BROWN

Approximate Logic Synthesis Using Boolean Matrix Factorization

Jingxiao Ma

Advisor: Prof. Sherief Reda

Submitted in partial fulfillment of the requirements for the
Master's of Science

Department of Computer Science
Brown University
May 15, 2020

Contents

Abstract

Approximate computing is an emerging computing paradigm, offering benefits in hardware metrics, such as design area and power consumption, by relaxing the requirement for full accuracy. In circuit design, a major challenge is to synthesize approximate circuits automatically from input exact circuits requiring minimal expert input. In this work we present a method for approximate logic synthesis based on Boolean matrix factorization, where an arbitrary input circuit can be approximated in a controlled fashion. Our methodology enables automatic computation of the dominant elements, *bases*, of the truth table of the circuit, and later combining the bases to approximate the original truth table. Such compression can reduce the complexity of the hardware implementation significantly, while introducing different degrees of error. Furthermore, in our approach, the factorization algorithm can be fine tuned as required by the application, to effectively improve control over degree of approximation. In this work, we provide a unified approach enabling the factorization algorithm to utilize semi-ring algebra, field algebra, and a combination of both for truth table factorization. In addition, we provide an automatic circuit breakdown approach and a design space exploration heuristic to navigate the search space. We implement our methodology using a full stack of open-source tools, and thoroughly evaluate our methodology on a number of representative circuits showcasing the benefits of our proposed methodology for approximate logic synthesis. Finally, we compare our methodology against a well-established library of approximate designs, to demonstrate our approach results in state-of-the-art performance.

I Introduction

Since the emergence of power as the main factor limiting the scale of the computational power, novel techniques have been proposed aiming at reducing the power and energy footprint of conventional computing systems. Approximate computing is an emerging low-power technique where computational accuracy is traded for improvements in hardware cost and complexity, e.g. design area, power consumption or energy cost. Effectively, approximate computing introduces a third dimension (specifically accuracy) to the conventional design area vs. design delay trade-offs. Approximate computing is effective for application domains that inherently tolerate small inaccuracies in their output. Such tolerance can originate from different sources including noise in input data, inherent approximate calculations, or human tolerance to variations in the outputs. Few examples of such domains include signal processing, machine learning, computer vision, and computer graphics.

A primary challenge of approximate computing is to devise techniques for automated approximate circuit synthesis that can generate approximate circuits from arbitrary exact input circuits, while offering a wide range of trade-off between accuracy and hardware metrics. Such techniques, while less optimized for specific designs, enable a more versatile approach where any input design amenable to approximations, can be readily optimized without requiring added guidance from the designer.

The proposed approach utilizes recent advance in multivariate analysis, namely Boolean matrix factorization [1], that can reduce the dimensionality of the problem, by identifying the common bases which can be later combined to yield the original Boolean matrix. Our methodology operates on truth tables and introduces approximations in the circuit by simplifying the input truth table based on statistical analysis [1], [2]. Compared to our previous publications, this article provides the following contributions.

- We provide a unified approach to approximate logic synthesis utilizing matrix factorization. Our approach utilizes three factorization techniques, relying on different algebra. Such methodology introduces an exponentially large search space which requires careful navigation.
- In order to improve the scalability of our methodology, we partition an input circuit into manageable subcircuits [3], and perform a detailed design space exploration over factorization degrees of subcircuits to optimize the resulting approximate top-level design. Meanwhile, our approach is able to handle various error metrics, such as Normalized Hamming Distance (HD), Mean Absolute Error (MAE), *etc*.
- We provide a more comprehensive set of experiments, including the well established EPFL [4] and ISCAS '85 [5] benchmark suites commonly used in the literature. Evaluation on 15 total

benchmarks, we clearly demonstrate the versatility of our proposed technique. Furthermore, we compare our approximation designs against EvoApproxLib, a library of approximated adder and multiplier circuits, in order to show that our approach reaches state-of-the-art performance.
- We implement our approach using a full stack of open-source tools, while adopting a more runtime aware approach and introducing techniques, e.g. parallelization and computation reuse, to reduce the runtime overhead of our methodology.

The organization of this paper as follows. In Section II we overview relevant previous work on approximate logic synthesis and the broader approximate computing paradigm. Next, in Section III we discuss the necessary background on Boolean matrix factorization, as it related to our methodology. In Section IV, we describe our new approaches, mainly the XOR field-based circuit approximation method We also describe the integration of our approach in a circuit decomposition and design space exploration technique. We provide a comprehensive set of experimental results in Section V. The conclusions of this work are summarized in Section VI.

II Previous Work

Within the approximate computing paradigm, approximations can be introduced in many different levels of the computing stack [6], [7], ranging from the software and algorithm [8]–[12], to system architectures [13]–[15], and circuit and transistor levels [16]–[20]. In this section, we briefly discuss some of the existing researches to explore different aspects of approximate computing and their findings.

First, in software and algorithmic domains, one popular methodology is loop perforation, where the iterative computation can be stopped prematurely, to reduce the computation cost while introducing errors in precision [8], [9]. In this domain, approximations based on approximate GPU kernels [10], approximate compression [11], and approximate parallelization [12] have also been proposed.

In addition, on the computer architecture front, approximate instruction set architectures (ISA) have also been explored. Esmaeilzdeh *et al.* proposed an approximate processing pipeline within which approximate versions of all main arithmetic and logical operations are implemented as an ISA extension [13]. Similarly, utilization of approximate computing techniques for many specific computing components, such as dynamic random access memories (DRAM) [14], and cache and register file subsystems [13], [15] have also been proposed.

On circuit level, voltage over-scaling (VOS) has received significant attention [16]. Here, the operating voltage is reduced beyond safe operation thresholds reducing the energy consumption. However, the indeterministic nature of such approximations has resulted in limited applicability of such methodologies. Logic approximation of the underlying hardware have also been explored. Here, two main approaches have been evaluated; (i) architectural approximations of specific designs (such as adders and multipliers), and (ii) automated approximations of arbitrary circuits. Arithmetic blocks, due to their utilization in many other applications, have received significant attention. Here, approximate adders [17], multipliers [18], [19], and dividers [20] are few examples where architectural approximations for specific hardware blocks are proposed.

Approximate synthesis methodologies operating on arbitrary circuits have also been proposed [21]–[28]. For example, in SALSA, a miter is created to compute the error between the original circuit and the approximated circuit [22] using existing methodologies in logic synthesis. The don't cares of the outputs of the approximate circuit with respect to outputs of the difference circuit can be used to simplify the approximate circuit using regular logic synthesis techniques. This approach was extended in ASLAN [23] to model error arising over multiple cycles. In SASIMI [24], a technique is proposed to identify similar signals, such that their values agree over a large number of input test cases, and then substitute one for the other, simplifying the logic.

For higher-level synthesis, ABACUS generates variants of an input high-level Verilog description file by applying a set of possible transformations on the circuit to generate a set of mutant approximate circuit variants [21]. A multi-objective design space exploration technique is then used to identify the best set of approximate variants. Vasicek *et al.* propose evolutionary approaches, EvoApprox, on datapath circuits that are composed of basic arithmetic blocks (e.g., adders and multipliers) and logic blocks [26], where the exact circuit is encoded in a string-based representation as a "chromosome" and then a genetic algorithm mutates the circuit to create approximate versions as long as the error is kept below target. Raising the approximate synthesis to C-based design, Lee *et al.* propose a new technique to synthesize approximate circuit directly from C descriptions [29].

Finally, approximate computing techniques have also been deployed in specific applications such as deep learning [30], and computer vision. More recently, impact of approximate computing on end-to-end systems such as biometric security [31], and smart camera system [32] has also been studied.

III Background

In this chapter, we describe problem of Boolean matrix factorization (BMF), as it forms the basis of our methodology. Then, we briefly discuss some existing algorithms of BMF. Matrix factorization (or decomposition) is a class of algorithms that propose to factor an input matrix $n \times m$ $\mathbf{A}$ into two matrices: a $n \times f$ matrix, $\mathbf{B}$, and a $f \times m$ matrix, $\mathbf{C}$, such that $\mathbf{A} \approx \mathbf{BC}$. In many applications the *factorization degree*, f, is required to be smaller than m in approximations in the multiplication results. Note that one can interpret the columns of $\mathbf{B}$ as *factors* or *bases* that are linearly combined using $\mathbf{C}$.

