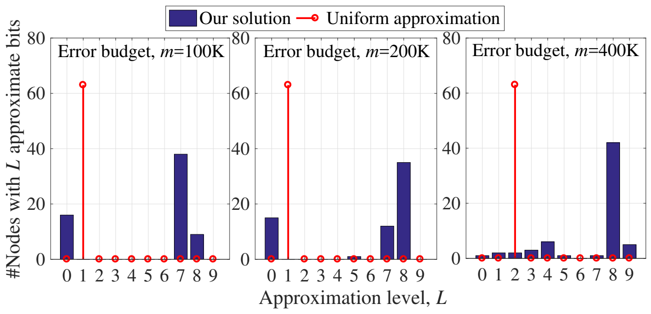

Figure 7: Distribution of the number of approximate LSBs over the 63 nodes of the FIR filter.

The power requirement of the approximate filters as a fraction of that of the accurate filter is listed in Table 3.

Table 3: Power dissipation of the approximate filters by SABER and uniform approximation, for three values of error budget, m, compared to the accurate filter.

	$m = 100\text{K}$	$m = 200\text{K}$	$m = 400\text{K}$
SABER	0.73×	0.70×	0.66×
Uniform approximation	0.95×	0.95×	0.90×
Improvement in power	30.14%	35.71%	36.36%

The second column denotes the values corresponding to the filter obtained by SABER with $m = 100\text{K}$, and by uniform approximation in the first two rows, respectively. The last row shows the percentage improvement of our method over the uniform approximation case for this m. Clearly, our algorithm not only achieves over 27% power savings over the exact implementation, but also outperforms the uniform approximation case by over 30%. These numbers increase as m is increased as seen from the last two columns of the table. Obtaining the solution through SABER can thus lead to significant power savings over the existing methodologies.

Table 4: SNR degradation (in dB) between the accurately filtered signal and those from the approximate filters constructed for different error budgets, m, using SABER.

Genre→	country	rock	hiphop	pop	disco	blues	metal	reggae
$m = 100\text{K}$	58.24	56.57	54.28	54.12	53.55	52.77	51.17	50.47
$m = 200\text{K}$	55.62	53.93	51.69	51.53	50.90	50.16	48.58	47.87
$m = 400\text{K}$	53.03	51.39	49.12	49.00	48.37	47.65	46.03	45.34

The resulting SNR degradations for the eight audio clips while using the three filters are summarized in Table 4. The audio clips are listed in the first row, and the SNR degradations for the error variance budgets, m=100K, 200K, 400K, are listed in the next three rows, respectively. The values are all around 50dB for all m, indicating that the user experience is not compromised in spite of the approximations in the filter, which can be verified by playing the audio clips from http://conservancy.umn.edu/handle/11299/185544. For each clip, the site contains the noisy version, the exact filtered version, and filtered versions corresponding to the three error variance budgets, m, respectively.

5. CONCLUSION

We have proposed a bit-level optimization framework to design approximate circuits under specified error budgets, built upon an analytical expression for the number of approximate LSBs for each computational unit. The runtimes to obtain an approximate configuration of a DAG are shown to be very small due to the closed form solution, and outperforms the conventional approximation methods by over 30% in power savings.

6. REFERENCES

[1] J. Han and M. Orshansky, "Approximate Computing: An Emerging Paradigm For Energy-Efficient Design," in *Proc. ETS*, pp. 1–6, 2013.

[2] M. Shafique, *et al.*, "Invited - Cross-layer Approximate Computing: From Logic to Architectures," in *Proc. DAC*, pp. 99:1–99:6, 2016.

[3] J. Huang, *et al.*, "A Methodology for Energy-Quality Tradeoff Using Imprecise Hardware," in *Proc. DAC*, pp. 504–509, 2012.

[4] E. Swartzlander, "Truncated Multiplication with Approximate Rounding," in *Proc. 33rd Asilomar Conf. Signals, Systems, Computers*, vol. 2, pp. 1480–1483, 1999.

[5] V. Gupta, *et al.*, "Low-Power Digital Signal Processing Using Approximate Adders," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 32, no. 1, pp. 124–137, 2013.

[6] J. M. Jou, *et al.*, "Design of Low-error Fixed-width Multipliers for DSP Applications," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 46, no. 6, pp. 836–842, 1999.

[7] L. Chen, *et al.*, "Design of Approximate Unsigned Integer Non-restoring Divider for Inexact Computing," in *Proc. GLSVLSI*, pp. 51–56, 2015.

[8] F. S. Snigdha, *et al.*, "Optimal Design of JPEG Hardware Under the Approximate Computing Paradigm," in *Proc. DAC*, pp. 106:1–106:6, 2016.

[9] S. Venkataramani, *et al.*, "SALSA: Systematic Logic Synthesis of Approximate Circuits," in *Proc. DAC*, pp. 796–801, 2012.

[10] K. Nepal, *et al.*, "ABACUS: A Technique for Automated Behavioral Synthesis of Approximate Computing Circuits," in *Proc. DAC*, pp. 1–6, 2014.

[11] C. Li, *et al.*, "Joint Precision Optimization and High Level Synthesis for Approximate Computing," in *Proc. DAC*, pp. 1–6, 2015.

[12] L. B. Soares, *et al.*, "Approximate Adder Synthesis for Area- and Energy-efficient FIR Filters in CMOS VLSI," in *Proc. NEWCAS*, pp. 1–4, 2015.

[13] B. Parhami, *Computer Arithmetic: Algorithms and Hardware Designs*. New York, NY: Oxford University Press, 2000.

[14] H. R. Mahdiani, *et al.*, "Bio-inspired Imprecise Computational Blocks for Efficient VLSI Implementation of Soft-computing Applications," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 57, no. 4, pp. 850–862, 2010.

[15] "MARSYAS Data Sets." http://marsyasweb.appspot.com/download/data_sets/.

[16] G. Tzanetakis and P. Cook, "Musical Genre Classification of Audio Signals," *IEEE Speech Audio Process.*, vol. 10, no. 5, pp. 293–302, 2002.

[17] L. Aksoy, *et al.*, "A Tutorial on Multiplierless Design of FIR Filters: Algorithms and Architectures," *Circuits, Systems, and Signal Processing*, vol. 33, no. 6, pp. 1689–1719, 2014.

[18] A. V. Oppenheim and A. S. Willsky, *Signals and Systems*. New Jersey, NJ: Prentice-Hall, 1997.