While generic matrix factorization algorithms allow for both negative and positive matrix entries, non-negative matrix factorization (NNMF) restricts the elements to non-negative values [33]. Non-negative values occur in many physical domains, such as computer vision and document clustering [34]. More recently, NNMF has been extended to Boolean matrix factorization, where all elements of all matrices are limited to '0's and '1's. Different algebra can be used for the arithmetic [35], [36]. Boolean matrix factorization algorithms have many applications, including data mining, noise detection, and document clustering.

Boolean matrix factorization has been proved to be NP-hard [37], which can also be formulated as an optimization problem solving,

$$\text{argmin}_{\mathbf{B},\mathbf{C}} |\mathbf{A} - \mathbf{BC}|, \tag{1}$$

where the elements of $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{C}$ are '0 or '1'. Therefore, many algorithms take a heuristic approach. For example, in ASSO, an association matrix is computed as candidates of *bases* vectors using association rule mining [37]. Intuitively, the association matrix evaluates the likelihood among all pairs of columns in the input matrix. Then, for each candidate *base* in the association matrix, ASSO calculates a paired column by enumerating all possibilities, and picks the optimal pair in order to greedily cover '1's in the input matrix. As it is fast and straightforward, there exists one drawback, such that errors of covering '0's by '1's are irreversible. Therefore, some improvements have also been proposed, such as clustering input matrix before factorization or transposing input matrix.

Besides heuristic approach, some other methods have also been studied, which first solve non-negative matrix factorization problem and then extend to binary case [38]. Penalty function algorithm attempts to build a loss function and optimize by computing derivatives. Thresholding algorithm aims to solve $\mathbf{B}$ and $\mathbf{C}$ in real numbers, and then find thresholds to binarize two matrices. Recently, more methodologies are proposed, such as using Minimum Description Length principle [36] or Message Passing [39].

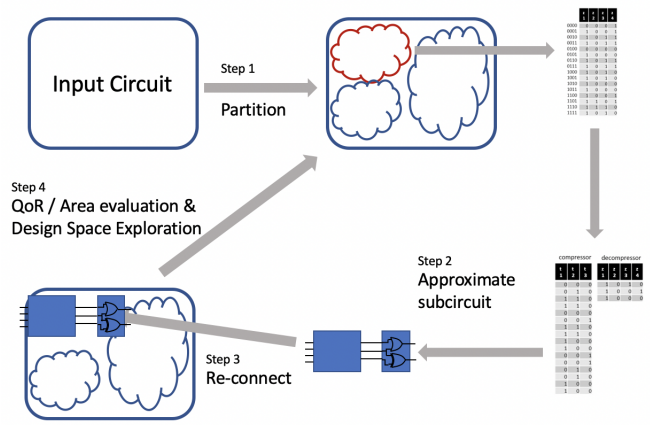


Fig. 1. General flow of BLASYS for approximate logic synthesis using Boolean matrix factorization (BMF).

IV Proposed Methodology

In this chapter, we first describe our proposed BLASYS methodology for utilizing Boolean matrix factorization (BMF) in automated approximate logic synthesis. Here, we also discuss our techniques for improving accuracy and versatility of our methodology, by introducing XOR algebra and weighting schemes in subsection IV-A. Later on, we discuss the consideration required, when applying the proposed methodology on larger circuits. Since, the proposed BMF based methodology operates on truth tables, in order to keep the truth table within manageable size, we propose to use circuit partitioning. We then introduce methodologies for design space exploration (DSE) of the resulting search space in subsection IV-B.

Figure 1 illustrates the general flow of BLASYS algorithm. As demonstrated in the figure, an input circuit can optionally be decomposed into smaller subcircuits, if required by its input size. Next, each subcircuit is approximated to a specific degree, and the approximate components are connected together to generate the approximate design. For each approximate design, the Quality of Results (QoR) and design area are evaluated, which is then used for design space exploration and guide the factorization degree during next iteration. Next subsections describe, in more details, the exact inner workings of the proposed technique.

A. Approximate Synthesis Using Boolean Matrix Factorization

As discussed in Section III, Boolean matrix factorization is a special extension of matrix factorization, where all elements of all matrices are limited to '0's and '1's. There exists an inherent connection between logic circuits and Boolean matrix, where truth tables of circuits can be represented by Boolean matrix.

To use Boolean matrix factorization methods for approximate logic synthesis, the truth table of the input circuit is first generated and given as the input matrix for a binary matrix factorization algorithm, where the factorization degree, f, is chosen to be smaller than the number of outputs of the original circuit. The two factorized matrices from the algorithm are then treated as truth tables synthesized into two subcircuits and connected together to generate the approximate circuit as illustrated in Figure 2. In Figure 2, the first subcircuit receives the n outputs as the original circuit, but instead produces $f < m$ outputs, and thus referred to as the *compressor circuit*. The second subcircuit receives $f < m$ inputs and produces m outputs and thus referred to as the *decompressor* circuits. In prior work where only semi-ring Boolean algebra is considered, the implementation of the decompressor is very simple as it uses a network of only OR gates [1].

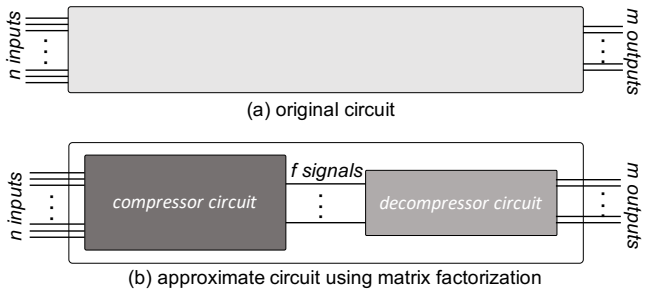


Fig. 2. Utilization of binary matrix factorization for approximate logic synthesis. (a) an arbitrary input circuit, and (b) the compressor and decompressor circuits used in binary matrix factorization methodology.

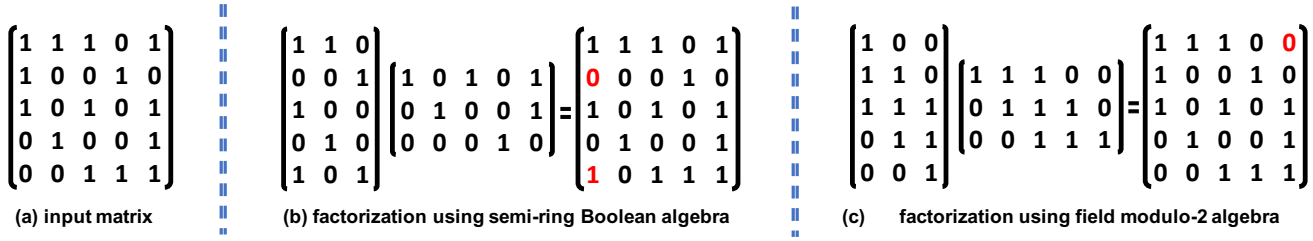


Fig. 3. Example of binary matrix factorization using different algebra. (a) input matrix, (b) matrix factorization using Boolean algebra where addition is carried out using logical ORs, and (c) matrix factorization using modulo-2 algebra, where the addition is carried out using logical XORs. The errors are highlighted in red.

1) Factorization Algebra

Boolean matrix factorization aims at minimizing the number of mismatches between an input matrix and the approximate multiplication result of the factorized matrices. In Boolean matrix factorization, the multiplications are carried out using the logical AND operation, and the addition operation can be either based on semi-ring Boolean algebra, or field modulo-2 algebra. In the case of Boolean matrix factorization (BMF), the algebra implements a semi-ring algebra, where the addition is carried out using logical OR, i.e., $1+1=1$. In the case of field modulo-2 algebra, the addition is carried out using logical XOR, i.e., $1+1=0$ Figure 3 shows an example of an input matrix as well as the factorized matrices and their multiplication result for both Boolean and Modulo-2 arithmetic.

Using different arithmetic can result in significantly different characteristics in the factorized matrices as well as the best approximation degree. In the specific case of Figure 3, modulo-2 algebra generates better quality of results. Next we describe the utilization of binary matrix factorization methodologies in the approximate logic synthesis problem.

One possible drawback of using OR-based Boolean arithmetic is that the number of bases from $\mathbf{B}$, i.e., outputs of the compressor circuits, that can be combined to produce one column in $\mathbf{C}$, i.e., output of the decompressor circuit, is limited. ORing two bases from $\mathbf{B}$ with a '1' in the same location will lead to a '1' in the corresponding location in the resulting output column, and this result will not change regardless of any additional bases that can be further ORed with the two. In contrast, in modulo-2 algebra, $1+1=0$, thus a '1' can be reduced back to '0' and therefore combining additional bases in modulo-2 implementation can offer more diversity in the results. Interestingly, modulo-2 based approximate logic synthesis closely resembles that of the Boolean based approach, where the only differences are (1) a modulo-2 approach is utilized for the matrix factorization, and (2) the decompressor circuit needs to be mapped to network of XOR gates instead of a OR gates.

Currently there are no modulo-2 matrix factorization algorithms and the complexity of the problem

is unknown [36]. Note that the Boolean counterpart is proven to be NP-Hard, and therefore all existing algorithms are based on heuristics. To enable our methodology using modulo-2 arithmetic, we devise a simple heuristic based on the methodologies used for the Boolean matrix factorization. More specifically, we use ASSO [35], [36] for initial matrix factorization, where we further do an exhaustive search for the decompressor matrix to minimize the error assuming modulo-2 arithmetic. Note that this operation incurs a timing complexity of $O(m2^n)$ as different columns of the decompressor circuit can be identified independently.

Finally, as different columns of the decompressor matrix represent different combinations of the compressor circuits, one can mix the OR-based and XOR-based methodologies, where some outputs are implemented using OR and other outputs are implemented using XORs, i.e., the decompressor circuit uses both OR and XOR gates. We refer to this approach as XOR/OR, as it chooses the better outcome of OR versus XOR results to implement. We will evaluate OR, XOR and OR/XOR methodologies in the experimental results highlighting the benefit of each in different circumstances.

2) Output Weights

In BMF algorithms, the objective is to minimize $||\mathbf{M}-\mathbf{BC}||_2$, which translates to Hamming distance in Boolean systems. In approximate circuit design, however, such metric does not provide a good representation of QoR in many cases. As an example, if the columns of a m-column matrix represent an m bit signal, minimizing the Hamming distance as the cost function can lead to significant errors in numerical value. For instance, a bit flip in the least significant bit will lead to a numerical error of 1, whereas a bit flip in the n^{th} bit leads to an error of 2^{n-1}.

To account for the bit significance, we augment existing BMF algorithms with custom QoRs enabling weighted cost functions. Specifically, we propose to define the cost function as $||(\mathbf{M}-\mathbf{BC})\mathbf{w}||_2$, where $\mathbf{w}$ is a constant weight vector, instead of $||\mathbf{M}-\mathbf{BC}||_2$ as the standard hamming distance cost function. Here, if the numerical difference is the objective QoR, then $\mathbf{w}$ will be defined to introduce bit significances based on powers-of-two (e.g., 8, 4, 2, 1); therefore, giving different numerical weights for different bit positions. In our experiments, we modify the ASSO [36] algorithm as to penalize mismatches on higher bit indices more than lower significant bits. We will provide experimental results showcasing the benefits of such weighting schemes in contrast to uniform weights (Hamming distance) in Section V.

B. Partitioning and Design Space Exploration

Since the truth table size of a circuit grows exponentially with the number of its inputs, we break down any large circuit into sub-circuits, where each sub-circuit has a limited number of inputs (e.g., $n \leq 10$) and then approximate each sub-circuit individually using the proposed binary matrix decomposition method with mixed OR/XOR decompressor implementation.

As our methodology operates on the truth table of the input circuit, the size of the input matrix, i.e. the number of rows, grows exponentially as the number of primary inputs increases. Furthermore, BMF is a NP-hard problem, and the existing methodologies are based on heuristics [33], [35], [36]. Therefore, the applicability of our method can be limited as the complexity of the circuit increases. Therefore, we propose a circuit decomposition technique to scale the BMF algorithm for larger circuits. The overall idea of our method is to first partition a large circuit into a number of subcircuits, such that each subcircuit has a maximum of k inputs as illustrated in Figure 4.a and then each of the subcircuits is approximated as shown in Figure 4.b. The values for k and m, the number of outputs, are determined based on the afforded runtime of the factorization algorithm.

To limit the number of inputs and outputs in subcircuits, we propose to use hypergraph partitioning algorithm [3] recursively until all subcircuits have a maximum of k inputs and maximum of m outputs. Also, we will discuss the relation between size of subcircuits and performance of approximation in section V.

Dividing a large circuit into smaller subcircuits of size $k \times m$ requires a change to the way we compute the QoR. More specifically, we can no longer evaluate the accuracy of a subcircuit in isolation, as errors

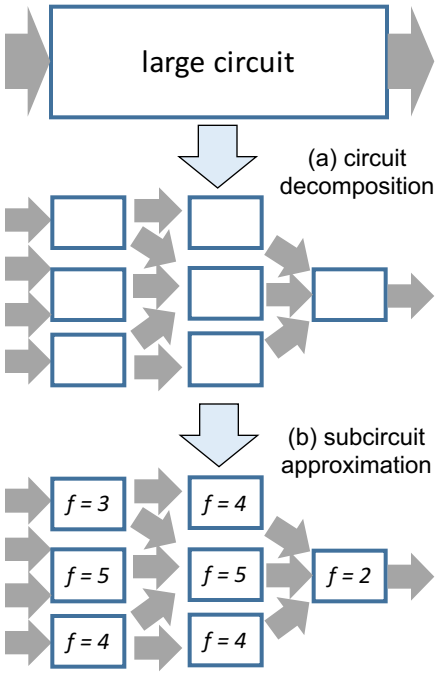


Fig. 4. Illustrated methodology for partitioning circuits.

in one component can propagate through the circuit leading to larger errors in the final outputs. Therefore, in our work instead of evaluating the QoR of a subcircuit individually, we evaluate the QoR of the entire approximate circuit, denoted by $Cir(s_i \rightarrow T_{s_i,f_i})$, where an accurate subcircuit, s_i, is substituted by its approximate version, T_{s_i,f_i}, with a factorization degree of f_i.

1) Greedy Heuristic DSE

Our design space exploration algorithm starts by identifying the sub-circuits; we calculate the possible approximate realizations for each sub-circuit using various factorization degrees, OR/XOR implementations. We then greedily explore the space of generated approximate sub-circuits to identify a good approximation order. We assess the QoR as measured by a user-defined error metric for each of its approximate realization by substituting the original subcircuit by its approximate realization and evaluating the outcome using the primary outputs of the circuit. The sub-circuit that leads to the smallest value of loss function is then chosen, and its approximated realization is then substituted in the main circuit. This sub-circuit approximation process is repeated until the maximum target error is reached.

Since a large input circuit will have multiple subcircuits, the order and the degree to which the approximations are introduced to the circuit has to be carefully analyzed. We devise Algorithm 1 to gradually approximate the circuit. In our algorithm, first, the circuit is partitioned into smaller subcircuits (line 1). In the next stage (lines 3-9) and for each subcircuit, the set of *potential* approximate versions under various approximation degrees are profiled. Next, starting from the accurate design, approximations are gradually added to the input design by exploring the neighbors of the current design (lines 14-24). Here, neighbors of a given design are defined as top-level circuits for which the degrees of approximation only reduce by *one* in *one* subcircuit. Here in lines 16-20, each neighbor is synthesized, where its QoR metric and chip area are assessed. The subcircuit with the least loss value, defined in line 18, is then chosen to replace the current circuit for next iteration in lines 21-23. The process is repeated iteratively until the QoR gets higher than a predefined threshold. The output approximation Cir is the one with smallest chip area in explored design space.

Algorithm 1: BLASYS: Boolean Level Approximate Circuit Synthesis

Input : Accurate Circuit $ACir$, Error Threshold
Output: Approximate Circuit Cir

subcircuits=Decompose input circuit $ACir$ by using k-way hypergraph partitioning recursively
// Factorization profiling Phase
for *each subcircuit* s_i *with* $m_i \leq m$ *outputs* **do**
    $\mathbf{M}$=Construct truth table of s_i
    // profile for every possible factorization degree
    for f=*1 to* m_i*-1* **do**
        $[\mathbf{B}, \mathbf{C}]$ = BMF($\mathbf{M}$, f)
        $T_{s_i,f}$=Construct truth table of $\mathbf{BC}$
    end
end
// Circuit Space Exploration Phase
Cir=ACir;
ExploredSpace=Empty List;
Let $f_i = m_i$ for all subcircuits s_i
while $QoR(Cir) \leq$ *threshold* $+\epsilon$ **do**
    for *each subcircuit* s_i *with* $f_i > 1$ **do**
        Cir'=$Cir(s_i \rightarrow T_{s_i,f_i-1})$
        $loss_i$ = $(area(Cir') - area(ACir))$ / $QoR(Cir')$
        Add Cir' into ExploredSpace
    end
    $b = \arg\min_i(loss_i)$
    $Cir = Cir(s_b \rightarrow T_{s_b,f_b-1})$
    $f_b = f_b - 1$
end
Cir=Best design in ExploredSpace
return Cir

2) Loss function

In Algorithm 1, our goal is to reduce design area and power consumption as much as possible with a fixed error threshold. We choose design area as an estimation of approximation degree, and propose the following loss metric to greedily explore the design space. Assuming we denote design area of accurate circuit by $area(ACir)$, the approximate circuit by $area(Cir_i)$, and degradation in QoR by $QoR(Cir_i)$, the loss is defined as

$$L_i = \frac{area(Cir_i) - area(ACir)}{QoR(Cir_i)} \tag{2}$$

For each iteration, we choose the neighbor with smallest loss to replace the current circuit. Recall that neighbors of a given design are defined as top-level circuits for which the degrees of approximation only reduce by *one* in *one* subcircuit. To minimize this loss metric, on one hand, a larger degradation in design area is preferable. On the other hand, since the loss value is negative, a smaller degradation in QoR is also preferable in order to minimize the loss. The intuition of the loss function is that, the design space of approximate circuit is expected to reduce sharply, while the design accuracy should remain relatively high. Thus, we balance the trade-off between reduction in design area and QoR. Although design area and power consumption are not strictly proportional to each other, design area is a better representative of circuit complexity, and able to reflect the changes in other metrics in general.

This loss metric performs even better with output weights scheme. Since different outputs could have different weights in QoR estimation, our loss metric will first explore design space which approximates less significant output bits, and then gradually move to more significant ones.

The loss function may be further modified in a stepwise manner. In each iteration, we first choose from designs with very small degradation of QoR (*e.g.* 0.01%). If there is no better design in this range,

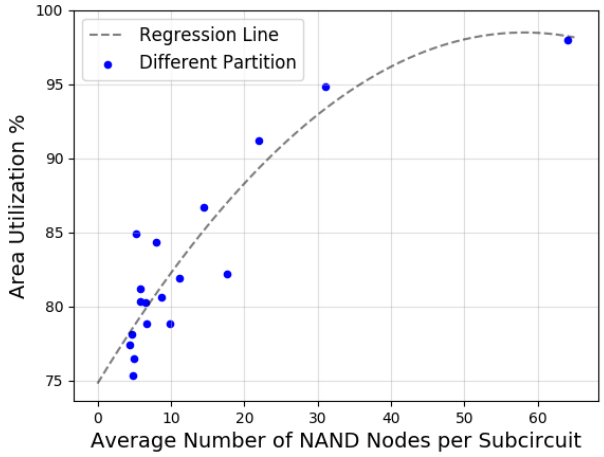


Fig. 5. Relationship between average size of subcircuits and design area, for 0.1% Mean Absolute Error on 7-bit unsigned multiplier.

we then gradually increase the range of QoR degradation. The reason for this stepwise loss metric is to prevent design accuracy from dropping rapidly.

C. Hyperparameters

Besides Algorithm 1, we also introduce a few hyperparameters, in order to control the range of explored design space and balance trade-off between runtime complexity and approximate performance.

1) Step size

In Algorithm 1, $Cir' = Cir(s_i \rightarrow T_{s_i,f_i-1})$ (line 17) means that the factorization degree for one subcircuit is decreased only by one. In practice, in order to factorize truth table efficiently, large input circuits might be partitioned into hundreds of subcircuits. To speed up Algorithm 1, we are able to set a larger integer as step size. With a larger value, each approximation realization will take a larger step, meaning that there will be a more significant reduction in design area and QoR. In this case, the algorithm will converge more quickly with a set error threshold. On the other hand, larger step size will ignore many approximate design in-between, lead to smaller exploration space.

2) Size of subcircuits

The first step of Algorithm 1 is to break down input circuit in subcircuits, whose number of inputs and outputs is limited. Algorithm 1 calls k-way hypergraph partitioning recursively, and may further break down subcircuits to smaller ones, which introduces more subcircuits. Figure 5 demonstrates relationship between average size of subcircuits, which is assessed by average number of NAND gates, and the area of output circuit. We test on 7-bit unsigned multiplier with 0.1% error threshold. Generally, when average size of subcircuits is smaller, which means input circuit is partitioned into more pieces of subcircuits, the approximate circuit has smaller design area. On one hand, our algorithm relies on synthesis capacity. Smaller subcircuit corresponds to smaller truth table, which then leads to smaller truth tables of *compressor* and *decompressor*. In practice, it is easier to optimize synthesis result with a smaller truth table. On the other hand, with smaller subcircuits, each of them represents less information in terms of top-level design, and each step of approximation leads to a slower degradation in QoR. With a fixed error threshold, we are able to explore more designs with smaller subcircuits, which is more likely to end up with a better approximate design.

However, with smaller subcircuits, the algorithm take longer to converge to the error threshold. As Algorithm 1 (line 16) suggests, for each iteration, it will evaluate n designs, where n is the number of

subcircuits. In practice, having more subcircuits is more likely to improve approximation results, but will dramatically increase runtime.

3) Multi-path exploration

In Algorithm 1 (line 22), the current design is substituted by the best approximation realization in each iteration based on the loss metric. In order to expand explored design space for global optimum, we also propose a multi-path version of greedy DSE. Instead of only choosing the best approximation realization, the first b best design are chosen as current designs and explored in each iteration. Specifically, all neighbors of b designs are assessed. Then again, among all neighbors, best b designs are chosen to substitute original b designs as starting point of next iteration. Multi-path exploration has a larger explored design space, which is roughly b times than before, and thus often leads to a better design with the same error threshold.

V Experimental Results

In this section, we discuss our experimental results and highlight the benefits offered by the proposed methodology. For hardware metrics, all designs are implemented in Verilog and synthesized using ABC logic synthesis tool [40] using an industrial 65 nm technology node at the typical processing corner. We evaluate combinational benchmarks available in ISCAS [5] and part of EPFL arithmetic benchmark suite [4]. For smaller benchmarks, we generate the truth table and directly pass the truth table to the factorization algorithm. For the larger ones, however, we first decompose the circuit as described in Subsection IV-B. Furthermore, we compare approximate designs from our algorithm against EvoApproxLib, a library of approximate arithmetic circuits, to demonstrate that our algorithm is able to reach state-of-the-art performance.

For design accuracy, we report the normalized Hamming distance (HD), which is defined as

$$\text{Normalized HD} = \frac{|\mathbf{A} - \mathbf{BC}|}{Nm}, \tag{3}$$

and mean absolute error (MAE) defined as

$$\text{MAE} = \frac{1}{N}\Sigma_{i=1}^{N}\frac{|R_i - R_i'|}{2^m}, \tag{4}$$

for logical and binary numerical outputs, respectively. Here, N represents the size of the test vectors while R_i and R_i', represent the accurate and approximate numerical results. m is the number of primary outputs. Furthermore, for smaller circuits, we define the accuracy over all possible inputs, while for larger networks, we estimate standard deviation of QoR with different number of test vectors, and choose a proper size as discussed in the first subsection.

A. Work Flow

In this subsection, we briefly describe the work flow of our methodology. Figure 6 demonstrates various tools in BLASYS tool-chain, which is used for all following experiments [41].

To begin with, Yosys [42] parses the input exact circuit and assesses its chip area with a given liberty file, which in our case, is an industrial 65 nm technology node. Using the provided set of test vectors, Icarus Verilog [43] simulates the input exact circuit, which is then used for QoR estimation.

Next, LSOracle [44] is used to partition the input circuit to multiple subcircuits, each of which has a similar size. Considering runtime efficiency of Boolean matrix factorization, our methodology partitions an input circuit until all subcircuits have less than 10 inputs and 10 outputs. Then a set of test vectors is generated for each subcircuit. We use the ASSO algorithm [36] to factor each truth table based on a vector called *f-stream*, which consists of factorization degree for each subcircuit. This vector is

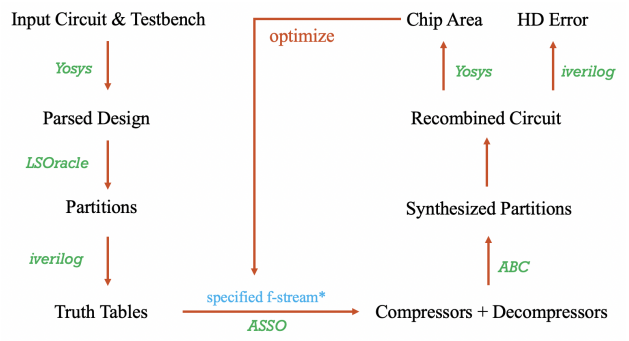


Fig. 6. Structure of BLASYS Tool-chain.

determined by the design space exploration method as discussed in Section IV-B. As a result, each truth table is factorized into a compressor and decompressor. We use ABC [40] to synthesize the compressor matrix to a circuit and uses a network of logic OR or XOR to represent decompressor, depending on heuristic search of XOR/OR-based approach. Thus, an approximated version of the input circuit can be obtained by recombining all approximated subcircuits. Afterwards, we use Yosys to estimate the chip area of the approximate circuit and executes a simulation using the input set of test vectors. From the original and approximated simulation results, QoR can be defined arbitrarily based on the functionality of input circuit. In our experiments, we consider the Normalized Hamming Distance error (HD) and Mean Absolute Error (MAE). The area reduction ratio and QoR are used to optimize f-stream iteratively as mentioned in Algorithm 1.

The implementation of work flow is available at http://github.com/scale-lab/blasys.

B. Number of test vectors

Before experimenting our methodology with various benchmarks, we need to create testbench for each benchmark. Since most benchmarks have large number of inputs, it is impossible to enumerate all possible combination of test vectors. Therefore, for each benchmark, we generate a set of distinct random test vectors of size s. To find out proper size for each benchmark, we evaluate standard deviation of Normalized Hamming Distance with different sizes of test vectors. Specifically, for one benchmark, we generate 200 random sets of test vectors respectively, from size 100 to 10,000 for every 100, and assess standard deviation of Normalized Hamming Distance for each size. Figure 7 illustrates relationship between number of test vectors and standard deviation of Normalized Hamming Distance in Max circuit of EPFL benchmarks. After reaching 0.1%, reduction of standard deviation becomes slower and standard deviation begins to converge. Considering runtime efficiency of our algorithm, the number of test vectors cannot be arbitrary large. Therefore, sizes with 0.1% standard deviation is reasonable in terms of both accuracy and efficiency. Table I demonstrates the number of test vectors required to achieve below 0.1% and 0.2% standard deviation of Normalized Hamming Distance in EPFL arithmetic benchmarks.

C. BMF-based Approximate Logic Synthesis

1) Semi-Ring vs Field Algebra

In this section, we compare approximate results among different boolean matrix factorization algebra. As Section IV-A1 mentions, semi-ring boolean algebra is implemented by ASSO algorithm, which is also referred to as OR-based. In order to implement field modulo-2 algebra (XOR-based), we perform an exhaustive search over the results of semi-ring algebra. Specifically, for $\mathbf{A} \approx \mathbf{BC}$, we fix $\mathbf{B}$ and greedily replace columns in $\mathbf{C}$ with field modulo-2 algebra. Moreover, we mix OR-based and XOR-based method and derive XOR/OR-based method. After computing OR-based $\mathbf{A} \approx \mathbf{BC}$ by ASSO algorithm, we fix $\mathbf{B}$, and for each column in $\mathbf{C}$, we do an exhaustive search with both semi-ring algebra and field modulo-2

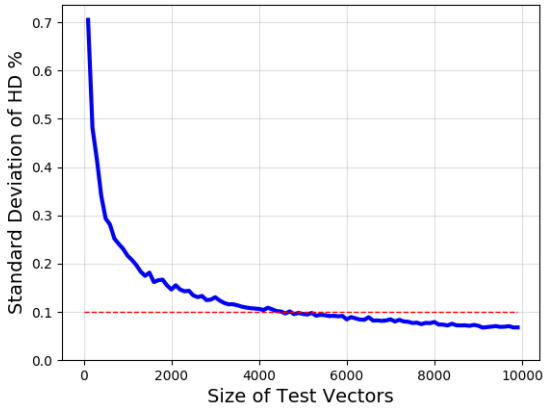


Fig. 7. Relationship between number of test vectors and standard deviation in Normalized Hamming Distance in benchmark Max.

TABLE I
SIZE OF TEST VECTORS REQUIRED TO ACHIEVE 0.1% AND 0.2% STANDARD DEVIATION OF HAMMING DISTANCE.

Name	I/O	Area (um^2)	0.2% σ Size	0.1% σ Size
Adder	256/129	1743.48	700	2100
Barrel shifter	135/128	4878.00	600	2100
Max	512/130	4320.00	1500	4600
Multiplier	128/128	37799.28	500	2000
Sine	24/25	8308.44	2300	9400
Square	64/128	25733.16	5400	-

algebra. Then the one which leads to smallest QoR degradation is chosen. In this case, the decompressor circuit uses both OR and XOR gates. We evaluate OR-based, XOR-based and XOR/OR-based method on x2 benchmark in LGSynth 91. Since x2 is a small benchmark, we generate the truth table and directly pass the truth table to the factorization algorithm without partitioning. Figure 8 demonstrates the approximate results from three methods. x2 benchmark has 7 output bits. Therefore, each method derives 6 approximate designs, ranging from approximation degree 1 to 6. According to Figure 8, with XOR-based and XOR/OR-based method, we make huge improvement in terms of area saving with similar Hamming distance error. And in most case, XOR/OR-based method has best performance. With 5.47% Hamming distance error, XOR/OR-based method can save 14.00% design area. For designs with higher error, XOR/OR-based method can save 34.11% design area with 10.74% Hamming distance error, which significantly outperforms other two methods.

2) Output Weight Schemes

As Section IV-A2 mentions, considering that significance of output bits may be different, output weights in BMF algorithm sometimes improve approximate results. For example, with arithmetic circuit which outputs binary numbers, bit flips in least significant bit and a more significant bit have different impact on QoR. Therefore, for unsigned arithmetic circuits, we introduce output weight into ASSO algorithm, where n^{th} output bit has weight 2^{n-1}. We approximate 8-bit unsigned adder with both unweighted and weighted BMF algorithm. To eliminate the interference of exhaustive search in XOR-based method and highlight the benefit of using output weights, we only use OR-based method in this section. Figure 9 demonstrates the necessity of using output weights. Since outputs of adder are numerical results, we use mean absolute error (MAE) as QoR metric. As Figure 9 shows, output weight scheme provides decent

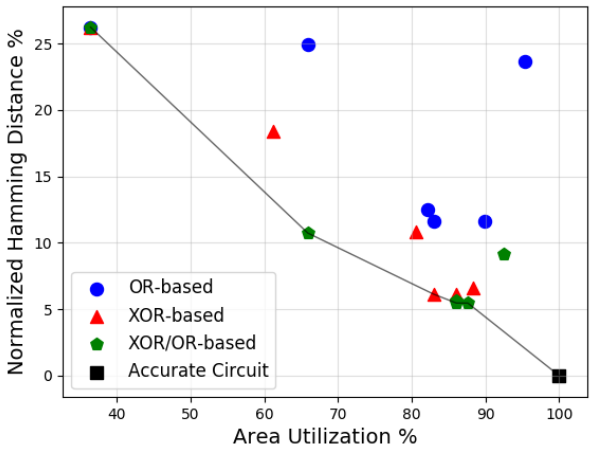


Fig. 8. Difference between OR-based, XOR-based and XOR/OR-based method on x2 benchmark.

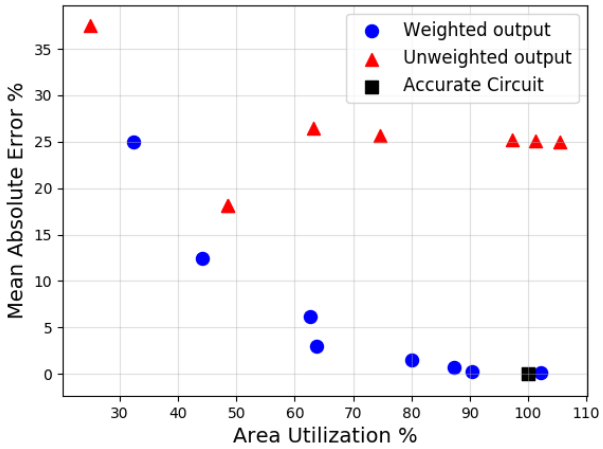


Fig. 9. Benefit offered by output weight scheme on 8-bit unsigned adder.

approximate results with good QoR performance, while approximate designs from unweighted scheme have much higher mean absolute error, which are all above 18%. If no output weight is provided, BMF algorithm will factorize truth table while minimizing number of total flipped bits. However, the algorithm does not consider bit significance. Therefore, although more bits in truth table are accurate, more significant bits might be flipped, which leads to much higher mean absolute error.

D. Design Space Exploration

In previous subsection, we approximate small benchmarks x2 and 8-bit unsigned adder by directly passing the truth table to the factorization algorithm without partitioning. As Section IV-B mentioned, the size of truth table grows exponentially with the number of primary inputs. In order to approximate larger circuit, we first partition input circuit into subcircuits with maximum 10 inputs and 10 outputs, generate truth table for each subcircuits, and perform BMF on each truth table of subcircuit. Since output bit significance within each subcircuit is hard to analyze, when approximating each subcircuit, we use XOR/OR-based method and the target QoR metric to evaluate the simulation results that guide the design

TABLE II
ISCAS '85 BENCHMARKS EVALUATED USING THE PROPOSED METHODOLOGY WITH NORMALIZED HAMMING DISTANCE.

Name	Original			5% Error Metric			10% Error Metric			15% Error Metric		
	Area (um^2)	Power (uW)	Delay (ns)	Area %	Power %	Delay %	Area %	Power %	Delay %	Area %	Power %	Delay
c1355	457.92	64.20	0.81	6.8	6.8	1.7	6.1	6.1	1.7	5.4	5.4	1.7
c17	-	-	-	-	-	-	-	-	-	-	-	-
c1908	339.84	52.90	1.25	39.3	37.8	28.8	23.9	22.9	23.2	20.0	19.9	23.7
c2670	625.68	219.00	1.16	36.0	28.5	65.4	24.3	19.0	50.6	15.2	13.2	34.5
c3450	959.76	222.00	1.75	60.7	71.6	93.1	56.2	67.1	88.8	50.3	64.4	90.7
c432	152.64	38.60	1.62	85.6	77.7	85.6	83.0	75.1	86.3	71.5	53.6	67.4
c499	460.80	91.50	0.88	47.0	39.8	99.3	21.3	21.9	57.9	19.1	18.7	57.8
c5315	1543.68	487.00	1.31	59.3	58.3	77.2	36.3	32.9	72.1	21.3	18.9	60.7
c6288	3066.84	264.00	4.39	96.3	83.3	91.4	93.7	92.8	92.0	90.3	110.2	97.3
c880	362.16	75.90	1.34	56.6	50.0	53.0	34.5	29.0	32.8	14.3	11.3	25.0
Average				54.2	50.4	66.2	42.1	40.8	56.2	34.2	35.1	51.0

TABLE III
EPFL ARITHMETIC BENCHMARKS EVALUATED USING THE PROPOSED METHODOLOGY WITH NORMALIZED HAMMING DISTANCE.

Name	Original			5% Error Metric			10% Error Metric		
	Area (um^2)	Power (uW)	Delay (ns)	Area %	Power %	Delay %	Area %	Power %	Delay %
Adder	1325.16	59.40	11.56	89.4	94.8	90.8	79.4	84.0	80.9
Barrel shifter	2828.88	1270.00	2.69	95.8	79.5	105.6	90.0	64.7	88.5
Max	3131.28	851.00	13.45	91.0	65.5	114.3	77.6	58.0	94.3
Multiplier	30417.48	1230.00	12.24	87.7	78.4	99.4	80.5	57.6	93.8
Sine	6608.16	754.00	10.08	84.3	81.2	93.1	71.7	65.2	79.9
Square	24736.32	876.00	9.48	95.8	93.6	85.8	88.5	80.7	75.5
Average				90.7	82.2	98.2	81.3	68.4	85.5

space exploration. In this section, we demonstrate the approximate result with design space exploration on ISCAS and EPFL benchmarks. We also compare our approximate results against EvoApproxLib, which is a well-established library of adders and multipliers.

Table II demonstrates approximate designs of ISCAS '85 benchmarks. Since these benchmarks are not arithmetic circuits, we use Normalized Hamming Distance as QoR metric. For each benchmark, we set 3 error thresholds, which are 5%, 10% and 15%, and evaluate best approximate designs for them. Since c17 benchmark only has 2 primary outputs, our algorithm has only 1 factorization degree, where Hamming distance is above 15%. Within 5% hammming distance error, on average the area utilization is 54.18% and power consumption is 50.44% of original. Within 10% Hamming distance error, the area utilization drops to 42.14% and power utilization is 40.75%. Therefore, our algorithm shows remarkable saving of area and power on ISCAS '85 benchmark.

Furthermore, Table III summarizes approximate designs of EPFL arithmetic benchmarks. This benchmark suite has 10 circuits, which have larger chip areas than ISCAS '85. Due to computational capacity, we test our algorithm on 6 benchmarks. Since EPFL benchmark suite does not provide bit numbering of outputs, we use normalized Hamming distance as QoR metric. For each benchmark, we set two error thresholds for approximate design, which are 5% and 10%. Within 5% Hamming distance error, the area utilization drops to 90.7% and power utilization drops to 82.2%.

Finally, we test our method on four commonly used arithmetic circuits and compare results against EvoApproxLib, which provides approximate designs for adders and multipliers. Since circuits in EvoApproxLib are synthesized from a different standard cell library, we first synthesize their approximate

TABLE IV
COMPARISON BETWEEN EVOAPPROXLIB AND BLASYS ON 7-BIT UNSIGNED MULTIPLIER

EvoApproxLib			BLASYS		
QoR	Area (um^2)	Power (uW)	QoR	Area (um^2)	Power (uW)
0.0299%	448.20	82.40	0.0290%	445.68	74.40
0.0515%	417.60	79.00	0.0488%	421.92	75.00
0.1400%	351.72	63.10	0.1337%	356.40	61.70
0.2428%	272.16	44.80	0.2369%	317.16	62.40
0.4583%	225.36	41.80	0.4532%	252.72	39.00
1.1330%	133.20	22.30	1.1203%	125.28	19.70
2.2738%	80.64	14.00	2.2298%	69.12	12.30
5.0938%	30.96	4.26	4.4771%	30.96	4.36

TABLE V
COMPARISON BETWEEN EVOAPPROXLIB AND BLASYS ON 8-BIT UNSIGNED MULTIPLIER

EvoApproxLib			BLASYS		
QoR	Area (um^2)	Power (uW)	QoR	Area (um^2)	Power (uW)
0.0002%	682.92	120.00	-	-	-
0.0014%	666.72	113.00	0.0011%	640.08	92.10
0.0076%	612.00	106.00	0.0069%	622.44	92.80
0.0370%	522.00	88.20	0.0346%	534.96	72.60
0.1812%	358.56	47.40	0.1757%	413.64	54.60
0.8859%	170.64	24.10	0.7973%	239.76	31.60
4.8338%	26.28	3.42	4.4782%	52.56	5.13

designs with the same industrial 65 nm technology node. Then we use our algorithm to generate designs using their QoR metrics as thresholds, and compare area and power utilizations. Since outputs represent numerical value, we use mean absolute error (MAE) as QoR metric. Table IV to VII compare approximate designs between EvoApproxLib and BLASYS on 7-bit unsigned multiplier, 8-bit unsigned multiplier, 16-bit unsigned multiplier and 16-bit unsigned adder respectively. As a unified approach that generates approximate designs for general circuit, our algorithm outperforms EvoApproxLib in terms of power consumption. Among 24 designs of unsigned multipliers, our algorithm has better power utilization in 17 designs. Although we only beat 6 designs in terms of area utilization, the numbers are close in other designs while ours have better QoR. Figure 10 illustrates the explored design space of our algorithm compared to designs of EvoApproxLib, where blue points are designs from our algorithm, and red points are designs from EvoApproxLib. It shows that our algorithm is competitive in terms of area utilization, and outperforms EvoApproxLib in terms of power utilization. Therefore, our algorithm is able to reach state-of-the-art performance in many commonly used circuits. Table VII shows that our algorithm has worse results in 16-bit unsigned adder. Since it is a relatively small design, it has less number of subcircuits, which leads to a small explored design space. In this case, lack of design space exploration might sometimes affect performance.

E. Runtime Characterization

In this subsection, we briefly discuss the improvement of runtime. As mentioned in Section IV-A1, the time complexity of exhaustive search for XOR/OR-based method is $O(m2^n)$, where m is the number of output bits, and n is the number of input bits. In order to speed up this process, we break down input circuits into subcircuits with maximum 10 inputs and 10 outputs. Also, in practice, to exhaustively search

TABLE VI
COMPARISON BETWEEN EVOAPPROXLIB AND BLASYS ON 16-BIT UNSIGNED MULTIPLIER

EvoApproxLib			BLASYS		
QoR	Area (um^2)	Power (uW)	QoR	Area (um^2)	Power (uW)
3e-10	3056.40	287.00	0.00%	3038.76	265.00
5.7e-09	2900.88	275.00	5.1e-09	2925.72	251.00
4.5e-08	2665.80	246.00	3.4e-08	2702.16	242.00
7.5e-07	2291.76	218.00	7.5e-07	2322.36	215.00
7.3e-06	1735.92	170.00	7.3e-06	1864.80	155.00
0.0110%	1182.24	121.00	0.0110%	1100.16	95.70
0.1000%	732.24	65.60	0.0958%	654.48	49.20
1.5400%	225.72	18.30	1.4824%	166.68	10.60
18.750%	2.16	0.09	-	-	-

TABLE VII
COMPARISON BETWEEN EVOAPPROXLIB AND BLASYS ON 16-BIT UNSIGNED ADDER

EvoApproxLib			BLASYS		
QoR	Area (um^2)	Power (uW)	QoR %	Area (um^2)	Power (uW)
0.0002%	167.40	56.50	-	-	-
0.0018%	134.28	46.40	0.0015%	164.16	51.60
0.0063%	119.52	41.10	0.0034%	145.44	49.10
0.0210%	101.16	34.60	0.0147%	138.60	44.10
0.0570%	85.68	27.60	0.0289%	119.88	35.80
0.2000%	63.72	20.10	0.1996%	92.88	24.90
0.9100%	42.12	13.10	0.7864%	77.40	17.50
3.5200%	24.84	6.42	3.2475%	60.84	11.20
9.9000%	9.72	1.84	-	-	-

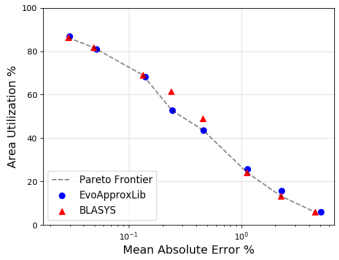
columns in truth table of decompressor, we compute all possible combinations of columns at first, and then choose the best for each column.

Then, instead of approximating all subcircuits at once as Algorithm 1 suggests, in practice we approximate subcircuits on-demand. The approximation realizations of subcircuits are stored and can be reused later for other designs. With a multi-core system, we are able to parallel evaluation of designs in each iteration, since the degrees of approximation is reduced by step size in different subcircuits.

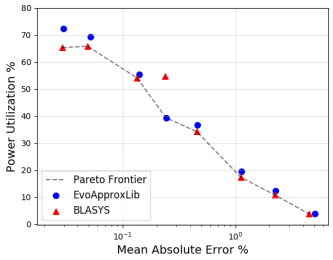
After implementing improvements mentioned above, our method is speed up by 35%. Figure 11 illustrates the distribution of runtime. Due to on-demand approximation of subcircuits and reusing, subcircuit approximating only takes 0.2% of runtime. Simluation, which is QoR estimation of approximate designs, takes 33.2% of runtime. And most of runtime is spent on synthesizing top-level designs from approximate subcircuits using Yosys.

VI Conclusions

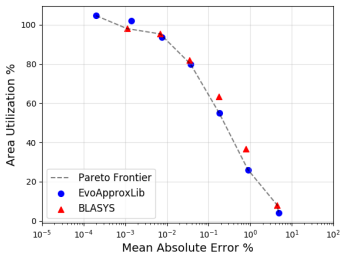
In this paper we proposed a new approach for approximate circuit synthesis by generalizing matrix factorization techniques to incorporate field (XOR) and semi-ring (OR) algebra implementations. This led to a wider range of possible approximate circuit realizations that can be explored to identify the best one. We integrated our approach into a design space exploration method with the capability to partition larger circuits into manageable sub-circuits for approximation. We implemented and evaluated our approach on a large range of circuits using a number of error metrics such as numerical differences



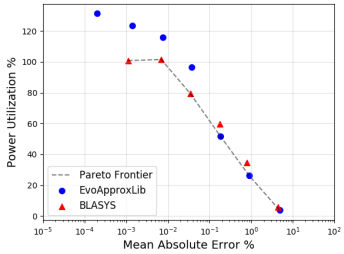
(a) Area utilization of 7-bit unsigned multiplier



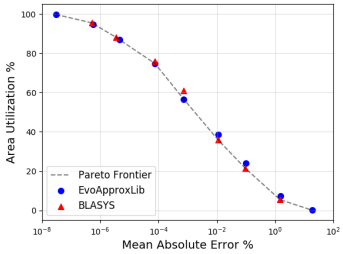
(b) Power utilization of 7-bit unsigned multiplier



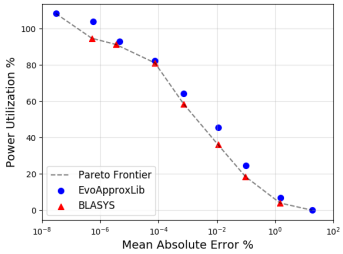
(c) Area utilization of 8-bit unsigned multiplier



(d) Power utilization of 8-bit unsigned multiplier



(e) Area utilization of 16-bit unsigned multiplier



(f) Power utilization of 16-bit unsigned multiplier

Fig. 10. Comparison between EvoApproxLib and BLASYS. Red points represent designs explored by BLASYS. Blue points represent designs provided by EvoApproxLib.

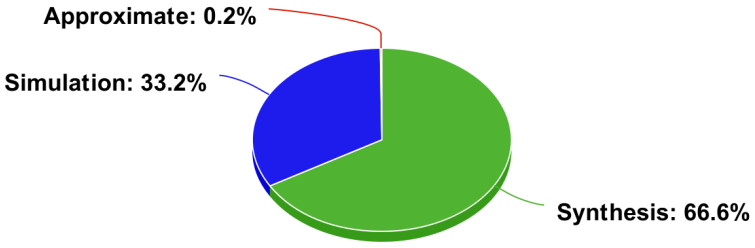


Fig. 11. Runtime distribution. Approximate corresponds to the time of approximating subcircuits. Synthesis corresponds to the time of synthesizing top-level design from sub-circuits. Simulation corresponds to QoR estimation.

and Hamming distances, and we have demonstrated that our method is able to reach state-of-the-art performance while being flexible for all kinds of input design. Furthermore, we elucidated the large space of possible approximate designs generated from our approach, and the trade-off between accuracy and design metrics such as power and area.

Acknowledgment

This work is partially supported by NSF grant 1814920 and DoD ARO grant W911NF-19-1-0484. We thank Dr. Soheil Hashemi for his work on this project.

References

[1] S. Hashemi, H. Tann, and S. Reda, "BLASYS: approximate logic synthesis using boolean matrix factorization," in *Design Automation Conference*, 2018, pp. 55:1–6.

[2] S. Hashemi and S. Reda, "Generalized matrix factorization techniques for approximate logic synthesis," in *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2019, pp. 1289–1292.

[3] S. Schlag, V. Henne, T. Heuer, H. Meyerhenke, P. Sanders, and C. Schulz, "k-way hypergraph partitioning via *n*-level recursive bisection," in *18th Workshop on Algorithm Engineering and Experiments, (ALENEX 2016)*, 2016, pp. 53–67.

[4] L. Amarú, P.-E. Gaillardon, and G. De Micheli, "The epfl combinational benchmark suite," in *Proceedings of the 24th International Workshop on Logic & Synthesis (IWLS)*, no. CONF, 2015.

[5] D. Bryan, "The iscas'85 benchmark circuits and netlist format," *North Carolina State University*, vol. 25, p. 39, 1985.

[6] Q. Xu, T. Mytkowicz, and N. S. Kim, "Approximate computing: A survey," *IEEE Design & Test*, vol. 33, no. 1, pp. 8–22, 2015.

[7] J. Han and M. Orshansky, "Approximate computing: An emerging paradigm for energy-efficient design," in *2013 18th IEEE European Test Symposium (ETS)*. IEEE, 2013, pp. 1–6.

[8] S. Sidiroglou-Douskos, S. Misailovic, H. Hoffmann, and M. Rinard, "Managing performance vs. accuracy trade-offs with loop perforation," in *Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering*, ser. ESEC/FSE '11. New York, NY, USA: ACM, 2011, pp. 124–134. [Online]. Available: http://doi.acm.org/10.1145/2025113.2025133

[9] S. Li, S. Park, and S. Mahlke, "Sculptor: Flexible approximation with selective dynamic loop perforation," in *Proceedings of the 2018 International Conference on Supercomputing*, ser. ICS '18. New York, NY, USA: ACM, 2018, pp. 341–351. [Online]. Available: http://doi.acm.org/10.1145/3205289.3205317

[10] A. Li, S. L. Song, M. Wijtvliet, A. Kumar, and H. Corporaal, "Sfu-driven transparent approximation acceleration on gpus," in *Proceedings of the 2016 International Conference on Supercomputing*, ser. ICS '16. New York, NY, USA: ACM, 2016, pp. 15:1–15:14. [Online]. Available: http://doi.acm.org/10.1145/2925426.2926255

[11] M. Samadi, J. Lee, D. A. Jamshidi, A. Hormati, and S. Mahlke, "Sage: Self-tuning approximation for graphics engines," in *2013 46th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, Dec 2013, pp. 13–24.

[12] S. Campanoni, G. Holloway, G.-Y. Wei, and D. Brooks, "Helix-up: Relaxing program semantics to unleash parallelization," in *Proceedings of the 13th Annual IEEE/ACM International Symposium on Code Generation and Optimization*, ser. CGO '15. Washington, DC, USA: IEEE Computer Society, 2015, pp. 235–245. [Online]. Available: http://dl.acm.org/citation.cfm?id=2738600.2738630

[13] H. Esmaeilzadeh, A. Sampson, L. Ceze, and D. Burger, "Architecture support for disciplined approximate programming," *SIGPLAN Not.*, vol. 47, no. 4, pp. 301–312, Mar. 2012. [Online]. Available: http://doi.acm.org/10.1145/2248487.2151008

[14] S. Liu, K. Pattabiraman, T. Moscibroda, and B. G. Zorn, "Flikker: Saving dram refresh-power through critical data partitioning," *SIGPLAN Not.*, vol. 46, no. 3, pp. 213–224, Mar. 2011. [Online]. Available: http://doi.acm.org/10.1145/1961296.1950391

[15] P. V. Rengasamy, A. Sivasubramaniam, M. T. Kandemir, and C. R. Das, "Exploiting staleness for approximating loads on cmps," in *2015 International Conference on Parallel Architecture and Compilation (PACT)*, Oct 2015, pp. 343–354.

[16] G. Karakonstantis and K. Roy, "Voltage over-scaling: A cross-layer design perspective for energy efficient systems," in *2011 20th European Conference on Circuit Theory and Design (ECCTD)*. IEEE, 2011, pp. 548–551.

[17] A. B. Kahng and S. Kang, "Accuracy-configurable adder for approximate arithmetic designs," in *Proceedings of the 49th Annual Design Automation Conference*, 2012, pp. 820–825.

[18] S. Hashemi, R. I. Bahar, and S. Reda, "Drum: A dynamic range unbiased multiplier for approximate applications," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, ser. ICCAD '15. Piscataway, NJ, USA: IEEE Press, 2015, pp. 418–425.

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

[20] S. Hashemi, R. I. Bahar, and S. Reda, "A low-power dynamic divider for approximate applications," in *2016 53nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, June 2016, pp. 1–6.

[21] K. Nepal, S. Hashemi, H. Tann, R. I. Bahar, and S. Reda, "Automated high-level generation of low-power approximate computing circuits," *IEEE Transactions on Emerging Topics in Computing*, pp. 1–13, 2016.

[22] S. Venkataramani, A. Sabne, V. Kozhikkottu, K. Roy, and A. Raghunathan, "Salsa: Systematic logic synthesis of approximate circuits," in *DAC Design Automation Conference 2012*, June 2012, pp. 796–801.

[23] A. Ranjan, A. Raha, S. Venkataramani, K. Roy, and A. Raghunathan, "Aslan: Synthesis of approximate sequential circuits," in *Design, Automation & Test in Europe Conference*, 2014, pp. 1–6.

[24] S. Venkataramani, K. Roy, and A. Raghunathan, "Substitute-and-simplify: A unified design paradigm for approximate and quality configurable circuits," in *Design, Automation and Test in Europe*, 2013, pp. 1367–1372.

[25] J. Miao, A. Gerstlauer, and M. Orshansky, "Approximate logic synthesis under general error magnitude and frequency constraints," in *Proceedings of the International Conference on Computer-Aided Design*, 2013, pp. 779–786.

[26] Z. Vasicek and L. Sekanina, "Evolutionary design of complex approximate combinational circuits," *Genetic Programming and Evolvable Machines*, vol. 17, no. 2, pp. 169–192, Jun 2016.

[27] S. Frohlich, D. Grobe, and R. Drechsler, "Error Bounded Exact BDD Minimization in Approximate Computing," in *International Symposium on Multiple-Valued Logic*, 2017, pp. 254–259.
[28] ——, "Approximate hardware generation using symbolic computer algebra employing grobner basis," in *Design, Automation and Test in Europe*, 2018, pp. 889–892.
[29] S. Lee, L. K. John, and A. Gerstaluer, "High-level synthesis of approximate hardware under joint precision and voltage scaling," in *Design, Automation and Test in Europe*, 2017.
[30] S. Venkataramani, A. Ranjan, K. Roy, and A. Raghunathan, "Axnn: energy-efficient neuromorphic systems using approximate computing," in *2014 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*. IEEE, 2014, pp. 27–32.
[31] S. Hashemi, H. Tann, F. Buttafuoco, and S. Reda, "Approximate computing for biometric security systems: A case study on iris scanning," in *2018 Design, Automation Test in Europe Conference Exhibition (DATE)*, March 2018, pp. 319–324.
[32] A. Raha and V. Raghunathan, "Towards full-system energy-accuracy tradeoffs: A case study of an approximate smart camera system*," in *2017 54th ACM/EDAC/IEEE Design Automation Conference (DAC)*, June 2017, pp. 1–6.
[33] D. D. Lee and H. S. Seung, "Learning the parts of objects by non-negative matrix factorization," *Nature*, vol. 401, pp. 788–791, 1999.
[34] W. Xu, X. Liu, and Y. Gong, "Document clustering based on non-negative matrix factorization," in *Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval*, 2003, pp. 267–273.
[35] P. Miettinen and J. Vreeken, "Model order selection for boolean matrix factorization," in *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2011, pp. 51–59.
[36] ——, "Mdl4bmf: Minimum description length for boolean matrix factorization," *ACM Transactions on Knowledge Discovery from Data*, vol. 8, no. 4, pp. 18:1–31, 2014.
[37] P. Miettinen, T. Mielikäinen, A. Gionis, G. Das, and H. Mannila, "The discrete basis problem," *IEEE transactions on knowledge and data engineering*, vol. 20, no. 10, pp. 1348–1362, 2008.
[38] Z. Zhang, T. Li, C. Ding, and X. Zhang, "Binary matrix factorization with applications," in *Seventh IEEE International Conference on Data Mining (ICDM 2007)*. IEEE, 2007, pp. 391–400.
[39] S. Ravanbakhsh, B. Póczos, and R. Greiner, "Boolean matrix factorization and noisy completion via message passing." in *ICML*, 2016, pp. 945–954.
[40] R. Brayton and A. Mishchenko, "ABC: An academic industrial-strength verification tool," in *International Conference on Computer Aided Verification*. Springer, 2010, pp. 24–40.
[41] J. Ma, S. Hashemi, and S. Reda, "Approximate logic synthesis using blasys," 2019.
[42] C. Wolf, "Yosys open synthesis suite," http://www.clifford.at/yosys/, 2016.
[43] S. Williams, "Icarus verilog," http://iverilog.icarus.com/, 2006.
[44] W. L. Neto, M. Austin, S. Temple, L. Amaru, X. Tang, and P.-E. Gaillardon, "Lsoracle: a logic synthesis framework driven by artificial intelligence," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, November 2019.